Universidad Nacional de La Plata
Facultad de Informática

Medidas de Invarianza y Equivarianza a Transformaciones en Redes Neuronales Convolucionales. Aplicaciones al reconocimiento de formas de mano.

Facundo Manuel Quiroga

Directora
Dra. Laura Lanzarini

Tesis presentada para obtener el grado de Doctor en Ciencias Informáticas

Febrero de 2020

Resumen

Las Redes Neuronales son los modelos de aprendizaje automático con mejor desempeño en la actualidad en una gran variedad de problemas. Son modelos generales y aproximadores universales. Con algoritmos de optimización basados en descenso de gradiente, pueden optimizar miles o millones de parámetros en base a una función de error. Se distinguen de otros modelos en que no requieren un diseño manual de características de los datos para funcionar; las características se aprenden automáticamente mediante el proceso de optimización, también llamado entrenamiento. Su diseño se organiza en capas que determinan su arquitectura. En los últimos años, se ha conseguido entrenar Redes Neuronales con múltiples capas mediante un conjunto de técnicas que suelen denominarse Aprendizaje Profundo (Deep Learning).

En particular, las Redes Convolucionales, es decir, Redes Neuronales que utilizan capas convolucionales, son el estado del arte en la mayoría de los problemas de visión por computadora, incluyendo la clasificación de imágenes. Las capas convolucionales permiten aplicar convoluciones con filtros aprendidos para un mejor desempeño y eficiencia.

Muchos de los problemas para los cuales las Redes Convolucionales son el estado del arte requieren que los modelos se comporten de cierta manera ante transformaciones de su entrada. Existen dos propiedades fundamentales que capturan dicho requerimiento; la invarianza y la equivarianza. La invarianza nos dice que la salida del modelo no es afectado por las transformaciones. La equivarianza permite que la salida sea afectada, pero de una manera controlada y útil.

Si bien los modelos tradicionales de Redes Convolucionales son equivariantes a la traslación por diseño, no son ni invariantes a dicha transformación ni equivariantes a otras en los escenarios usuales de entrenamiento y uso. Existen dos opciones principales para otorgar invarianza o equivarianza a un modelo de red neuronal. La tradicional ha sido modificar el modelo para dotarlo de esas propiedades. La otra opción es entrenarlo con aumentación de datos utilizando como transformaciones el mismo conjunto al que se desea la invarianza o equivarianza.

Dotar con invarianza o equivarianza a los modelos tiene utilidades en varios dominios, como la clasificación de imágenes de galaxias, imágenes de microscopios o formas de mano. En particular, el reconocimiento de formas de mano en imágenes es una de las etapas más importantes de los sistemas de reconocimiento de lenguas de señas o gestos mediante imágenes o video. En muchos casos, la rotación, traslación o escalado de la mano en la imagen no afectan a su forma, y por ende se requiere dotar de invarianza a la red para mejorar el desempeño del sistema.

No obstante, no está claro cómo los modelos adquieren estas propiedades, tanto al usar aumentación de datos como al modificar el modelo. Tampoco está claro como las modificaciones de modelos afectan la eficiencia y el poder de representación de los mismos. Más aún, en los modelos tradicionales tampoco es conocido cómo se adquieren dichas propiedades con aumentación de datos, así como cuál es la mejor estrategia para aumentar los datos con este fin.

En el **primer aporte** de esta tesis, analizamos diversas estrategias para obtener invarianza o equivarianza en modelos de clasificación de imágenes con redes neuronales. Comparamos los modelos tradicionales AllConvolutional y LeNet, y los modelos especializados Group CNN y Spatial Tansformer Networks para determinar su desempeño. Realizamos experimentos con varios conjuntos de datos conocidos (MNIST y CIFAR10) utilizando aumentación de datos. Los resultados arrojan evidencia en favor de la hipótesis de que aún con ingeniosas modificaciones de las redes convolucionales, la aumentación de datos sigue siendo necesaria para obtener un desempeño similar al de los modelos no invariantes. Más aún, en varios casos la aumentación de datos por si sola puede proveer un desempeño similar al de los modelos especializados, siendo al mismo tiempo más simples de entrenar y comprender. Además, analizamos cómo re-entrenar una red previamente generada para convertirla en invariante, y encontramos que el entrenamiento de las últimas capas permite convertir un modelo no invariante en uno que si lo sea con un bajo costo computacional y leve pérdida de desempeño.

Si bien estos mecanismos permiten imbuir de invarianza o equivarianza una red, la forma en que la misma codifica o representa dichas propiedades no están claros. La comprensión de la invarianza o equivarianza de una red o cualquier sistema puede ayudar a mejorar su desempeño y robustez. Estas propiedades pueden estimarse midiendo los cambios en las salidas de la red en base a las transformaciones realizadas a su entrada. Las metodologías actuales de evaluación y comprensión de la invarianza y equivarianza se enfocan solamente en las capas de salida de la red. No obstante, para poder comprender como se codifican, el análisis debe realizarse en base a toda la red, es decir, considerando las representaciones intermedias.

En el **segundo y principal** aporte de esta tesis, por ende, desarrollamos métri-

cas para medir la invarianza y equivarianza de las redes. Dichas métricas permiten cuantificar estas propiedades de forma empírica no solo en la salida de la red sino también en sus representaciones internas. De esta forma, podemos visualizar y cuantificar que tan invariante o equivariante es una red, ya sea en su totalidad, por capas, o por activaciones individuales. Las métricas son aplicables a cualquier red neuronal, sin importar su diseño o arquitectura, así como a cualquier conjunto de transformaciones. Realizamos una implementación de las métricas en una librería de código abierto, con soporte para la librería tensorial *PyTorch*. Las métricas fueron validadas para verificar su correcto funcionamiento y utilidad. Además, estudiamos sus propiedades, como la variabilidad ante los conjuntos de datos, transformaciones, inicialización de los pesos, y otras.

Utilizando las métricas, también se evaluamos modelos de redes neuronales convolucionales conocidos para caracterizarlos en términos de su invarianza o equivarianza. Asimismo, caracterizamos diversos tipos de capas como las de *Batch Normalization*, *Max Pooling*, diversas funciones de activación, capas convolucionales con distintos tamaños de filtro, y otros. Los resultados otorgan una primera mirada de los modelos de redes en términos de estas propiedades, y esperamos que puedan fomentar un mejora en ese área.

Por último, hacemos un **tercer aporte** al reconocimiento automático de lengua señas basado en video. El reconocimiento de señas es un subárea del reconocimiento de gestos o acciones. Tiene como objetivo traducir al lenguaje escrito un video en donde una persona se comunica mediante lengua de señas. Desde la aparición de tecnologías de captura de video digital existen intentos de reconocer gestos y señas con diferentes fines. Es un problema multidisciplinar complejo y no resuelto aún de forma completa.

Un paso fundamental en el reconocimiento de señas es la clasificación de formas de mano, ya que estas conllevan una gran parte de la información de una seña. El motivante principal de las interrogantes planteadas sobre modelos de invarianza y equivarianza surge a partir del estudio de técnicas de clasificación de formas de mano. Si bien las redes convolucionales proveen un desempeño ejemplar en varios dominios, su desempeño para la clasificación de formas de mano no ha sido evaluado rigurosamente. Por ende evaluamos diversos modelos de redes neuronales para determinar su aplicabilidad en este dominio.

Utilizando los conjuntos de datos de formas de mano LSA16 y RWTH-PHOENIX-Weather, realizamos experimentos con los modelos LeNet, VGG16D, ResNet, Inception y AllConvolutional para determinar su eficacia como clasificadores en este dominio. Los resultados indican que todos los modelos tienen un desempeño razonable en ambos conjuntos de datos, con resultados iguales o mejores que otros mode-

los diseñados específicamente para la tarea. No obstante, el modelo VGG16D obtuvo los mejores resultados. Incluimos también evaluaciones de transferencia de aprendizaje, con y sin re-entrenamiento de las capas; en ambos casos dichas estrategias obtuvieron un desempeño peor que los modelos entrenados sin transferencia de aprendizaje. Además, realizamos un estudio de varias estrategias de pre-procesamiento de las imágenes, encontrando que la segmentación de las manos del fondo otorga un incremento de desempeño significativo. Por último, también desarrollamos una librería de código abierto para facilitar el acceso y preprocesamiento de bases de datos de formas de manos.

Agradecimientos

A Laura, por la oportunidad de trabajar en el grupo y su dirección en estos años.

Al resto de compañeros del III-LIDI, y en especial a los del grupo: Waldo, César, Augusto, Germán, Majo y Gena.

A Franco en especial, por el trabajo compartido.

A mis viejos y a Flor, por todo su apoyo.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El tema principal de esta tesis es el desarrollo de métricas de equivarianza para el análisis de modelos de aprendizaje automático basados en redes neuronales que requieran estas propiedades.

Dicho tema de investigación surge del tema secundario de la tesis, la clasificación de formas de mano para el reconocimiento de gestos y lengua de señas.

1.1. Motivación

1.1.1. Redes Neuronales y Métricas de Invarianza y Equivarianza a las transformaciones

El Aprendizaje Automático es una rama de la Inteligencia Artificial que estudia sistemas capaces de aprender a realizar una tarea a partir de datos de ejemplo. Es de naturaleza inductiva, a diferencia de la inteligencia artificial clásica, y comprende técnicas y métodos para realizar clasificación, optimización y predicción, mayormente en dominios en donde los problemas no pueden definirse de forma explícita o no existen soluciones analíticas aplicables. Por estos motivos, las técnicas que presenta resultan adecuadas para el procesamiento de imágenes y otras señales [GBC16; LBH15].

Las redes neuronales son modelos de Aprendizaje Automático que consisten en un grafo de computación no lineal con capacidad de aproximación universal y una gran cantidad de parámetros. Dichos parámetros son ajustados mediante un proceso de optimización llamado *entrenamiento* a partir de datos de ejemplo. La optimización busca en general minimizar cierta función de error asociada con alguna tarea en particular. Los modelos de aprendizaje automático basados en redes neuronales han

permitido mejoras de desempeño muy significativas, estableciéndose en los últimos años como la tecnología base del estado del arte.

Dicha prominencia se ha logrado mediante el uso de un conjunto de técnicas de Aprendizaje Automático denominada Redes Neuronales Profundas o Aprendizaje Profundo (Deep Learning). Ese conjunto de técnicas extiende los modelos previos de redes neuronales artificiales con arquitecturas y algoritmos de optimización que permiten entrenar redes de varias capas con grandes cantidades de datos de entrenamiento. Las redes neuronales profundas están siendo utilizadas en todos los campos en donde se utiliza aprendizaje automático [GBC16; LBH15]. Además, se han propuesto una gran cantidad de modelos distintos de redes para distintas tareas [GBC16; LBH15].

Las redes neuronales profundas típicamente constan de una serie de *capas*, que transforman paulatinamente un ejemplo de entrada en la salida de la red. Dichas capas tradicionalmente eran de dos tipos: capas lineales o afines, y funciones de activación no lineales. Las capas lineales tienen la forma $f(x) = W \times x + b$, donde W y b son parámetros optimizados (aprendidos) por la red. Las funciones de activación no lineales otorgan una mayor capacidad de expresión a la red. Son funciones comunes, como la tangente hiperbólica, o la sigmoidea. No tienen parámetros optimizables, y generalmente se ubican entre cada capa lineal, debido a que dos capas lineales consecutivas son equivalentes a una sola. Los valores intermedios que calcula la red y que llamaremos *activaciones* son representaciones intermedias de la entrada. Dichas representaciones también son llamadas características o *features*, y son aprendidas por la red, debido a que los parámetros W y b son optimizados por la red. El espacio en que dichas características existen suelen llamarse *espacio latente u oculto*, debido a que no se observa directamente, como la entrada y la salida de la misma El aprendizaje de estas características es una propiedad fundamental de las redes profundas y una de sus mayores fortalezas.

Las Redes Neuronales Convolucionales (CNN, por *Convolutional Neural Network*) [GBC16], en particular, han demostrado una especial capacidad para el procesamiento de imágenes y otras señales. Las redes convolucionales son redes neuronales con capas Convolucionales. Las capas convolucionales aplican la operación de convolución a su entrada con filtros FIR. Los parámetros de estos filtros son optimizados o *aprendidos* por la red durante el entrenamiento en lugar de ser determinados de antemano. En el caso de las imágenes, tanto la entrada como la salida de una capa convolucional es una serie de imágenes o *feature maps*. Dichos feature maps son matrices 2D que representan alguna característica de la entrada, y que también llamaremos *activaciones* de la red.

Las capas convolucionales pueden verse como una versión con *pesos atados* de

una capa lineal [GBC16]. Debido a esto, pierden capacidad de expresión y universalidad, pero a cambio son mucho más eficientes y establecen *priors* útiles a la red, como la localidad espacial o temporal en el procesamiento. Por este motivo, las capas convolucionales tienen problemas para lidiar (individualmente) con algunos tipos de entrada [DWD15; MVT16].

En los últimos años, se han propuesto miles de arquitecturas CNNs y sus variaciones [Liu+17; Kha+19]. No obstante, se destacan algunos modelos emblemáticos como LeNet, All Convolutional, VGG y ResNet que han elevado significativamente el estado del arte anterior en cada ocasión[LeC+98; Spr+15; SZ14; He+16; GBC16].

Si bien el desempeño de las redes en varios dominios es excepcional, y se ha avanzado en la comprensión sobre su funcionamiento [Goo+09; ZD19], quedan todavía varias incógnitas respecto a la manera en que las redes neuronales pueden aprender a partir de ejemplos a resolver problemas. Las redes neuronales son modelos de caja negra en donde el resultado del aprendizaje se cristaliza en miles o millones de parámetros aprendidos mediante un proceso de optimización. Estos parámetros permiten a la red representar su entrada en espacios latentes que no son fácilmente interpretables [Goo+09; ZF14; Cad+18; ZD19].

En ocasiones buscamos que una red se comporte de una manera determinada ante ciertos cambios de su entrada [DDFK16; Xie+17]. Por ejemplo, que ignore ciertas características de un objeto, o que cuantifique ciertos cambios del mismo. No obstante, dada la difícil tarea de interpretar los valores de los parámetros, puede ser difícil dotar a las redes de estas propiedades [CW16b; SG18].

Estos requerimientos pueden formalizarse mediante los conceptos de invarianza, auto-equivarianza y equivarianza a las transformaciones.

Coloquialmente, una red es invariante a un conjunto de transformaciones si su salida no cambia cuando una de esas transformaciones se aplica a su entrada. Por ejemplo, dada una red que clasifica imágenes, si el conjunto de transformaciones T son las rotaciones de 0°, 90°, 180° y 270°, la red es invariante a T si su clasificación (es decir, su salida) siempre es la misma aún al aplicar cualquiera de estas rotaciones a la imagen de entrada [DDFK16].

La auto-equivarianza (*same-equivariance* en inglés) es un concepto relacionado. La auto-equivarianza nos permite cuantificar que tan equivalente es transformar la entrada x o la salida $f(x)$. Por ejemplo, dada una red que cambia el estilo de una imagen, la red convierte una imagen x en otra imagen $f(x)$. Si deseáramos además rotar la imagen, tenemos dos opciones, no necesariamente equivalentes: rotar x y luego aplicar f, o aplicar f y luego rotar $f(x)$. La auto-equivarianza nos indica que tan

equivalentes son estas dos opciones [DDFK16].

Finalmente, la equivarianza generaliza ambos conceptos, y nos permite establecer relaciones entre una transformación de la entrada x y su salida $f(x)$. La medición de estas tres propiedades nos permite comprender mejor la forma en que se comporta una red [DDFK16; LV15].

Si bien los conceptos de invarianza, auto-equivarianza y equivarianza son agnósticos al dominio, nos interesa en particular aplicar dichas propiedades en términos del procesamiento de imágenes, que poseen relaciones naturales con ricos conjuntos de transformaciones como las afines (rotaciones, escalados, traslaciones), de color, etc. Por ende, también nos enfocaremos en las redes neuronales convolucionales como modelos.

Por simplicidad, en el resto de esta tesis utilizaremos el término equivarianza para referirnos de forma general a la invarianza, auto-equivarianza y la misma equivarianza como conjunto de propiedades.

La aumentación de datos es una forma de otorgar equivarianza a transformaciones de la entrada a una red. La aumentación de datos consiste en transformar los ejemplos de entrenamiento. Dicha transformación puede implementarse eligiendo una transformación aleatoria para un ejemplo cada vez que la red se entrena con el mismo. Alternativamente, se pueden pre-generar todas las transformaciones posibles de cada ejemplo para obtener un conjunto de datos aumentado. Para cualquiera de los casos, si utilizamos el conjunto de transformaciones al que se busca lograr la invarianza, y se proveen suficientes ejemplos, dicha aumentación induce al modelo a obtener la invarianza. Un proceso similar puede realizarse para la equivarianza, al transformar tanto la entrada como la salida esperada de la red.

Para modelos de redes convolucionales, la invarianza a las transformaciones a través de la aumentación de datos ha sido estudiada ligeramente [Lar+07; LV15]. Estos resultados presentan evidencia favorable para la hipótesis de que las CNNs tradicionales pueden aprender representaciones invariantes y equivariantes aplicando aumentación de datos. No obstante, estas redes pueden requerir un mayor presupuesto computacional para ser entrenadas, ya que deben explorar el espacio de transformaciones de la entrada.

El otro enfoque posible consiste en modificar el modelo o la arquitectura de CNN en lugar de la entrada para adquirir la equivarianza. Por ejemplo, algunos modelos emplean filtros invariantes a la transformaciones, o combinan múltiples predicciones realizadas con versiones transformadas de la entrada. En estas líneas, se han propuestos varias modificaciones para obtener equivarianza a las transformaciones [Lap+16; Zha+17; Lua+17; CW16b; CW16a; LB15; Dai+17; Zho+17; DWD15; Jad+15].

En varios casos, estos modelos requieren de todas formas un entrenamiento con aumentación de datos [CW16a], o incluyen un proceso interno similar [Lap+16].

No obstante, no está claro si estas modificaciones superan a los modelos tradicionales entrenados con aumentación de datos, tanto en términos de eficiencia como de poder de representación. Más aún, los mecanismos mediante los cuales las CNNs tradicionales obtienen equivarianzas tampoco se comprenden completamente, así como la mejor estrategia para adquirir equivarianza mediante aumentación de datos.

Como mencionamos anteriormente, dado el gran número de modelos de redes neuronales, y adicionalmente, la gran cantidad de opciones para obtener equivarianza en las mismas, la tarea de elegir un modelo equivariante apropiado para una tarea determinada no es simple.

Para dominios donde los modelos han sido evaluados, encontramos que varios modelos tienen desempeños similares, y no es claro cuál tiene mejor desempeño [Kha+19]. No obstante, dicha situación no es común para la mayoría de dominios, debido a la gran cantidad de modelos que se proponen. Esta situación es más compleja aún en el caso de modelos con equivarianza, ya que se multiplican las posibilidades entre los posibles modelos y técnicas de equivarianza.

Para poder comparar dichos modelos, es necesario hacerlo no solo en términos de su desempeño en términos métricas de error o tasa de aciertos, sino también en términos de su equivarianza. Es decir, se requiere una métrica de equivarianza de una red neuronal.

La equivarianza puede estimarse midiendo los cambios en las salidas de la red. Siguiendo este principio se han propuesto varios métodos para medir equivarianza en redes neuronales [Lar+07; Pen+14; FF15; Amo+18; BRW18; Kan17; Kau18].

Algunos trabajos proveen un análisis teórico de la invarianza, como el de [AW18] estudian la falta de equivarianza en algunas CNNs mediante conecciones entre el teorema de muestreo de Shannon con las convoluciones espaciadas.

La mayoría, no obstante, utilizan medidas de error o tasas de acierto para medir indirectamente la equivarianza de la red [Kau18; SG18]. Por ejemplo, el mapa de sensibilidad a la traslación de [Kau18] relaciona la tasa de acierto de un clasificador con la posición central de un objeto en una imagen; [SG18] utiliza gráficos 3D para evaluar la equivarianza a varios tipos de transformaciones. Todos estos métodos tienen en común que solo analizan la equivarianza en relación a la tasa de acierto final, sin tomar en cuenta la representación interna de la red.

Los trabajos de [Goo+09] y [LV15], por otro lado, definen métricas para evaluar

la equivarianza en términos de la representación interna de las redes. Sin embargo, dichas métricas no han sido utilizadas para evaluar modelos equivariantes. Aún más, sólo han sido utilizadas en otras dos ocasiones [CT17; Sha+16], lo cual demuestra la necesidad de avanzar en este enfoque, dada su importancia teórica y práctica.

En conclusión, todavía quedan por comprender varios aspectos del funcionamiento de las redes profundas. Entre ellas, cuál es la mejor manera de otorgarles equivarianza a la rotación y a otras transformaciones afines, cuáles son los mecanismos mediante los cuales adquieren dicha equivarianza, y que propiedades tienen las equivarianzas obtenidas. La utilización de métricas de equivarianza que tengan en cuenta la estructura interna de la red y su representación pueden otorgar otra perspectiva sobre las redes neuronales, así como una metodología para diseñar mejores modelos [Goo+09; LV15].

1.1.2. Clasificación de formas de manos

Por otro lado, el reconocimiento de acciones o gestos permite reconocer diversos tipos de movimientos, poses y expresiones de una persona. Un subproblema particular de este campo es el reconocimiento de la lengua de señas. Las lenguas de señas tienen el mismo poder de expresividad que las lenguas escritas, y las utilizan mayormente las comunidades sordas para comunicarse entre ellos y con otras personas. No obstante, la penetración de la lengua sorda entre las comunidades hablantes es muy baja, lo cual dificulta la comunicación entre las comunidades. La integración de la lengua de señas es una temática muy impulsada actualmente por gobiernos y universidades para una mayor integración de las comunidades sordas y hablantes [Bra+19].

Por ende, sería de mucha utilidad contar con una herramienta que pueda 1) traducir la lengua escrita a la lengua de señas y 2) viceversa. Si bien ambos problemas están relacionados, cada uno tiene problemáticas distintas. En particular, nos interesa el segundo problema; traducir, a partir de un video, de la lengua de señas a la lengua escrita, el cual llamaremos *Reconocimiento automático de Lengua de Señas*. Una herramienta así podría utilizarse no sólo para la traducción sino también para la enseñanza de la lengua [Bra+19].

El reconocimiento automático de lengua de señas es un problema multidisciplinar sumamente complejo que hoy en día sigue sin ser resuelto en forma total. Si bien en el último tiempo han habido avances en el reconocimiento de gestos, impulsados principalmente por el desarrollo de nuevas tecnologías, aún queda un largo camino por recorrer para construir aplicaciones precisas y robustas que permitan la traducción e interpretación de los gestos realizados por un intérprete. La compleja

naturaleza de los gestos motivan esfuerzos de diversas áreas de investigación como interacción hombre-máquina, visión por computador, análisis de movimientos, aprendizaje automático y reconocimiento de patrones [Bra+19].

Las señas tienen diversas características que permiten definirlas y distinguirlas unas de otras. Sus principales son la posición, forma y movimiento de las manos, así como la expresión del rostro y la pose del cuerpo. Debido a que es muy difícil definir de forma escrita que caracteriza a cada seña, los enfoques de Aprendizaje Automático basados en datos son especialmente importantes. La tarea completa de reconocer una seña involucra diferentes pasos: la ubicación de las manos del intérprete, el reconocimiento de las formas de las manos, el seguimiento de las manos para detectar el movimiento realizado, la interpretación de los rasgos no manuales y el contexto, la interpretación semántica y traducción al lenguaje escrito [Bra+19].

En estudios previos se ha determinado que la forma de la mano es el atributo más importante para reconocer correctamente una seña. Por ende, en esta tesis nos enfocamos en dicha etapa del reconocimiento de lenguas de señas. La clasificación de formas de mano para lengua de señas tiene algunas dificultades adicionales. Dado que la traslación, rotación o escala de la mano no afecta a la forma de las manos, para aumentar la robustez de los clasificadores de formas de manos es necesario dotarlos con equivarianza a estas transformaciones[Bra+19]. De la misma forma, varios problemas de otras áreas que lidian con el reconocimiento de objetos que tienen equivarianzas naturales a la rotación y otras transformaciones se beneficiarían de estos métodos, como el análisis de texturas, imágenes de microscopios o clasificación de galaxias [DWD15; Chi+19; Kan+18].

Dado estos antecedentes, resulta natural aplicar Redes Profundas al problema del reconocimiento de Lengua de Señas. En particular, las Redes Convolucionales Profundas (redes neuronales profundas con capas convolucionales) se han aplicado al reconocimiento de secuencias de acciones, gestos y señas en videos con resultados que superan al estado del arte [KNB16]. No obstante, no ha habido un análisis riguroso sobre su aplicación en el reconocimiento de formas de mano, con el objetivo de determinar qué tipo de arquitecturas funciona mejor, en especial en el contexto de la necesidad de invarianza a las transformaciones.

En resumen, si bien hay varios métodos propuestos para otorgar equivarianzas a las redes, y en consecuencia aumentar su capacidad de reconocer formas de manos de la Lengua de Señas, no existe ninguna comparación exhaustiva entre los distintos métodos, y lo que es aún más acruciante, existen pocos análisis en profundidad sobre como dichos métodos codifican las equivarianzas.

De estas interrogantes surgen los objetivos de investigación de esta tesis.

1.2. Objetivos

Nuestro objetivo general en esta tesis es contribuir al entendimiento y mejora de la equivarianza de los modelos de redes neuronales, en particular aplicados a la clasificación de formas de mano para la lengua de seña y otros tipos de gestos mediante modelos de redes convolucionales.

Para ello, establecimos los siguientes objetivos particulares:

1. Analizar los modelos específicos para equivarianza en CNNs.
2. Comparar los modelos específicos y la aumentación de datos para obtener equivarianza. Evaluar estrategias de transferencia de aprendizaje para obtener modelos equivariantes a partir de modelos que no lo son.
3. Desarrollar métricas de equivarianza para las activaciones o representaciones internas de las redes neuronales. Implementar las métricas en una librería de código abierto. Analizar el comportamiento de las métricas. Comparar con las métricas existentes.
4. Caracterizar modelos de CNN para la clasificación de imágenes en términos de su equivarianza con las métricas propuestas.
5. Comparar los modelos de CNN, con y sin equivarianza, para la clasificación de formas de mano.

Debido a la existencia de múltiples métodos para lograr equivarianza, y a la falta de comparaciones rigurosas y profundas de los mismos, el alcance de esta tesis se limita al análisis y no propone nuevos modelos equivariantes de redes.

1.3. Contribuciones

En esta tesis, presentamos varias contribuciones:

- Un análisis comparativo de modelos basados en Redes Neuronales para la clasificación de formas de mano de la lengua de señas.
- Un análisis de estrategias para lograr equivarianza a las rotaciones en redes neuronales:
    - Comparación del desempeño de estrategias basadas en aumentación de datos vs diseños especiales de redes y capas.

    - Determinación de estrategias para reentrenar redes de forma que adquieran equivarianza a las rotaciones.

- Un conjunto de métricas para analizar empíricamente la equivarianza de las redes neuronales, así como de cualquier otro modelo basado en representaciones latentes.

    - Validación de las métricas para determinar que logran medir lo requerido.
    - Análisis de distintas variantes de las métricas propuestas.
    - Un análisis de las métricas propuestas, en términos de variabilidad ante distintas transformaciones, modelos, inicialización de pesos.
    - Análisis del cambio en la estructura de la equivarianza ante distintos hiperparámetros, como el tipo de función de activación, uso de capas de *Max Pooling*, *Batch Normalization*, y tamaño del *kernel* de las capas convolucionales.
    - Análisis de la estructura de la equivarianza de distintos modelos reconocidos redes convolucionales, como ResNet [He+16], AllConvolutional [Spr+15] y VGG [SZ14].
    - Análisis del impacto en la equivarianza al utilizar modelos especializados para obtener equivarianza como Transformational Invariance Pooling [Lap+16]
    - Análisis de la estructura condicional de clase de la equivarianza de los modelos.
    - Determinación del efecto de la diversidad y complejidad de las transformaciones en las métricas.

1.4. Publicaciones

Las siguientes publicaciones están relacionadas directamente con las contribuciones que presentamos en esta tesis:

1. Facundo Quiroga y col. «A study of convolutional architectures for handshape recognition applied to sign language». En: *XXIII Congreso Argentino de Ciencias de la Computación (La Plata, 2017)*. Springer International Publishing. 2017

2. Facundo Quiroga y col. «Revisiting Data Augmentation for Rotational Invariance in Convolutional Neural Networks». En: *International Conference on Modelling and Simulation in Management Sciences*. Springer. 2018, págs. 127-141

3. Facundo Quiroga y col. «Measuring (in) variances in Convolutional Networks». En: *Conference on Cloud Computing and Big Data*. Springer. 2019, págs. 98-109

4. Ulises Jeremias Cornejo Fandos y col. «Recognizing Handshapes using Small Datasets». En: *XXIII Congreso Argentino de Ciencias de la Computación (Rio Cuarto, 2019)*. Ed. por Springer. 2019

Las siguientes publicaciones constituyen trabajos previos relacionados realizados antes y durante la tesis:

1. Facundo Quiroga y Leonardo César Corbalán. «A novel competitive neural classifier for gesture recognition with small training sets». En: *XVIII Congreso Argentino de Ciencias de la Computación (CACIC)*. Red de Universidades con Carreras en Informática (RedUNCI). 2013

2. Franco Ronchetti y col. «Distribution of Action Movements (DAM): a Descriptor for Human Action Recognition». En: *Frontiers of Computer Science* 9.6 (2015), págs. 956-965. ISSN: 2095-2236. DOI: 10.1007/s11704-015-4320-x

3. Facundo Quiroga y col. «Handshape recognition for argentinian sign language using probsom». En: *Journal of Computer Science & Technology* 16.1 (2016). ISSN: 1666-6038

4. Facundo Quiroga y col. «Sign languague recognition without frame-sequencing constraints: A proof of concept on the argentinian sign language». En: *Ibero-American Conference on Artificial Intelligence*. Springer International Publishing. 2016, págs. 338-349

5. Facundo Quiroga y col. «LSA64: An Argentinian Sign Language Dataset». En: *XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016)*. Ed. por Springer. Red de Universidades con Carreras en Informática (RedUNCI). 2016, págs. 794-803

- **Métricas transformacionales**: Una librería de código abierto para computar métricas transformacionales como las definidas en esta tesis para modelos de redes neuronales, utilizando el framework *PyTorch* y *Numpy* (https://github.com/facundoq/transformational_measures).

- **Handshape Datasets**: Una librería de código abierto para facilitar la descarga y preprocesamiento de conjuntos de datos de formas de mano (https://github.com/midusi/handshape_datasets).

- **LSA16**: Una base de datos de imágenes de formas de mano para la Lengua de Señas Argentina, disponible públicamente y de forma gratuita bajo licencia creative commons (http://facundoq.github.io/unlp/lsa16/).
- **LSA64**: Una base de datos de videos de señas dinámicas de la Lengua de Señas Argentina, disponible públicamente y de forma gratuita bajo licencia creative commons (http://facundoq.github.io/unlp/lsa64/).

1.5. Organización de la tesis

El Capítulo 2 presenta el marco teórico de la misma, en el cual se describe en más detalle el funcionamiento de las redes neuronales y convolucionales, los conceptos de equivarianza, y los conceptos relacionados de invarianza y auto-equivarianza. Además, se analiza la bibliografía existente en la actualidad acerca de los modelos de redes convolucionales y conjuntos de datos utilizados, modelos de redes para obtener equivarianza, y otras métricas de equivarianza existentes. Por último, se detalla la problemática del reconocimiento de lengua de señas y en particular la clasificación de formas de manos.

Los Capítulos 3, 4 y 6 contienen las contribuciones de esta tesis.

En particular, el Capítulo 3 compara modelos CNN especializados en invarianza a la rotación contra la estrategia de aumentación de datos, para problemas de clasificación clásicos de las bases de datos MNIST y CIFAR10. Además, se realiza un análisis de estrategias de transferencia de aprendizaje para obtener modelos invariantes a partir de modelos que no lo son.

A su vez, el Capítulo 4, la contribución principal de esta tesis, presenta las métricas de equivarianza propuestas. El Capítulo 5 complementa este capítulo, presentando los análisis de validez y comportamiento de las métricas, y la caracterización de modelos a partir de las mismas.

Luego, en el Capítulo 6 se desarrolla una comparación de modelos de CNN para la clasificación de formas de manos, con énfasis en invarianza a la translación, rotación y escalado.

Finalmente, el Capítulo 7 presenta las conclusiones y los trabajos futuros.

Capítulo 2

Marco teórico

2.1. Aprendizaje Automático

Un programa que aprende automáticamente es aquel que no se programa explícitamente para resolver un problema, sino que utiliza un algoritmo de aprendizaje en base a datos de instancias o ejemplares del problema para generar un modelo del mismo (Figura 2.1).

Con ese modelo, se pueden analizar ejemplares del problema general, obteniendo alguna inferencia, con cierto grado de error (Figura 2.2).

Siguiendo la definición más formal de Mitchell [Mit97] donde la ''experiencia'' son los ejemplares del problema:

> Un programa de computadora aprende de la experiencia E respecto a una clase de tareas T y medida de error P, si su error en las tareas en T, medida por P, mejora con la experiencia E. [1]

En general, las técnicas de aprendizaje automático se aplican a problemas donde

[1] La P es por *performance*. En general, hablar de performance o desempeño es equivalente a hablar de error, ya que a mayor performance menor error y viceversa

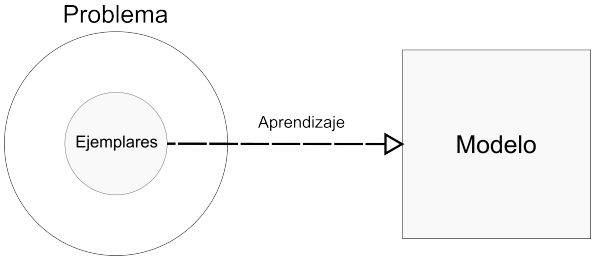


Figura 2.1: Generación de un modelo a partir de ejemplares de un problema: la esencia del aprendizaje automático.

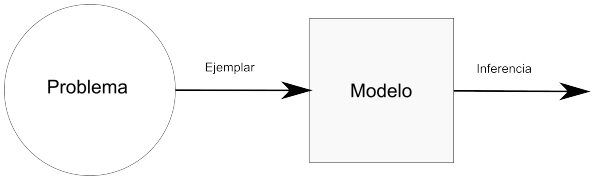


Figura 2.2: Obtención de una inferencia a partir de algún ejemplar del problema.

no se conoce una solución algorítmica definitiva o esta tiene características indeseables: es computacionalmente demandante, numéricamente inestable, requiere programación explícita para adaptarse a cambios, es poco robusta a fallos de sensores, o requiere intervención manual.

Una clase muy importante de problemas en donde se aplica aprendizaje automático es la de aquellos donde los humanos tienen un bajo nivel de error innato, como el reconocimiento de voz, de gestos, de objetos, etc, pero es difícil codificar reglas para realizar dichas tareas explícitamente. El aprendizaje automático es importante, entonces, porque las ocurrencias de la existencia humana acontecen en forma de patrones. La información del lenguaje, el habla, el dibujo y el entendimiento de las imágenes, todas involucran patrones.

No hay un aspecto o secuencia de aspectos que determine absolutamente y sin ambigüedad el significado de un patrón, y dicho significado varía con el contexto. Los humanos solemos ser bastante buenos en este tipo de tareas ya que tenemos un cerebro capaz de procesar la información relevante a dichas tareas de forma eficiente, adaptativa y eficaz. Si bien esto no es algo del todo sorprendente, ya que nos interesan y nos parecen importantes justamente porque podemos hacerlos, lograr que una computadora emule ciertas capacidades de nuestro cerebro es un objetivo de gran interés para el aprendizaje automático. De hecho, se puede argumentar que, en esencia, muchos de los algoritmos de aprendizaje automático son aproximaciones matemáticas y computacionales al funcionamiento del cerebro [Sha53; Mel+09; Lio12].

2.1.1. Ejemplo

Para ilustrar la idea del aprendizaje automático, tomaremos como ejemplo el problema del reconocimiento de formas de mano.

Supongamos que en una base de datos (BD) existe un conjunto de 20 imágenes que representan ejemplares de formas de mano: 10 son imágenes de manos abiertas y 10 de manos cerradas. Requerimos un programa que pueda, dado una nueva imagen de ejemplo, decidir si es de la primera clase (abierta) o de la segunda (cerrada).

En un enfoque más tradicional, ignoraríamos la BD de 20 ejemplares, e intentaríamos definir matemáticamente el concepto de forma de mano. Podríamos describir una forma de mano en base a la posición de los dedos, para los casos abierto o cerrado. Luego, escribimos un algoritmo que tome un nuevo ejemplar, realice un ajuste del mismo a los dos modelos, y decida si pertenece a alguno de los dos, o a ninguno.

Este enfoque corresponde al de Hoare, donde existe un programa *f* de reconocimiento automático, y se pueden definir tuplas de Hoare del tipo:

{Precondición} *Programa* {Poscondición}

Por ejemplo:

{La entrada se ajusta al modelo de forma de mano abierta } *f*
{La imagen es de una mano abierta}

o:

{La entrada no se ajusta a ninguno de los modelos} *f*
{La imagen no tiene una mano cerrada ni una abierta}

Si bien es cierto que estas tuplas pueden ser triviales, lo importante es notar que *podemos* escribirlas porque definimos el modelo de formas de mano *explícitamente*.

En el enfoque de aprendizaje automático, en cambio, podríamos entrenar un clasificador *f* que aprenda las características particulares de los ejemplares de cada clase de forma de mano *a partir de los ejemplares de la BD*. De esa manera, el modelo surge de los datos, con lo cual prescinde de la necesidad de programación explícita de cada nuevo tipo de forma de mano. Luego evaluamos el clasificador con nuevos ejemplares de formas de manos. En el caso ideal, el clasificador clasifica correctamente todos los nuevos ejemplares de manos. En la práctica, obtendremos un porcentaje de clasificación incorrecta que habla del **error del clasificador**.

Entonces, hablando ahora en general, en el enfoque de aprendizaje automático nos desligamos de la noción de Hoare de computación, y modelamos a *f* como un programa o una función que realiza una tarea de **inferencia** con cierto grado de error, de forma similar a cómo trabajamos con métodos numéricos o simulaciones. Es usual llamar a *f* el **modelo** del problema. Para generar el modelo, utilizamos un conjunto de ejemplares D, que es un subconjunto del conjunto de ejemplares de un problema o **dominio** $\mathcal{D}$ (en el ejemplo, $\mathcal{D}$ es el conjunto de todas las posibles formas de mano).

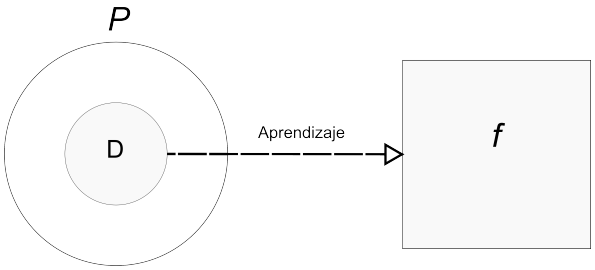


Figura 2.3: Generación de un modelo f a partir de ejemplares de un problema, D

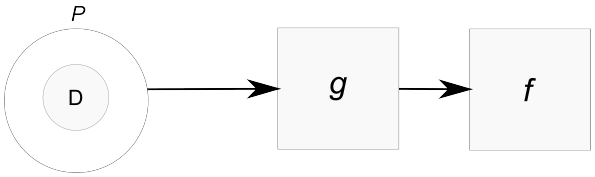


Figura 2.4: Generación de un modelo f a partir de un conjunto de ejemplares del problema, D y una función de entrenamiento, g

Las distintas maneras de generar la función f dan lugar a distintos algoritmos de aprendizaje automático. En general un esquema de aprendizaje automático está definido por un **algoritmo o función de entrenamiento** g, que genera o **entrena** una función f en base al conjunto de ejemplares D. [2]

La elección de la función de entrenamiento g depende fuertemente del problema a resolver. Distintas funciones de aprendizaje automático g generarán distintos modelos f.

Como mencionamos, toda función f generada por un algoritmo de aprendizaje automático g y un conjunto de datos D cometerá algún tipo de **error**, ya que el aprendizaje casi nunca es perfecto, y D representa solo una parte del dominio a modelar.

De este modo, dado un conjunto de datos D podemos definir como objetivo general de un algoritmo de entrenamiento la resolución del problema de optimización, en donde buscamos una función de inferencia f que minimice el **error** del modelo, medido con el conjunto de datos D:

$$\underset{f}{\text{Minimizar}} \quad error(f, D)$$

Formalmente, $g :: [\mathcal{P}] \mapsto f$, donde $\mathcal{P}$ es el dominio de los ejemplares del problema. D es una lista de ejemplares de ese dominio (o sea, un conjunto de datos), y f es una **función de inferencia** tal que $f :: \mathcal{P} \mapsto Inferencia$. Aquí, $Inferencia$ es un conjunto de inferencias posibles a realizar sobre $\mathcal{P}$, dependiente del problema a re-

[2] De la misma manera en que nos referimos a f como función de inferencia o modelo, hablaremos de función y algoritmo de entrenamiento g de forma intercambiable.

solver. En el ejemplo de las formas de mano, *Inferencia* sería un conjunto de etiquetas para cada clase y la etiqueta que indica que el ejemplar no pertenece a ninguna clase conocida, es decir, $Inferencia = \{Abierto, Cerrado, Ninguno\}$.

Entonces, f es una función de inferencia generada por g, una función de aprendizaje: una vez ''entrenada'' f con un conjunto de datos de $\mathcal{P}$, se puede utilizar para hacer inferencia sobre el dominio $\mathcal{P}$. Por otro lado, si asumimos la existencia de una ''verdadera'' función $f' :: \mathcal{P} \mapsto Inferencia$ que asigna correctamente una inferencia para cada ejemplar del problema, podemos considerar que f aproxima o estima f' en base a un conjunto de datos $D \subset \mathcal{P}$ y una función de aprendizaje g.[3].

Volviendo al ejemplo, el problema de reconocimiento de formas de mano es un problema de **clasificación**, en donde buscamos inferir la clase de los ejemplares (abierta o cerrada). No obstante, un programa de aprendizaje automático se puede entrenar para aprender una función f arbitraria.

En otras palabras, la imagen de f no tiene que ser un conjunto clases a asignar a los ejemplares, como en el ejemplo; podría ser que busquemos aprender la función $t(x) = x^3$, que es una función que convierte números reales a números reales (o sea, $t : \mathbb{R} \to \mathbb{R}$) y entonces en ese caso $Inferencia = \mathbb{R}$ y $\mathcal{P} = \mathbb{R}$.

Con este ejemplo, esperamos haber establecido un contexto y una motivación para el desarrollo de las siguientes secciones. A continuación detallamos varias de las aplicaciones de aprendizaje automático, a modo de comprender los alcances de esta técnica, y luego el proceso de aprendizaje junto con sus propiedades más importantes, profundizando el esquema de generación del modelo f.

2.1.2. Aplicaciones

Las técnicas de aprendizaje automático se usan con éxito en una enorme variedad de problemas. Un artículo de Widrow [WRL94] de 1994 ya mencionaba una gran cantidad de aplicaciones de las redes neuronales artificiales, uno de los modelos más utilizados de aprendizaje automático. Entre ellas, encomtramos aplicaciones en:

- Telecomunicaciones para la mitigación del eco y la ecualización de la señal.
- Detección de eventos y disminución del error de operación en aceleradores de partículas.

[3]En verdad, generalmente no podemos definir a g como una verdadera función ya que el entrenamiento suele contener elementos aleatorios; es decir, un algoritmo de entrenamiento puede generar distintos clasificadores f aunque se entrene con el mismo conjunto de datos D. Podemos obviar este hecho asumiendo que en cada generación de f se utiliza un g_s de la misma familia de generadores que solo difieran en una semilla aleatoria s distinta.

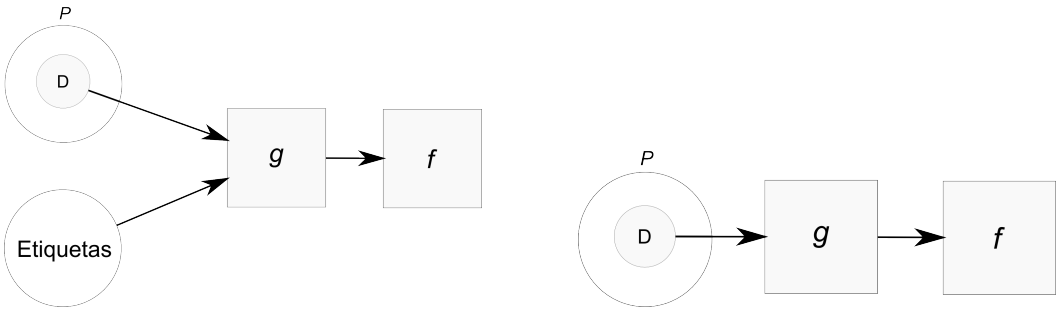
- Detección de fraudes en operaciones bancarias y tarjetas de crédito.
- Reconocimiento óptico de caracteres de máquina y manuscritos.
- Detección de cáncer y otras enfermedades en imágenes médicas y muestras biológicas.
- Enfoque automático en telescopios.

En la actualidad, la utilización de dichas técnicas no ha menguado. En el área de la industria y los servicios, se han utilizado para clasificar el tráfico en conexiones a Internet [Yua+10] y desarrollar sistemas de control integral de manufactura guiados por aprendizaje automático de reglas en base a experiencias previas [SGL12]. También se han empleado en tareas de predicción de índices de la bolsa [KH00] y predicción de la disponibilidad de recursos hídricos [MD00]. Se utilizan en sistemas de control de manufactura, para manejar automáticamente automóviles, aviones, helicópteros y robots [Pom91; Pom93; AH99; RBS12].

El procesamiento de señales médicas en particular se ha visto muy beneficiado por el uso de aprendizaje automático debido a que este suele superar a los métodos estadísticos tradicionales. Se ha utilizado para desarrollar marcadores biológicos para el diagnóstico por computadora de cáncer [Abe+10] y para la detección de Alzheimer y Trastornos con Déficit de Atención e Hiperactividad [Leh+07; Sid+12], respectivamente. También se han utilizado técnicas de aprendizaje automático en la evaluación de datos neurológicos para resolver diversos problemas, como el diagnóstico clínico de enfermedades, la elaboración de pronósticos y el mapeo de estructuras cerebrales y su funcionamiento. En el área de reconocimiento de patrones cerebrales, se han desarrollado verdaderos juegos mentales [FLR09], así como interfaces para interactuar con personas en estado vegetativo [Lul+12], con posibles aplicaciones concretas debido a la comercialización masiva de dispositivos de captura.

Tienen gran utilidad en problemas de reconocimiento de voz, del hablante, de objetos en imágenes y videos y de gestos. Su efectividad en estas aplicaciones ha aumentado enormemente en los últimos años, especialmente en el reconocimiento de secuencias de acciones en videos y en etiquetado de escenas [Le+11a; Le+11b; Far+13] y el reconocimiento de voz [Hin+12; Den+13].

Debemos considerar que varias técnicas de aprendizaje automático como las Redes Neuronales entrenan funciones de inferencia con la propiedad de ser aproximadores universales, es decir, pueden aproximar cualquier función continua en un hipercubo M-dimensional $[0,1]^M$ con un error arbitrariamente pequeño, dado un



Supervisado No supervisado

Figura 2.5: Esquema de entrenamiento de un algoritmo supervisado y otro no supervisado.

modelo con la complejidad requerida [Hay94], lo cual sienta bases teóricas que apoyan la experiencia de su gran aplicabilidad. Por ende se han utilizado para aproximar funciones como la Transformada Discreta de Fourier [Vel08], implementar Análisis de Componentes Principales No-Lineal (NPCA) [Kra91] y realizar optimización combinatoria [Smi99].

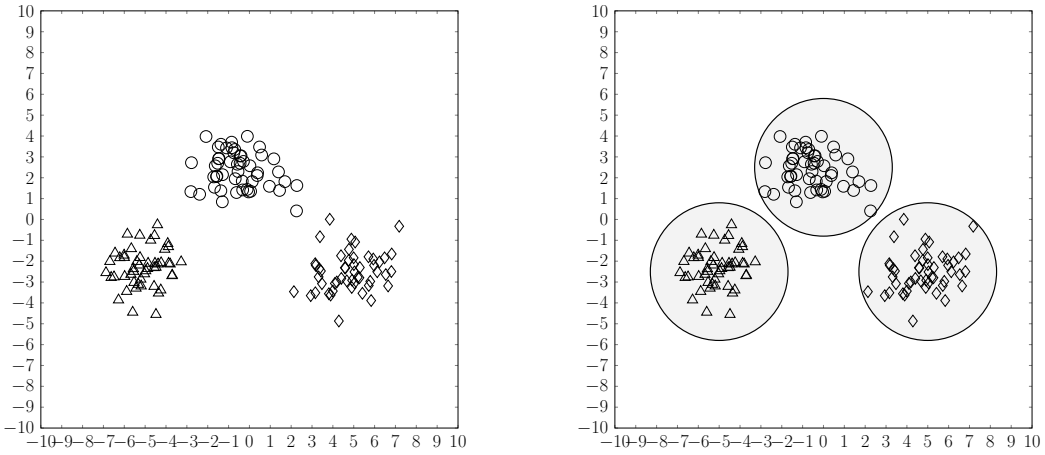
2.1.3. Aprendizaje Supervisado

Si bien existen varias clasificaciones de los métodos y técnicas de aprendizaje automático, la más simple y quizás la más significativa distingue entre algoritmos de entrenamiento **supervisado** y **no supervisado**.

Los algoritmos supervisados son aquellos donde se conoce a priori el resultado que se busca aprender para cada ejemplar de entrenamiento del problema. En el ejemplo de las formas de mano, el aprendizaje es supervisado si para cada una de las 20 imágenes de la BD se conoce su clase, es decir, se sabe a priori si son manos abiertas o cerradas. El programa entonces aprende a clasificar nuevas manos en base a etiquetas anteriores.

En el aprendizaje no supervisado no hay o se ignoran los resultados previamente conocidos, no hay etiquetas asociadas a los ejemplares. El programa aprende relaciones entre los datos, descubre las estructuras subyacentes del espacio de ejemplares $\mathcal{P}$, pero no se guía por ninguna etiqueta asociada a los ejemplares, sino que usa solamente los datos de los ejemplares mismos. En el ejemplo, sería como tener sólo las veinte imágenes de manos pero no saber (al menos a priori) de qué clase son.

En la práctica, ambas técnicas se pueden utilizar conjuntamente, por lo que si bien un algoritmo puede ser tildado de supervisado o no, en general un sistema completo de aprendizaje automático emplea varias combinaciones de algoritmos de los dos tipos.



a b

Figura 2.6: (a) Ejemplos de cada clase. (b) Regiones de cada clase estimadas por un clasificador entrenado con dichos ejemplos.

2.1.4. Problemas de clasificación

El aprendizaje automático, en general, y la clasificación automática, pueden considerarse efectivamente desde un punto de vista geométrico. En el mismo, los modelos de clasificación ven a los ejemplares a clasificar como puntos o vectores en un espacio d dimensional.

Por ejemplo, si se deben clasificar personas para saber si pertenecen a una población de riesgo de obesidad, y de ellas se conoce su edad y su peso, entonces los ejemplares son vectores de $d = 2$ componentes que existen en un espacio bi-dimensional, como $\mathbb{R}^2$. Este espacio sería el que previamente se identificó como $\mathcal{P}$. Además, como se tiene un problema de clasificación supervisada, se conoce el conjunto de clases a asignar a los ejemplares, y los del conjunto de entrenamiento D se encuentran etiquetados con la clase correspondiente.

Un algoritmo de entrenamiento genera un modelo de clasificación f en base a estos ejemplares. La hipótesis básica detrás de la mayoría de los modelos es de **localidad**, es decir, que si un ejemplar x tiene una etiqueta y, entonces los ejemplares x' cercanos a x probablemente tendrán la misma etiqueta. Al considerar todos los ejemplares, esta hipótesis da lugar a que el modelo determine distintas **regiones** de $\mathcal{P}$, que corresponden a ciertas clases.

Entonces, el desafío de entrenar un modelo de clasificación f es el de estimar las regiones de cada clase, en base a los ejemplares de entrenamiento. Luego de entrenado, a la hora de clasificar, f recibe un ejemplar como entrada, y da como resultado una clase estimada (Figura 2.7).

Formalmente, un clasificador puede definirse como una función $f :: \mathcal{P} \mapsto C$,

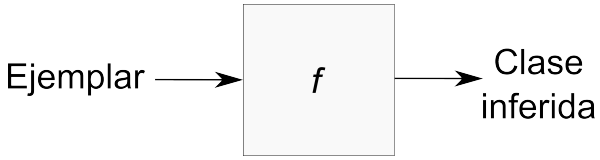


Figura 2.7: Funcionamiento de la función de clasificación o inferencia f.

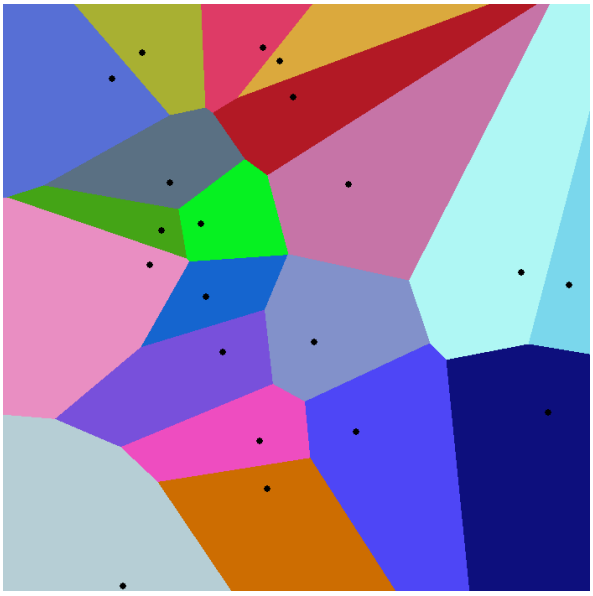


Figura 2.8: Regiones de Voronoi para un problema de clasificación con dominio $\mathcal{P} = \mathbb{R}^2$. Cada región $\mathcal{P}_i$ está representada con un color distinto. Los puntos representan a los centros de la región. Los colores indican que cada a región le corresponde a alguna clase. Es posible que más de una región corresponda a la misma clase.

donde $\mathcal{P}$ es el conjunto de todos los ejemplares del problema, y $C = \{1..N\} \cup \perp$ es el conjunto de etiquetas de las N clases ($\perp$ representa la clase nula, es decir, que un ejemplar no pertenece a ninguna clase).

Para generar el clasificador f utilizamos base a un conjunto ordenado D de n ejemplares x_i, $D = \{x_i \in \mathcal{P}\} \quad i = 1..n$, donde por ejemplo podría ser que $\mathcal{P} = \mathbb{R}^d$, como será en el caso de las formas de mano, o $\mathcal{P} = \mathbb{R}^2$, como en el de clasificar personas en base a su edad y peso. A su vez, cada ejemplar pertenece a una de las $|C|$ clases conocidas, por ende asociado a cada x_i hay una etiqueta $y_i \in C$ que indica a qué clase pertenece x_i.

Volviendo al punto de vista geométrico, podemos interpretar a f como una función que particiona $\mathcal{P}$ en subconjuntos disjuntos $\mathcal{P}_i$ (posiblemente infinitos) llamados **regiones de Voronoi**, con $\cup_i \mathcal{P}_i = \mathcal{P}$, y luego asigna a cada $\mathcal{P}_i$ una clase (o ninguna) (Figura 2.8).

Como vimos anteriormente, generamos la función f con un algoritmo de entrenamiento en base a un conjunto de ejemplares de entrenamiento y a ciertos conocimientos específicos del dominio del problema a resolver. De forma análoga al argumento

general sobre aprendizaje automático, en esencia f aproxima una función f' que codifica las clases ''verdaderas'' de los ejemplares de $\mathcal{P}$; es decir, f' conoce las regiones verdaderas de cada clase.

2.1.5. Experimentos

Una vez entrenado un clasificador, típicamente se quiere conocer que tan bien clasifica a los ejemplares del dominio del problema a resolver. Para evaluar un clasificador podemos realizar el siguiente esquema:

- Obtener los datos de los ejemplares y preprocesarlos.
- Seleccionar los hiperparámetros que definen el clasificador.
- Entrenar un clasificador con los ejemplares y un algoritmo de entrenamiento.
- Probar el modelo generado obteniendo alguna medida de error.

Llamaremos a esta serie de pasos un **experimento**. El resultado del experimento es el modelo entrenado y una medida de error del mismo respecto al dominio $\mathcal{P}$ y la asignación de etiquetas y_i para los ejemplares.

El objetivo del experimento es el último punto, la evaluación del desempeño del modelo. Para ello, pensaremos en el experimento como un experimento estadístico, que estimará el desempeño del modelo dado el dominio $\mathcal{P}$ y la asignación de etiquetas y_i.

Como en toda prueba estadística, nos interesará el *comportamiento promedio* del modelo, y poder caracterizar su variabilidad.

Fuentes de variabilidad

Un experimento de clasificación tiene dos fuentes principales de variabilidad o aleatoriedad.

La primera está dada por los datos utilizados para entrenar y probar el clasificador. Distintos datos para entrenar generan modelos distintos, y por ende el error será distinto. De la misma forma, usar datos distintos para probar el clasificador hará que varíe el valor del estimador calculado. En general, cuantos más datos para entrenar pueda usar el algoritmo de entrenamiento y mejor sea su calidad (representatividad de la variabilidad del dominio del problema, poco error de medición), mejor será el

modelo generado, por las mismas razones que el ajuste a una curva en base a puntos de ejemplo de la misma es más preciso con más puntos.

Del mismo modo, cuanto más y mejores datos se utilicen para evaluar el error del clasificador mejor será la confianza de las pruebas, por las mismas razones que una muestra grande de datos es deseable para realizar un test estadístico.

La segunda se encuentra en la generación del clasificador. Muchos algoritmos de entrenamiento utilizan el enfoque de partir de un estado generado aleatoriamente, y luego mejorarlo iterativamente durante el entrenamiento. La inicialización es aleatoria, y como el estado final depende del estado inicial, también es aleatorio el error del clasificador. En otros casos, el estado inicial puede estar fijado de antemano, pero los pasos para llegar al estado final contienen elementos aleatorios.

Si bien esta fuente de aleatoriedad puede eliminarse utilizando una semilla fija en la generación de números aleatorios, dicha estrategia impediría el análisis de una serie de experimentos con herramientas estadísticas, lo cual es necesario para obtener una medida de error del clasificador en promedio realizando varios experimentos y agregando los resultados.

El concepto de experimento será útil para hacer referencia a esta serie de pasos en las siguientes secciones. El objetivo de un experimento es generar y evaluar un clasificador. En la evaluación, interesa esencialmente el comportamiento del clasificador entrenado en *nuevos* ejemplares del problema, no vistos en la etapa de entrenamiento; este es el problema de la **generalización**.

2.1.6. Generalización

Un clasificador se genera en base a un conjunto de datos de *entrenamiento*. Una vez generado el clasificador, interesa conocer su desempeño, no en el conjunto de datos de entrenamiento, sino en su dominio $\mathcal{P}$. El desempeño de un clasificador en ejemplares del problema no vistos en la etapa de entrenamiento se denomina como la **capacidad de generalización** del clasificador.

Para conocer esta capacidad de generalización, se aplican métodos estadísticos para estimar el error del clasificador en la población de posibles ejemplares $\mathcal{P}$. Dado que es imposible probar todos los elementos de $\mathcal{P}$, se utiliza algún conjunto de datos $D \subset \mathcal{P}$ para estimar el error.

Hay distintos estimadores de error, y algunos son específicos al tipo de clasificador a utilizar. El más simple y genérico es el porcentaje de ejemplares clasificados incorrectamente o *tasa de aciertos*.

La definición de la funcion de error del modelo generalmente se hace en base a una función de error de una ejemplo en particular, que llamaremos $\ell(x,f)$, y cuantifica el error de f al querer clasificar el ejemplar x.

Utilizando la función de error por ejemplar ℓ, podemos definir el error esperado de f (Ecuación [2.1]).

$$\rho_\ell(\mathcal{P},f) = E_x[\ell(x,f)] \qquad [2.1]$$

Como $\mathcal{P}$ suele ser infinito, aproximamos $\rho_\ell(\mathcal{P},f)$ con $\rho_\ell(D,f)$.

En resumen, para evaluar un modelo, un experimento utiliza un conjunto de datos D para entrenar a f mediante $f = g(D)$ y luego estima su error con $\rho_\ell(D,f)$.

Validación con retención

Un problema con la estrategia anterior es que los algoritmos de entrenamiento en general están basados, en forma implícita o explícita, en la idea de encontrar un f que justamente minimice una medida de error como $\rho_\ell(D,f)$. Entonces, si se utiliza el mismo conjunto de datos para entrenar un clasificador y estimar su error se obtendrá una estimación que subestima el error real de f en $\mathcal{P}$. Es como generar un modelo a partir de ciertos datos, y luego validar dicho modelo con esos mismos datos; claramente, si el modelo está bien generado, es improbable que tenga un alto grado de error en los datos de entrenamiento.

En un caso extremo, si $y(x)$ es la etiqueta correcta para x, se puede definir un clasificador f_D tal que:

$$f_D(x) = \begin{cases} y(x) & \text{si } x \in D \\ \text{clase aleatoria} & \text{de lo contrario} \end{cases} \qquad [2.2]$$

Dicha función f_D[4] tiene un error de clasificación $\rho_\ell(D,f_D) = 0$ en el conjunto D, pero clasifica de forma aleatoria cualquier otro conjunto de ejemplares. Mientras que este es un caso claramente trivial, sirve para ilustrar la importancia de no hacer estimaciones de error con los mismos datos con los cuales fue entrenado un clasificador. De todas formas, en la práctica es usual obtener funciones f que no tengan errores de clasificación en el mismo conjunto en que fueron entrenados, pero que varían ampliamente en su capacidad de clasificar otros ejemplares de $\mathcal{P}$.

[4]Para que f_D sea realmente una función, se asume que dichas etiquetas de clase aleatorias se asignan en la definición de la función y no en su evaluación

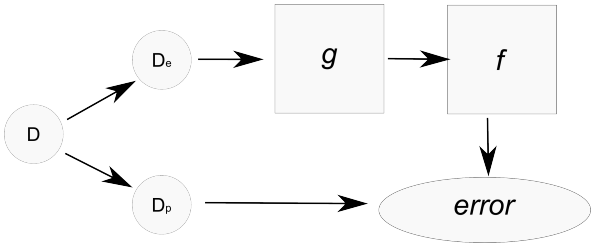


Figura 2.9: Esquema del proceso de validación con retención. El conjunto de ejemplares D se divide en dos, el de entrenamiento D_e y el de prueba D_p. Mediante la función g y D_e, se entrena el modelo f. Luego, se obtiene una medida del error de f en $\mathcal{P}$ utilizando el conjunto D_p.

Para solucionar este problema, es necesario entonces probar a f con ejemplares distintos a los utilizados para entrenar. La idea más simple es tomar a D y en lugar de utilizar *todos* los ejemplares para entrenar, dividir el conjunto en dos: D_e, el conjunto de entrenamiento, y D_p el conjunto de prueba que se reserva para hacer las pruebas de generalización, calculando $\rho_\ell(D_p,f)$, donde $f = g(D_e)$. Este método se denomina **holdout validation** (validación con retención).

Se puede considerar entonces a D_p como una muestra de $\mathcal{P}$ *limpia*, es decir, independiente de D_e y representativa de $\mathcal{P}$, que permite estimar el error esperado del clasificador entrenado f en $\mathcal{P}$, $E_x(\ell(x,f))$.

Entonces, dada una medida de error ρ_ℓ, se define formalmente el concepto de un modelo bien entrenado y de generalización.

- Un modelo f bien entrenado es aquel para el cual $\rho_\ell(D_e,f) \simeq 0$.
- Un modelo f que generaliza es aquel para el cual $\rho_\ell(D_e,f) \simeq \rho_\ell(\mathcal{P},f)$. En la práctica el segundo término es imposible de conocer, y por eso se estima como $\rho_\ell(D_e,f) \simeq \rho_\ell(D_p,f)$.

Nuestro ejemplo de la función f_D es un caso en el cual el modelo estaría bien entrenado, ya que $\rho_\ell(D_e,f_D) = 0$, pero claramente $\rho_\ell(\mathcal{P},f_D)$ resulta altísimo. Por ende, ambas condiciones son necesarias para un clasificador efectivo.

2.1.7. Regularización y *overfitting*

Como ya se mencionó, hacer un experimento de clasificación involucra estimar el error de f en $\mathcal{P}$ usando D. Para eso divide D en D_p y D_e, y se mide el error en D_p. Si dicho error es aceptable, se considera que el clasificador es adecuado para la tarea. Ahora bien, si el error es muy grande puede ser que el clasificador no sea adecuado para la tarea, no se haya entrenado lo suficiente, o los datos sean insuficientes o de

mala calidad. Si además se mide el error de f en D_e y también es alto, se obtiene una confirmación de ello. Pero si f se comporta bien con D_e, existe otra razón posible: el clasificador se adaptó tanto a los ejemplares de D_e que no puede generalizar cuando se presentan los de D_p.

Entonces, si un clasificador se especializa tanto en el conjunto de ejemplares con el cual fue entrenado que pierde o no adquiere la capacidad de generalizar, es decir, clasificar correctamente ejemplares no vistos, ejemplares de $\mathcal{P}$ en general, se dice que sufre del fenómeno de **overfitting** (sobre-especialización) que se da cuando un clasificador tiene un nivel de error significativamente menor en el conjunto de entrenamiento que en nuevos ejemplares del problema. Esto puede suceder cuando un clasificador se entrena de forma excesiva o utilizando métodos que no se adaptan bien al dominio del problema.

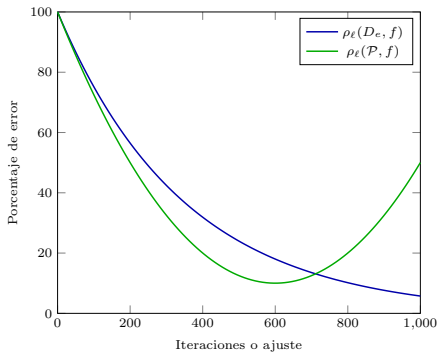
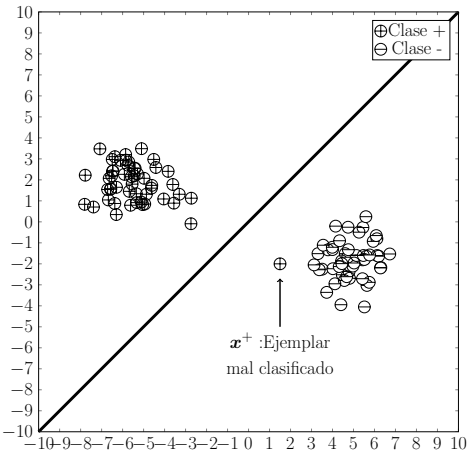
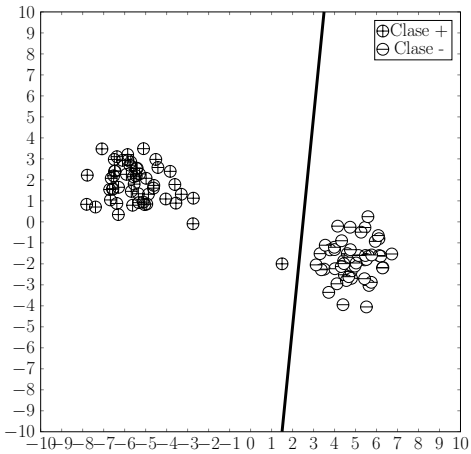


Figura 2.10: Curvas típicas de: $\rho_\ell(D_e, f)$ y $\rho_\ell(\mathcal{P}, f)$ en función de la cantidad de iteraciones de entrenamiento o fuerza del ajuste de f con D. El error se puede decrementar arbitrariamente para el conjunto de entrenamiento, pero llega un momento donde el entrenamiento extra incrementa el error sobre el dominio del problema $\mathcal{P}$ (mejor visto en color).

En este sentido, si bien es importante entrenar un clasificador adecuadamente para minimizar el error en D_e, un entrenamiento que ajuste f demasiado a D_e traerá un error mayor en D_p.



Conjunto de datos D con dos clases, donde hay un outlier. Si bien el hiperplano no clasifica correctamente todos los ejemplares, provee una separación coherente entre clases.



Mismo D, con un hiperplano sobre-entrenado: clasifica bien todas las instancias de D pero no refleja la distribución real de las clases.

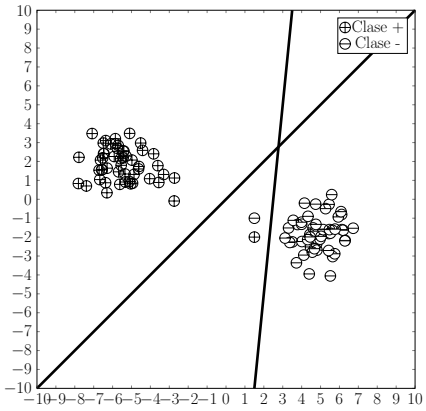
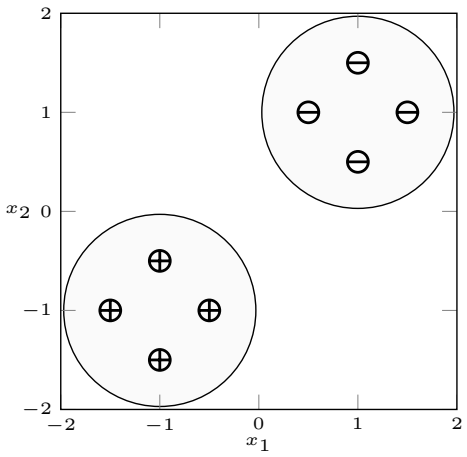
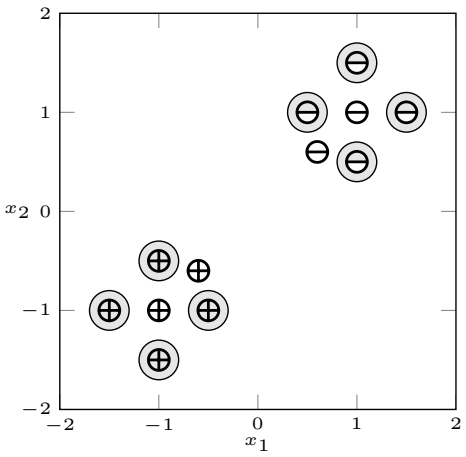


Figura 2.11: Comportamiento de ambos clasificadores ante un nuevo ejemplar.

El problema del overfitting también se da, en ocasiones, cuando la potencia del clasificador utilizado excede la necesaria para resolver un problema dado. En este caso, puede que el mismo se ajuste tan bien a los datos de entrenamiento que no pueda lidiar con desviaciones mayores de los mismos, como se muestra en la figura.



Distribución real de ejemplares por clases



Distribución aprendida por el clasificador. Las regiones de Voronoi se ajustan tan perfectamente a cada ejemplar que no consideran el espacio entre ejemplares de una clase como de la misma clase.

En ocasiones, lo que se necesita es limitar el poder del clasificador. En la figura anterior, se consideró el poder computacional del clasificador como bastante alto ya que debe de alguna manera recordar o incluir en su modelo todos los ejemplares usados para el entrenamiento de forma bastante específica. Si se limita la cantidad de ejemplares a recordar en el clasificador en el caso anterior, se podría obtener una clasificación mejor:

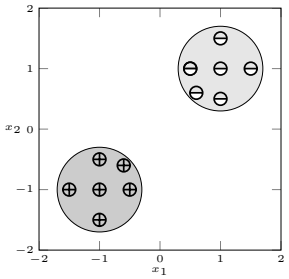


Figura 2.12: Distribución aprendida por el clasificador con capacidad limitada. Hay menos regiones de Voronoi que en el primer caso, pero son más grandes y menos específicas, obteniendo una clasificación más acertada.

En este caso podemos considerar la cantidad de ejemplares en los que basarse como un parámetro del modelo. Este método de limitación del poder computacional se conoce como un esquema de **regularización**, y sigue el principio de la navaja de Occam: si dos modelos explican un fenómeno, por principio se elige el más simple de los dos. Dicho esquema puede estar basado en la selección de algún parámetro del entrenamiento del clasificador, o puede estar incluido directamente mediante un

proceso de entrenamiento que penalice clasificadores complejos en preferencia de los más simples. En el último caso, el esquema general para entrenar un clasificador podría ser:

$$\underset{f=g(D_e)}{\text{Minimizar}} \quad error(f, D_e) + complejidad(f)$$

Donde *complejidad* es alguna función que mide la complejidad de f, dependiente del algoritmo de entrenamiento del clasificador utilizado. Veremos un ejemplo de regularización en la 2.2 de redes neuronales.

En otros casos, cambiar a un modelo de clasificador más simple que se adapte mejor al problema puede solucionar el problema.

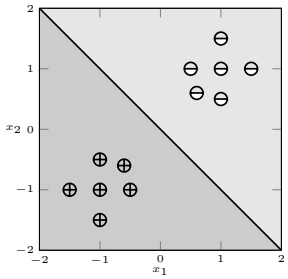


Figura 2.13: Distribución aprendida por un clasificador basado en hiperplanos.

De todas maneras, el clasificador con hiperplanos gana en simplicidad porque implícitamente está haciendo una hipótesis (fuerte) de que la mitad de $\mathbb{R}^2$ pertenece a una clase, y la otra mitad a otra; es decir, para todo punto en el espacio de ejemplares, hay una clase correspondiente. El modelo de hiperesferas, por otro lado, sólo asigna una clase a los puntos cercanos a los conocidos durante el entrenamiento.

2.1.8. Resumen

En esta sección, presentamos los conceptos básicos de aprendizaje automático para el resto de la tesis. Mediante un ejemplo de clasificación de manos, de interés en esta tesis, introdujimos los conceptos generales del área. A su vez, se hace referencia a la gran cantidad de aplicaciones de estas técnicas y su importancia para todas las áreas.

Luego distinguimos entre el aprendizaje supervisado y el no supervisado, y en particular, un modelo supervisado de clasificación de ejemplares, del tipo que utilizamos en los experimentos. La sub-secciones sobre experimentos, generalización, sobre-entrenamiento y regularización describen conceptos claves para poder entrenar un modelo con éxito, es decir, un modelo que tenga poco error.

Estos conceptos sientan las bases teóricas para los siguientes capítulos. A continuación, se construye sobre dichos conceptos para presentar los modelos particulares de Redes Neuronales (sección 2.2).

2.2. Redes Neuronales

Las redes neuronales son modelos de aprendizaje automático[5]. Desde su introducción en 1970 han sufrido varios cambios e iteraciones hasta llegar a los modelos actuales [GBC16]. En esta sección realizamos una breve introducción a las redes neuronales desde un enfoque moderno, haciendo énfasis en problemas de clasificación (aprendizaje supervisado) y en redes convolucionales[6].

2.2.1. Introducción

La versión más moderna de una red neuronal puede considerarse como un grafo de computación dirigido. Este grafo de computación está compuesto de nodos o capas[7] donde cada capa representa una operación, como la suma de dos números, una multiplicación, una exponenciación, etc. Cada capa tiene un conjunto de entradas y salidas determinada (Figura 2.14).

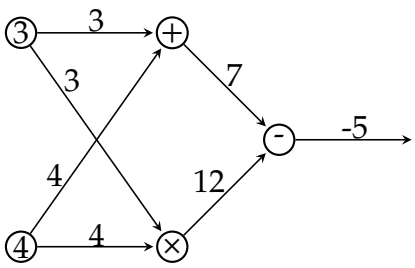


Figura 2.14: Grafo de computación con valores escalares.

En cada entrada la capa recibe un tensor n dimensional de números reales, como un vector, una matriz, o simplemente un escalar. En base a esas entradas la capa calcula un tensor de salida (único), y ese tensor se transmite por todas las aristas de salida. En varios casos el tensor puede ser simplemente un escalar (tensor 1D de tamaño 1) o un vector 1D (Figura 2.16).

No obstante, las capas o nodos se pueden definir a varios niveles de abstracción, y pueden componerse para formar otras capas. De esta forma, generalizamos la entrada a una capa o nodo, la cual puede definirse en términos de la dimensión del

[5]Al ser relativamente novedoso, el campo de Redes Neuronales, y en especial de las Redes Neuronales Profundas, utiliza muchos términos en inglés que aún no tienen una traducción universal al castellano. Por ende, hemos preferido mantener varios términos o siglas en inglés en lugar de traducirlos al español para evitar confusiones.

[6]Para más detalles, recomendamos el libro de Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. MIT Press, 2016.

[7]En otros textos podemos encontrar una diferenciación entre estos conceptos, en donde las capas están compuestas por varias *neuronas* o nodos, donde estos dos últimos conceptos se usan como sinónimos. No obstante, en este texto consideraremos el concepto de nodo como sinónimo de *capa*, en línea con una visión más genérica de los modelos de redes neuronales.

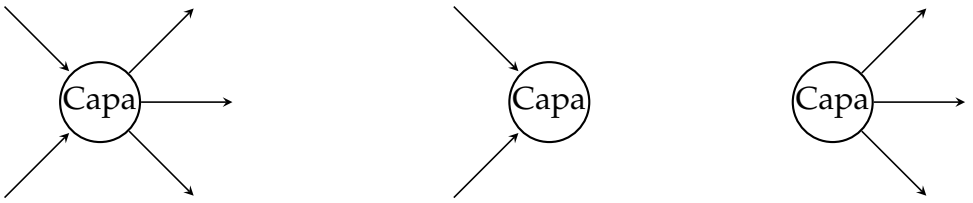


Figura 2.15: (a) Un nodo con entradas y salidas. (b) Un nodo sin salidas. (c) Un nodo sin entradas.

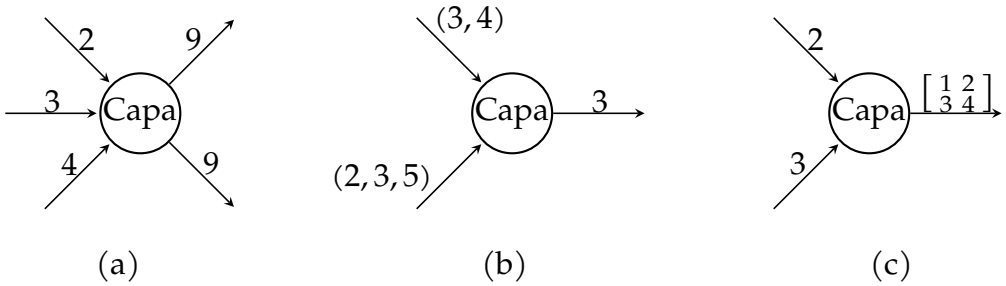


Figura 2.16: (a) Capa que recibe tres escalares y genera un escalar como salida, que se conecta con dos capas. (b) Capa con dos vectores de entrada de tamaños 2 y 3 y un escalar como salida. (c) Capa con dos escalares de entrada y una matriz como salida.

tensor de entrada en base a la necesidad. Por ejemplo, podemos considerar que una capa tiene tres entradas, correspondiente a tres valores escalares distintos, o una sola entrada, correspondiente a un vector de 3 valores (Figura 2.17).

Si bien las capas pueden clasificarse por el tipo de operación que realizan (suma, resta, etc), otra manera usual de categorizarlos es en términos de capas de entrada, de salida y capas ocultas (Figura 2.18). Las capas de entrada son aquellas que no tienen aristas entrantes. De forma simétrica, las de salida son aquellas que no tienen aristas de salida. Las capas ocultas o intermedias permiten factorizar y modularizar

Figura 2.17: (a) Una capa con tres entradas escalares (b) Una capa equivalente con un vector de entrada de 3 elementos

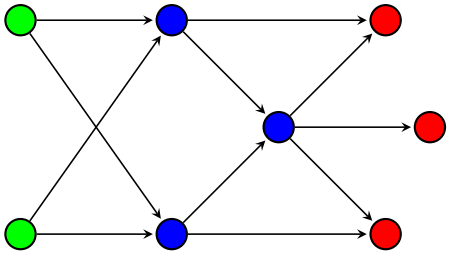
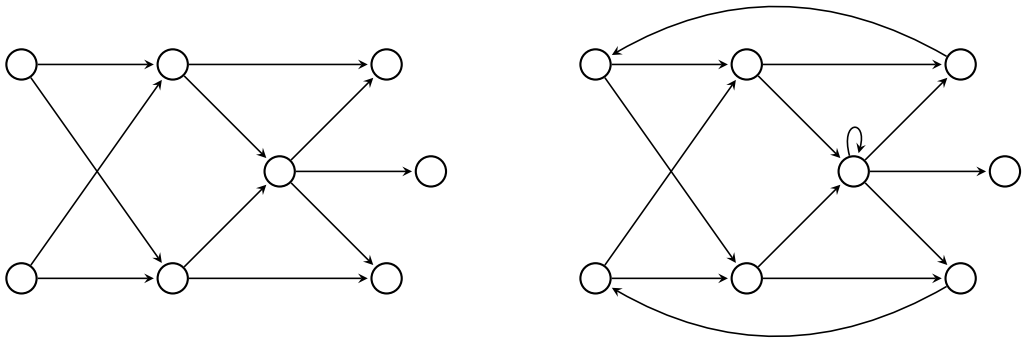


Figura 2.18: Red neuronal con capas de entrada (verde), capas intermedias (azul) y capas de salida (rojo).



(a) (b)

Figura 2.19: (a) Una red acíclica. (b) Una red con ciclos (recurrente)

el cálculo de la salida a partir de la entrada, permitiendo que la red sea eficiente y tenga un buen desempeño. En general, las capas de entrada no computan ningún valor; simplemente se les puede asignar un valor, que propagan hacia las otras capas. En cambio, de capas de salida se puede leer el resultado del cómputo.

2.2.2. Topología de la red

La topología de la red o el grafo no tiene restricciones a priori. De hecho, una clase particular de redes neuronales llamadas redes recurrentes permite utilizar hasta grafos con ciclos (Figura 2.19). No obstante, en lo siguiente nos limitamos a redes acíclicas, ya que son las que utilizamos en esta tesis.

La red se diseña para realizar una tarea, codificada en la función final que calcula la red. El tipo de tarea de la red está determinada por la estructura de sus entradas y salidas. No obstante, qué tan bien la realiza depende de su topología interna, es decir, la estructura de las operaciones en el grafo.

La topología *alimentada hacia adelante* o *feedforward* (Figura 2.20) es la más simple y está compuesta por una serie de capas. El concepto de *alimentar* a una capa o red es equivalente a poner un valor en su entrada. Las capas intermedias siempre tienen

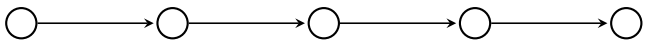


Figura 2.20: Red neuronal con topología feedforward. Las capas intermedias se conectan en serie.

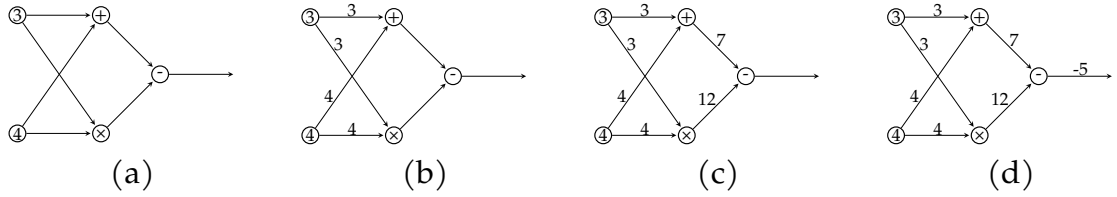


Figura 2.21: Propagación hacia adelante de los valores desde las capas de entrada hacia las de salida de la red.

una entrada y una salida, salvo por las capas de entrada y salida. De hecho, el nombre de *capa* se origina en estos modelos en donde los nodos se consideran sucesivas capas de cómputo. Varios modelos de redes que veremos más adelante utilizan esta topología.

2.2.3. Propagación hacia adelante o evaluación de la red

En el caso de las redes sin ciclos, el cómputo de la salida se realiza de forma discreta, ordenando de forma topológica las capas del grafo. A las capas de entrada se les asigna el valor de entrada deseado, y se propagan los valores del cómputo hasta calcular las salidas. Este proceso se conoce como *propagación hacia adelante* (Figura 2.21) y permite evaluar las salidas de la red.

A continuación, describimos las dos capas básicas para crear una red neuronal: las lineales, y las funciones de activación.

2.2.4. Capas Lineales

Las capas lineales fueron uno de los primeros tipos de capas utilizados. Su entrada es un vector 1D, y como su nombre indica calculan una transformación lineal del mismo, donde la matriz de coeficientes de la transformación w es un parámetro de la red, generando así un vector de salida. Es decir, la capa computa la función $f(x) = wx$. La matriz de coeficientes también se denomina la matriz de pesos o simplemente los pesos de la capa.

En términos de dimensiones, si la entrada es un vector de n dimensiones, y la salida es un vector de m dimensiones, entonces $w \in R^{n \times m}$, es decir, w es una matriz de números reales de tamaño n por m. Estas dimensiones se eligen en base a la tarea

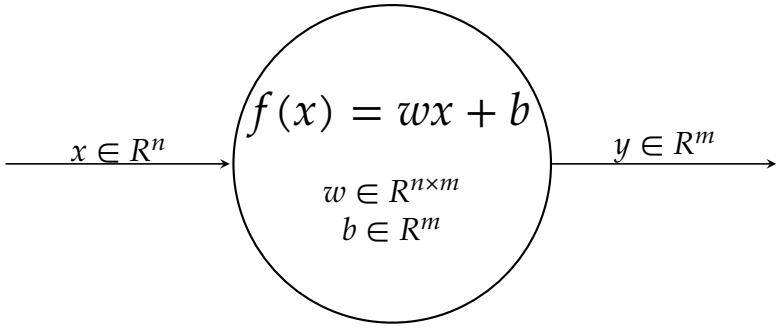


Figura 2.22: Capa lineal, con sus dimensiones de entrada (x) y de salida$(y = f(x))$, y la de sus parámetros.

a realizar.

Las capas lineales suelen llamarse así, pero en general calculan una función afín. Una función afín es simplemente una función lineal a la cual se le suma un coeficiente de sesgo b, de modo que se definen como $f(x) = wx + b$ (Figura 2.22). Estos coeficientes de sesgo se organizan en un vector de m valores, que permiten cambiar la escala de la salida de la capa.

Las capas lineales funcionan como bloques básicos de la red, y permiten realizar aproximaciones lineales mediante el ajuste de sus parámetros.

Por otro lado, desde una perspectiva geométrica las capas lineales definen un hiperplano en n dimensiones. Este hiperplano permite separar el espacio en dos partes, asignando un valor positivo o negativo a los ejemplos de acuerdo al *lado* del hiperplano en el que se encuentren. Una capa lineal con m valores de salida permite trazar m hiperplanos en un espacio n dimensional, con lo cual hay $\sum_{i=0}^{n} \binom{m}{i}$ posibles regiones del espacio.

La derivada de las capas lineales respecto de los parámetros y su entrada es simple y eficiente para calcular (Ecuación [2.3]). Esto resultará de gran utilidad luego para el entrenamiento de la red.

$$\begin{aligned} \frac{\partial wx + b}{\partial w} &= x \\ \frac{\partial wx + b}{\partial b} &= [1, \dots, 1] \\ \frac{\partial wx + b}{\partial x} &= w \end{aligned} \qquad [2.3]$$

Por estos motivos, las capas lineales son muy potentes para representar funciones. Al aplicar varias capas lineales, se puede transformar la entrada paulatinamente hasta obtener el resultado deseado, según la tarea a resolver. No obstante, la composición de dos o más capas lineales de forma consecutiva sufre de una limitación. Al

componer dos capas lineales directamente, las mismas *colapsan* en una sola capa. Si $f_1(x) = w_1x + b_1$ es una capa lineal y $f_2(x) = w_2x + b_2$ es otra capa lineal, entonces $f_2(f_1(x))$ también es una capa lineal (Ecuación [2.4])

$$\begin{aligned} f_2(f_1(x)) &= w_2(w_1x + b_1) + b_2 \\ &= w_2w_1x + w_2b_1 + b_2 \\ &= wx + b \\ \text{donde} \quad b = w_2b_1 + b_2 \quad &\text{y} \quad w = w_2w_1 \end{aligned} \tag{2.4}$$

Es decir, dos capas lineales consecutivas no tienen mayor poder de expresión que una sola capa lineal. Por ende, al componer capas lineales para generar una red suelen utilizarse también funciones no lineales, llamadas *funciones de activación*.

2.2.5. Funciones de Activación

Las funciones de activación permiten modificar los valores en la red de una forma no lineal. Estas se colocan generalmente después de cada capa lineal, de modo que la composición final no se colapse en una sola linealidad.

Las más conocidas son:

1. Logística o Sigmoide
2. Tangente hiperbólica (TanH)
3. Lineal rectificada (ReLU, por *Rectified Linear Unit*)
4. Lineal rectificada paramétrica (PReLU, por *Parametric Rectified Linear Unit*)
5. Lineal rectificada con perdida (LeakyReLU, por *Leaky Rectified Linear Unit*)
6. Exponencial lineal (ELU, por *Exponential Linear Unit*)

La tabla 2.1 muestra varias funciones activación conocidas, y la Figura 2.23 presenta gráficos de las mismas.

Las funciones de activación obtienen su nombre de su capacidad para activar o desactivar su entrada. En este contexto, generalmente activar se refiere a tener un valor alto, y desactivar a tener un valor bajo, o en ocasiones negativo. De esta forma la red puede codificar la presencia o ausencia de características en la entrada. En general, admiten cualquier número real como entrada, pero su rango o imagen están limitados.

Función	Definición	Rango	Derivada
Sigmoidea	$sig(x) = \frac{1}{1+e^{-x}}$	$[0,1]$	$sig(x)(1-sig(x))$
TanH	$tanh(x) = 2\frac{1}{1+e^{-2x}} - 1$	$[-1,1]$	$1 - tanh(x)^2$
ReLU	$relu(x) = max(0,x)$	$[0,\infty]$	$\begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{d.l.c} \end{cases}$
LeakyReLU	$lrelu(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{d.l.c} \end{cases}$	$[min(0,\alpha),\infty]$	$\begin{cases} 1 & \text{si } x > 0 \\ \alpha & \text{d.l.c} \end{cases}$
PReLU	$prelu(x) = \begin{cases} x & \text{si } x > 0 \\ ax & \text{d.l.c} \end{cases}$	$[min(0,a),\infty]$	$\begin{cases} 1 & \text{si } x > 0 \\ a & \text{d.l.c} \end{cases}$
ELU	$elu(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha(e^x - 1), & \text{d.l.c} \end{cases}$	$[-1,\infty]$	$\begin{cases} 1 & \text{si } x > 0 \\ \alpha e^x, & \text{d.l.c} \end{cases}$

Tabla 2.1: Tabla de las funciones de activación Sigmoidea, TanH, ReLu, LeakyReLU, PReLU y ELU.

Por ejemplo, la función de activación Sigmoidea tiene como salida números entre 0 y 1. Además, convierte los valores negativos a números entre 0 y 0.5, y los números positivos a números entre 0.5 y 1. De esta forma, combinada con una capa lineal, permite codificar con valores cercanos a 1 a los ejemplos que se encuentran de un lado del hiperplano de aquella capa, y con valores cercanos a 0 a los del otro lado. Por último, la salida también puede interpretarse como una probabilidad, por ejemplo, de la presencia de una característica.

Por otro lado, la función *TanH* cumple un objetivo similar, pero codifica los valores negativos de la entrada con valores cercanos a -1. La función *ReLU* codifica los valores negativos con 0, y no modifica a los valores positivos; de esta forma, permite desactivar de manera similar a una Sigmoidea, pero no restringe a los positivos.

La estabilidad numérica y eficiencia para el cálculo son importantes para la función activación. Pero además, la existencia y simplicidad de la derivada de estas funciones (tabla 2.1, columna 4) también es de suma importancia, ya que, como veremos más adelante, dichas derivadas serán de suma utilidad para entrenar la red.

Por último notamos que algunas funciones como la *ReLU* no son derivables en todo su dominio. No obstante, estas funciones se utilizan de todas formas ya que se define arbitrariamente el valor de la derivada para los puntos donde no es derivable y esto no presenta problemas en la práctica.

Las funciones de activación en general no tienen parámetros, es decir, su funcionamiento es fijo. La función PReLU es una excepción a esta regla, ya que en su caso α es un parámetro entrenable que le permite aprender la pendiente para las entradas

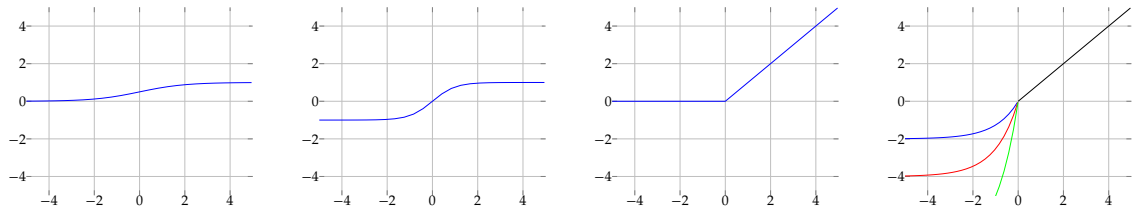


Figura 2.23: Gráfico de las funciones de activación (a) Sigmoidea, (b) TanH, (c) ReLU, y (d) ELU (los colores azul, rojo y verde denotan valores de α de 2, 4 y 8 respectivamente.).

$$f\left(\begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{bmatrix}\right) = \begin{bmatrix} f(x_{1,1}) & f(x_{1,2}) \\ f(x_{2,1}) & f(x_{2,2}) \end{bmatrix}$$

Figura 2.24: Aplicación de una función de activación a un tensor. La función se aplica a cada uno de los elementos del tensor por separado.

negativas. Las funciones de activación se aplican generalmente a cada valor de un tensor. Por ejemplo, si se aplica una función de activación a una matriz de 5×4, la función se aplicará 20 veces, una vez por cada valor escalar (Figura 2.24).

Las redes formadas con al menos dos capas lineales y funciones de activación no lineales son aproximadores universales y por ende pueden representar cualquier función continua con un error arbitrariamente pequeño usando una cantidad de parámetros acorde. Esta arquitectura entonces es sumamente potente para entrenar modelos de aprendizaje automático.

Por otro lado, el uso de capas extra permite que la aproximación sea más eficiente, ya que se utilizan varias transformaciones sucesivas. Si consideramos cada una como un *paso* de un algoritmo, entonces una red de varias capas se asemeja a un algoritmo entrenable, que se determina de forma automática en base a los datos. Es por esto que en los últimos tiempos se tiende a utilizar redes con gran cantidad de capas, dando lugar a las Redes Neuronales Profundas o *Deep Learning*.

A continuación, presentamos la función Softmax, que permite convertir las salidas de la red en distribuciones de probabilidad.

Función de activación Softmax

La función de activación Softmax es un caso particular ya que no actúa elemento por elemento. Es un caso particular de función de activación ya que generalmente sólo se utiliza en la salida de la red en problemas de clasificación.

Permite convertir un vector de valores con rango $(-\infty, +\infty)$, considerados como

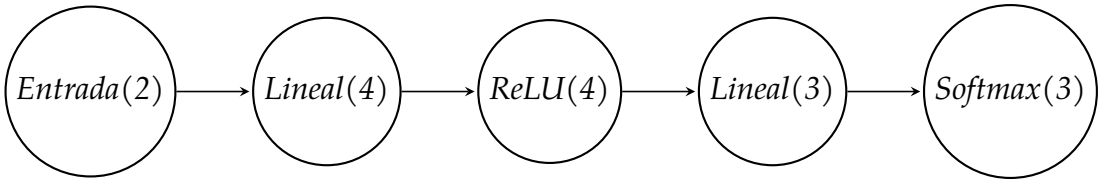


Figura 2.25: Grafo de la red $Entrada(2) \rightarrow Lineal(4) \rightarrow ReLU(4) \rightarrow Lineal(3) \rightarrow Softmax(3)$.

$$Entrada(2) \rightarrow Lineal(4) \rightarrow ReLU(4) \rightarrow Lineal(3) \rightarrow Softmax(3)$$

Figura 2.26: Notación abreviada para redes neuronales feedforward.

puntuaciones, en un vector con una estructura de distribución de probabilidad. Es decir, los valores del vector resultante cumplen los axiomas de una distribución de probabilidad: son mayores a cero, y suman a uno.

Para ello, la Softmax calcula la exponencial de cada uno de los valores del vector x por separado, y luego normaliza los valores resultantes con la suma de todos ellos (Ecuación [2.5]).

$$\begin{aligned} \text{Softmax}(x) = \text{Softmax}([x_1, \dots, x_n]) &= [\frac{e^{x_1}}{N(x)}, \dots, \frac{e^{x_n}}{N(x)}] \\ N(x) = N([x_1, \dots, x_n]) &= \sum_{i=1}^{n} e^{x_i} \end{aligned} \quad [2.5]$$

De esta forma, la red puede convertir un vector de valores arbitrarios en un vector de probabilidades, lo cual es útil para problemas de clasificación, ya que cada valor de salida puede representar la probabilidad de una clase.

2.2.6. Ejemplo de red

En base a lo visto, podemos considerar una red neuronal de ejemplo con topología de feedforward (Figura 2.25). La entrada a la red es un vector con 2 elementos. La red está compuesta por una capa lineal de dimensión 4, una función de activación *ReLU*, otra capa lineal de dimensión 3 y finalmente una función Softmax.

Podemos describir dicha red de acuerdo a la notación de la Figura 2.26, donde el número entre paréntesis indica la dimensión de salida de la capa. En este caso incluimos explícitamente la capa de entrada, en otras ocasiones simplemente omitiremos esa información dado que puede deducirse de la descripción de los datos a utilizar.

$[0, 1, 0, 0]$ (a) $[0.3, 0.6, 0.03, 0.07]$ (b)

Figura 2.27: (a) Salida esperada para la clase 2 en un problema de clasificación con 4 clases (b) Posible salida de la red para el mismo problema, con las probabilidades por clase.

2.2.7. Perspectiva de grafo y de función de la red.

Si bien el concepto de grafo nos permite diseñar la red, también podemos considerarla simplemente como una (complicada) función f, tal que si x es la entrada a la red, f nos permite calcular la salida y, o sea $y = f(x)$.

Por ejemplo, para la red anterior de la Figura 2.25 podemos escribir la función resultante que calcula. Si w_1 y b_1 son los pesos de la primera capa lineal, y w_2 y b_2 los de la segunda, la Ecuación [2.6] detalla la función f equivalente que calcula la red:

$$f(x) = \text{Softmax}(w_2 \,\text{ReLU}(w_1 x + b_1) + b_2) \qquad [2.6]$$

2.2.8. Codificación de la salida de la red

La tarea que realiza una red neuronal está determinada por la salida de la misma. Como mencionamos anteriormente, la función de activación Softmax permite codificar la salida para tareas de clasificación. En este caso, si el problema es predecir una de C clases posibles, entonces la salida será un vector de C elementos y la última capa será una Softmax. Por ende, la salida verdadera de los ejemplos de la base de datos también se codifica como un vector y de C elementos. Dado que para estos elementos conocemos con total seguridad su clase, el vector y contiene todos valores 0, excepto por el elemento cuyo índice correspondiente a la clase, que tendrá valor 1. Es decir, si un ejemplo tiene clase c entonces su salida será $y = [y_1, \ldots, y_c, \ldots, y_C] = [0, \ldots, 1, \ldots, 0]$ donde el valor 1 se encuentra en el índice c (Figura 2.27).

2.2.9. Funciones de error para un ejemplo

Para saber qué tan bien una red realiza una tarea de aprendizaje supervisado, podemos definir una función de error que compare los valores que produce la red $f(x)$ con el valor verdadero o esperado y. Es decir, buscamos una función de error $e(f(x), y)$ que pueda cuantificar el error de la red en un ejemplo particular.

Si una red tuviese varias salidas, se requeriría una función de error por cada una

de ellas; en lo siguiente asumiremos que la red tiene una sola salida y por ende hablaremos de una sola función de error.

Las funciones de error tienen como salida un valor escalar, que cuantifica el error de un ejemplo particular. La función suele cumplir que vale 0 en el caso en que $f(x) = y$, y tiene un valor mayor a 0 de lo contrario. Dependiendo de la tarea de la red, la magnitud de este error puede variar, debido a que e siempre está asociada a los valores que produce la capa de salida de la red. Es decir, la función e debe definirse en conjunto con la red, para que la cuantificación del error tenga sentido.

Función de error cuadrático

El error cuadrático es uno de los más simples, y se basa en calcular la distancia euclidiana al cuadrado entre la salida de la red $\hat{y} = f(x)$ y la salida esperada y. De esta forma, si $\hat{y}$ e y son vectores con d elementos, entonces la Ecuación [2.7] define la función de error ErrorCuadratico$(f(x), y)$ como:

$$\begin{aligned} \text{ErrorCuadratico}(f(x), y) &= \text{ErrorCuadratico}(\hat{y}, y) \\ &= |\hat{y} - y|^2 \\ &= \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \end{aligned} \qquad [2.7]$$

La función de error cuadrático suele asociarse a redes que no tienen una función activación después de la última capa lineal, es decir, cuya salida es la salida de una capa lineal. De esta forma, la salida de la red puede ser un vector con valores reales arbitrarios. También puede utilizarse con redes cuya última capa utiliza una función activación *ReLU*, por ejemplo en dominios donde la salida siempre debe ser positiva.

Función de error de Entropía Cruzada

En el caso de una red que se utiliza para clasificación, lo más común es que la capa final sea de una función de activación Softmax, que genera un vector de C elementos representando las probabilidades de clase. En este caso, existe una función de error llamada entropía cruzada (EntropiaCruzada), que permite comparar el vector de la salida verdadera y de C elementos, con $f(x)$. La EntropiaCruzada es una medida de distancia entre distribuciones de probabilidad, y nos permite comparar la generada por la red $(f(x))$ con la esperada (y). La Ecuación [2.8] es la definición general de la entropía cruzada entre una distribución fija y y la distribución generada por la red $\hat{y} = f(x)$

$$\begin{aligned} \text{EntropiaCruzada}(f(x), y) &= \text{EntropiaCruzada}(\hat{y}, y) \\ &= -\sum_{i=1}^{d} y_i \log(\hat{y}_i) \end{aligned} \quad [2.8]$$

En el caso particular de un problema de clasificación, el vector y solo tiene un valor distinto de 0. Sea c el índice de dicho valor, que también es el índice de la clase, por lo cual $y_c = 1$. Podemos simplificar entonces la definición de la EntropiaCruzada, dejando sólo uno de los términos (Ecuación [2.9]).

$$\begin{aligned} \text{EntropiaCruzada}(f(x), y) &= \text{EntropiaCruzada}(\hat{y}, y) \\ &= -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \\ &= -y_c \log(\hat{y}_c) \\ &= -\log(\hat{y}_c) \end{aligned} \quad [2.9]$$

La EntropiaCruzada entonces funciona bien para evaluar la salida de una función activación Softmax, ya que compara distribuciones de probabilidad, y suelen utilizarse casi siempre de forma conjunta.

2.2.10. Evaluación por lotes

Si bien para evaluar la salida a partir de una entrada es posible hacerlo con un solo ejemplo, esto no es lo más eficiente ya que varias de las operaciones de las capas pueden optimizarse para su cálculo si se emplean varios ejemplos al mismo tiempo.

Por otro lado, las redes neuronales pueden utilizar una gran cantidad de memoria para almacenar parámetros así como los valores intermedios al calcular la salida, por ende la cantidad de ejemplos a utilizar está limitada. En otras palabras, no podemos evaluar todos los ejemplos de una base de datos significativa al mismo tiempo.

Este conjunto de ejemplos se conoce como *lote*. Entonces, las redes se diseñan de forma tal de que reciben un lote de ejemplos como entrada, y generan de forma correspondiente un lote de salidas, donde hay una salida para cada ejemplo.

Por ejemplo, en el caso de las capas lineales, si tenemos un solo ejemplo $x \in R^n$, y $w \in R^{n \times m}$, entonces $xw \in R^m$. Ahora si tenemos un lote de l ejemplos, éstos pueden codificarse en una matriz con l filas y n columnas ($x \in R^{l \times n}$). No obstante, la fórmula de la capa lineal sigue funcionando, pero ahora $xw \in R^{l \times m}$, es decir, tenemos l de salidas de la capa cada una de dimensión m, organizadas en una matriz de $l \times m$ (Figura 2.28).

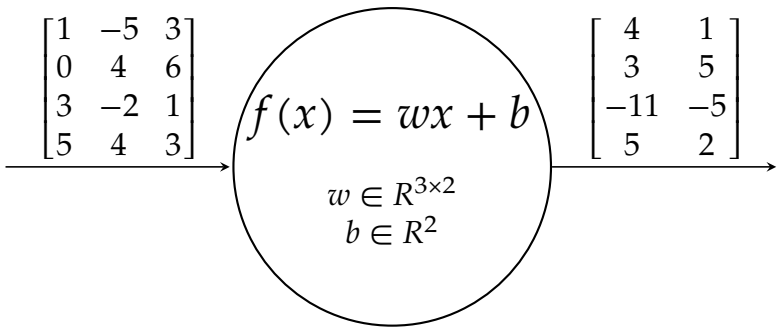


Figura 2.28: Evaluación por lotes de una red con una capa lineal, utilizando 4 ejemplos por lote.

Error promedio por lotes

En el caso de la función de error, podemos también calcular el error de todo un lote. En tal caso, también es frecuente calcular el error *promedio* del lote. Dado un lote de l ejemplos X con sus etiquetas Y, donde $X = [x_1, \dots, x_l]$ e $Y = [y_1, \dots, y_l]$ y una función de error $e(f(x), y)$ para un ejemplo, el error promedio del lote E se define según la Ecuación [2.10]:

$$E(X, Y, e) = \frac{\sum_{i=1}^{l} e(f(x_i), y_i)}{l} \qquad [2.10]$$

En ocasiones, abreviaremos la notación $E(X, Y, e)$ a $E(X, Y)$ o incluso a E, si estos argumentos no son relevantes o están claros dado el contexto.

El error E permite obtener una idea general del error de la red, tanto para evaluarla como para poder minimizar este error durante su entrenamiento, de manera que la red funcione mejor para su tarea. La definición en términos de lotes es flexible, ya que la misma definición sirve también para evaluar el error de todos los ejemplos de una base de datos.

2.2.11. Entrenamiento de la red

Hasta ahora hemos visto los elementos para definir y evaluar una red para una tarea. La función de error de la red depende de 3 cosas:

1. El diseño o la topología de la red.
2. El valor de los parámetros de la red.
3. El valor del conjunto de ejemplos con los que se evalúa el error (X e Y).

El diseño de la red generalmente se realiza de forma manual, con conocimiento experto del dominio a resolver. El objetivo de la red como modelo de aprendizaje automático es aprender a partir de un conjunto de ejemplos; estos ejemplos generalmente están fijos, ya que se obtienen de una base de datos.

En efecto, para que la red *aprenda*, nuestra única opción es cambiar los parámetros de la misma.

La red aprende de los datos al combinarlos junto con la función de error para modificar sus *parámetros* hasta encontrar valores de los mismos que ofrezcan un bajo error de la red. Este proceso se llama optimización o entrenamiento de la red.

El conjunto de parámetros de la red está compuesto por la unión de los parámetros de cada capa, y generalmente se modifican todos ellos para entrenar la red. En lo siguiente, asumiremos una red con L capas, donde los parámetros de cada una se indican como p_i.

Existen varios algoritmos de optimización que pueden utilizarse para entrenar la red. Algunos están basados en metaheurísticas, otros en algoritmos evolutivos, pero los más utilizados emplean algoritmos de optimización basados en la derivada de E, la función de error promedio de la red.

En este caso, nos interesa considerar el error promedio E como una función que depende solamente de los parámetros p_i, ya que tanto los datos (X, Y) como la función error e están fijos durante el entrenamiento. Si bien anteriormente definimos $E = E(X, Y, e)$ como una función que depende de los datos y la función error, y dejaba implícita la dependencia de los parámetros, para lo siguiente, ahora consideraremos a E como una función de los parámetros de modo que si tenemos L parámetros, entonces $E = E(p_1, \dots, p_L)$. Dado que el algoritmo que utilizaremos se basa en las derivadas de los parámetros, un elemento fundamental del mismo será el valor $\frac{\partial E}{\partial p_i}$.

Para el entrenamiento de la red también resulta de utilidad pensar en la función de error simplemente como otra capa de la red, de modo que la salida final de la misma es un número escalar que representa su error.

Entonces para entrenar la red aprovecharemos el hecho de que el grafo es derivable. Esto permitirá calcular las derivadas del error respecto de cada uno de los parámetros p_i con un algoritmo llamado *propagación hacia atrás*. Luego, se pueden utilizar las derivadas para optimizar estos parámetros respecto de E, mediante un algoritmo de optimización llamado *descenso de gradiente* [8]. A continuación, detallamos estos dos algoritmos.

[8] Existen también otros algoritmos de optimización o entrenamiento basados en descenso de gradiente más avanzados, como Descenso de gradiente con Momentum, AdaDelta, ADAM, ADAMW, etc, pero están fuera del alcance de este texto [GBC16].

Figura 2.29: Topología de la propagación hacia adelante con el grafo original (a) y hacia atrás con el grafo invertido (b).

Propagación hacia atrás (*backpropagation*)

La propagación hacia atrás es un algoritmo para calcular las derivadas de los parámetros de la red respecto del error. Dada una red f, un lote de ejemplos X e Y, y una función de error e, el objetivo de la propagación hacia atrás es calcular las derivadas parciales de $E = E(X, Y, e)$ respecto a cada uno de los parámetros de la red.

En la sección 2.2.3 definimos la propagación hacia adelante de la red. El algoritmo de *propagación hacia atrás* obtiene este nombre por su funcionamiento, que comienza con la última capa, la del error, y luego recorre las capas *hacia atrás*, en el orden inverso al topológico. Por ende, el algoritmo utiliza una inversión del grafo original, que era de propagación *hacia adelante* (Figura 2.29).

Para ello, el algoritmo trabaja por capas. Para cada capa i, con parámetros p_i, tensor de entrada e_i y de salida s_i, el algoritmo calcula dos derivadas: $\frac{\partial E}{\partial p_i}$ y $\frac{\partial E}{\partial e_i}$ (Figura 2.30). Para ello, recibe la derivada del error respecto de su *salida*, es decir, $\frac{\partial E}{\partial s_i}$, desde las capas a las cual alimenta, es decir, desde las capas conectadas a sus aristas de salida. Estas derivadas tienen los siguientes objetivos:

- $\frac{\partial E}{\partial s_i}$ se utiliza para calcular las derivadas $\frac{\partial E}{\partial p_i}$ y $\frac{\partial E}{\partial e_i}$ utilizando la regla de la cadena.
- $\frac{\partial E}{\partial p_i}$ es el resultado de este proceso, que luego se utilizará para optimizar la red.
- $\frac{\partial E}{\partial e_i}$ se utiliza para que las capas anteriores también puedan realizar este proceso.

La regla de la cadena es el mecanismo que posibilita la propagación hacia atrás entre las capas. Cada capa i, por ser derivable, puede calcular la derivada de la salida respecto a su entrada y sus parámetros, o sea $\frac{\partial s_i}{\partial e_i}$ y $\frac{\partial s_i}{\partial p_i}$. Por ejemplo, en la sección 2.2.4 vimos justamente como calcular estos valores para las capas lineales. La

(a) (b)

Figura 2.30: Propagación hacia adelante (a) y hacia atrás (b) para una capa arbitraria. Si la capa i tiene parámetros p_i también se calcula $\frac{\partial E}{\partial p_i}$. Este valor no se propaga, pero se almacena para ser utilizado por el proceso de entrenamiento.

sección 2.2.5 muestra las derivadas de las funciones de activación, que no son más que los elementos de $\frac{\partial s_i}{\partial e_i}$; como las funciones de activación no tienen parámetros, no es necesario calcular $\frac{\partial s_i}{\partial p_i}$, salvo por la función *PReLU*.

En cualquier caso, dado $\frac{\partial s_i}{\partial e_i}$ y $\frac{\partial s_i}{\partial p_i}$, podemos utilizar la regla de la cadena dentro de la capa para calcular $\frac{\partial E}{\partial e_i}$ y $\frac{\partial E}{\partial p_i}$ de acuerdo a la Ecuación [2.12] (Figura 2.31).

$$\frac{\partial E}{\partial e_i} = \frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial e_i} \quad [2.11]$$

$$\frac{\partial E}{\partial p_i} = \frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial p_i} \quad [2.12]$$

Para simplificar la presentación, en esta ecuación estamos asumiendo que la capa tiene una sola arista de salida. En el caso general $\frac{\partial E}{\partial e_i}$ y $\frac{\partial E}{\partial p_i}$ pueden calcularse con una sumatoria sobre todas las aristas de salida.

El algoritmo itera por cada una de las capas en orden inverso al topológico, comenzando por las capas de error, que en este caso deberían ser las de salida, y terminando cuando se alcanzan las capas de entrada.

Por último, la derivada parcial $\frac{\partial E}{\partial p_i}$ es un tensor con tantos elementos como p_i. Por ende, el coste de memoria para almacenar todas las derivadas es el mismo que el necesario para almacenar los parámetros de la red.

Descenso de gradiente

El algoritmo de propagación hacia atrás nos ofrece una manera eficiente de calcular las derivadas del error promedio E con respecto de cada uno de los parámetros ($\frac{\partial E}{\partial p_i}$). Con esa información, podremos modificar los parámetros p_i de forma iterativa

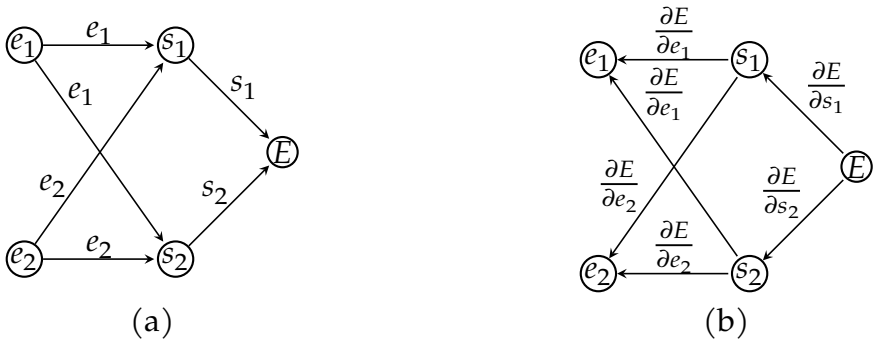


Figura 2.31: Valores de la propagación hacia adelante con el grafo original (a) y hacia atrás con el grafo invertido (b).

hasta encontrar valores de los pesos que tengan un error bajo mediante el algoritmo de *Descenso de gradiente*.

El término *gradiente* técnicamente se define como un tensor de derivadas parciales. En este caso, es equivalente a lo que venimos llamando la derivada del error respecto de los parámetros, con lo cual seguiremos utilizando esta nomenclatura.

El algoritmo de descenso del gradiente utiliza el hecho de que la derivada de una función en un punto indica la dirección hacia la cual moverse a partir de ese punto para incrementar el valor. El *opuesto* de la derivada nos indica la dirección óptima para decrementar el valor de la función.

Podemos visualizar esta situación para una función de error con un parámetro real ($E : R \to R$) mediante la Figura 2.32.

La Figura 2.33 sugiere una analogía importante para comprender el descenso de gradiente. Cada posición en el gráfico corresponde a un vector 2D que contiene una combinación de los valores de los parámetros. La altura en cada posición indica el error de esa combinación de parámetros. A cada posición además corresponde una derivada, que también es un vector 2D, que indica la dirección a tomar para maximizar el error, partiendo de esa posición en particular.

De esta forma, partiendo de un valor inicial de los parámetros p_i, podemos calcular la derivada, y *hacer un paso* en la dirección en la que indica el opuesto de la derivada. Podemos repetir este esquema n veces, haciendo n pasos o *iteraciones*. Repitiendo n pasos o *iteraciones* este proceso, podemos llegar a un conjunto de parámetros finales (Figura 2.34). Como detallaremos más abajo, con una cantidad de iteraciones suficiente, este error generalmente será bajo. El Algoritmo 1 presenta un pseudocódigo del algoritmo de descenso de gradiente. Las líneas 1 y 2 inicializan los pesos con valores aleatorios. La línea 3 genera I pasos del algoritmo. La línea 4 calcula la propagación hacia atrás, obteniendo las derivadas de cada parámetro. Las líneas 5 y

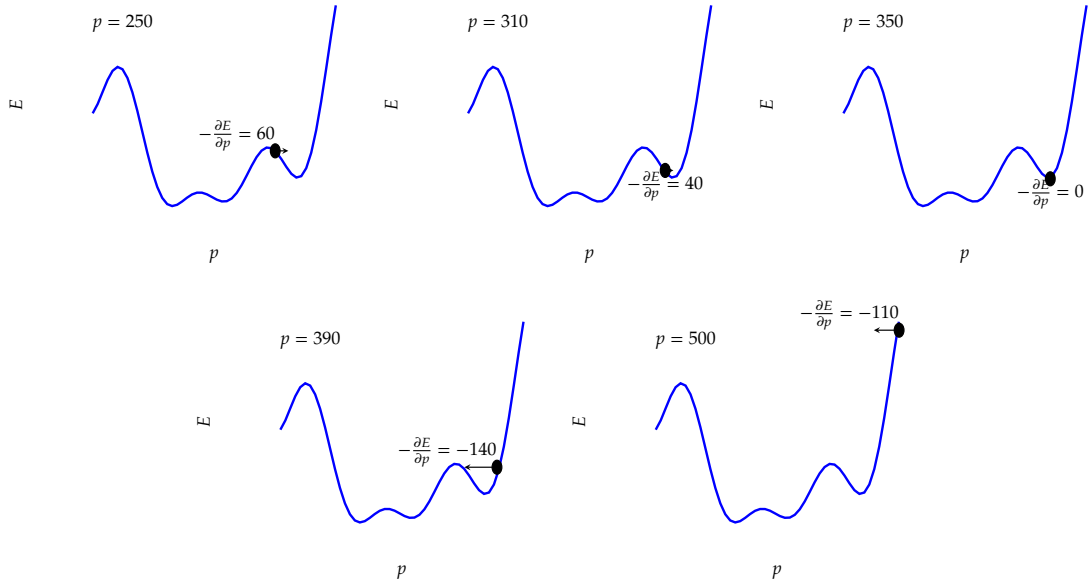


Figura 2.32: Visualización de la función de error $E(p)$ para distintos valores de p. El opuesto de la derivada en cada punto $(-\frac{\partial E}{\partial p})$ indica la dirección para minimizar la función.

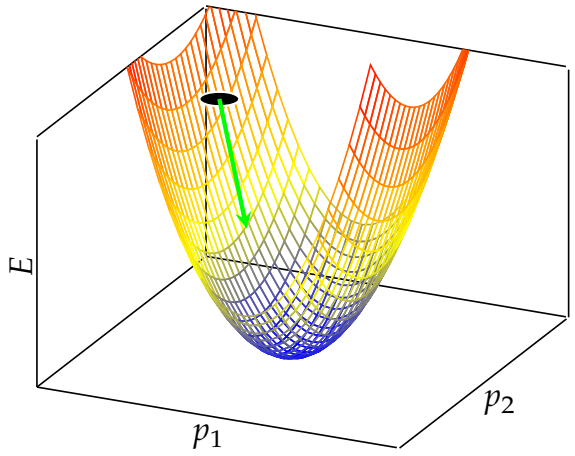


Figura 2.33: Visualización de la función de error E, y la derivada $\frac{\partial E}{\partial (p_1, p_2)}$ en un punto repecto de los dos parámetros (flecha verde). Los ejes x e y indican el valor de los parámetros p_1 y p_2. El eje z indica el valor de la función de error promedio E.

6 mueven los parámetros en la dirección opuesta al gradiente.

Algoritmo 1: Algoritmo de descenso de gradiente

Datos: Conjunto de ejemplos X e Y
Red con L parámetros $p_1, \dots, p_L$
Función de error E
Cantidad de iteraciones I
Resultado: Pesos finales de la red $p_1, \dots, p_L$

```
1 for j ←1 to L:
2     p_j ←random();
3 for i ←1 to I:
4     ∂E/∂p_1, ..., ∂E/∂p_L = propagacionHaciaAtras(E, X, Y, p_1, ..., p_L);
5     for j ←1 to L:
6         p_j ← p_j − α ∂E/∂p_j;
```

Tasa de aprendizaje α En el Algoritmo 1, la línea 6 modifica un peso en base a su gradiente. Dicha línea utiliza una variable α, que se conoce como tasa de aprendizaje, que multiplica la derivada, cambiando su escala. De esta forma, se puede controlar el tamaño de los pasos que realiza el algoritmo, es decir, la magnitud de los cambios de p_i, $|\alpha \frac{\partial E}{\partial p_i}|$.

El gradiente indica la dirección de máximo crecimiento, pero para un valor de p_i particular. Dicha dirección es de máximo crecimiento solo en un contorno infinitesimal alrededor de p_i. Por ende, el tamaño de paso debe ser pequeño para que el algoritmo funcione. Si el tamaño de paso es muy grande, la dirección pierde significado, y el algoritmo dará pasos en falso. Por otro lado, valores de α demasiado pequeños causan actualizaciones pequeñas y el algoritmo requiere más iteraciones. En general, se eligen valores pequeños de α, en el orden de 0.001.

Por último, el valor α es un ejemplo de un hiperparámetro, es decir, un valor que se elige durante el diseño de la red pero no se optimiza con el descenso de gradiente.

Valor inicial de los parámetros El valor final de los parámetros depende de su valor inicial. El valor inicial de los parámetros generalmente se elige de forma aleatoria. De esta forma, podemos repetir el entrenamiento si el conjunto de parámetros finales no tiene un error suficientemente bajo. Existen varios esquemas de inicialización de parámetros, los cuales están fuera del alcance de este texto [GBC16]. En general, estos esquemas asignan valores muy pequeños para los parámetros iniciales, y buscan posicionarlos en un punto del espacio de parámetros bueno, para que el aprendizaje

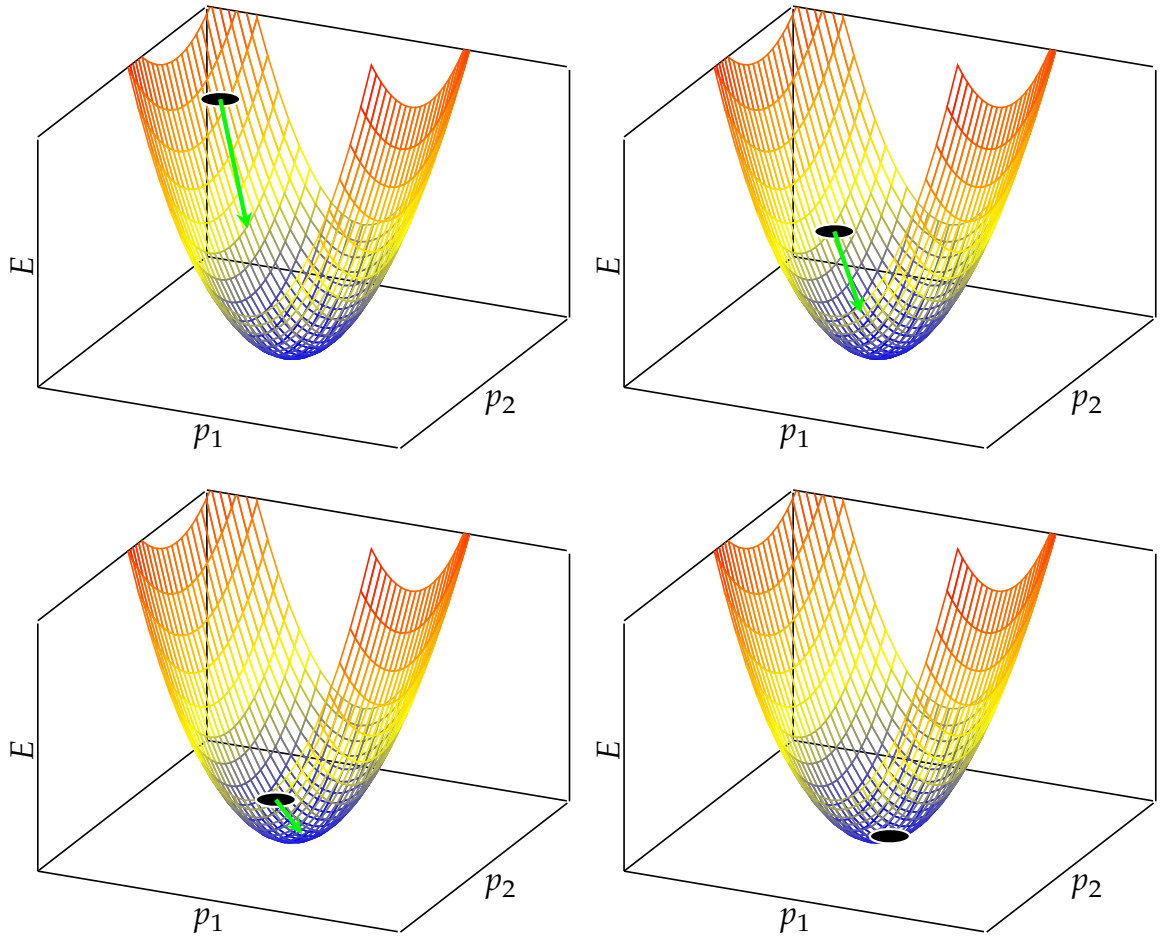


Figura 2.34: Visualización de los pasos del descenso de gradiente hasta su convergencia en un gráfico 3D. Los ejes x e y indican el valor de los parámetros p_1 y p_2. El eje z indica el valor de la función de error promedio E.

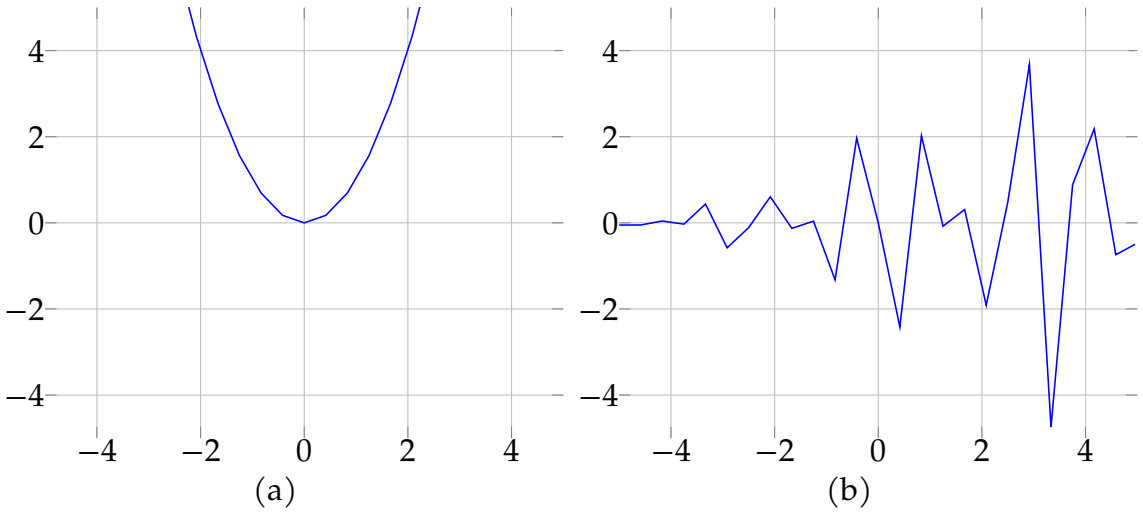


Figura 2.35: (a) Función de error convexa, donde el único mínimo es el local, y las derivadas siempre indican la dirección de este mínimo. (b) Función de error no convexa, con múltiples minimos (locales) y un único mínimo global.

de la red sea corto y alcance un error bajo con los parámetros finales.

Tipos de funciones de error Las funciones de error a optimizar se pueden clasificar en funciones convexas y no convexas (Figura 2.35).

Las primeras resultan fáciles para optimizar ya que tienen un solo mínimo, llamado mínimo global. El entrenamiento utilizando descenso de gradiente y un valor de α suficientemente chico tiene garantizada la convergencia para funciones convexas. De esta forma siempre alcanzan el mínimo global sin importar en qué valores fueron inicializado los parámetros.

Las funciones que no son convexas pueden tener varios mínimos, de los cuales uno o varios serán los mínimos globales, y el resto serán mínimos locales. Los mínimos locales son conjuntos de parámetros cuyo valor de error asociado es menor que el de todo su entorno. En algunos casos, el valor de error es bajo, y por ende el mínimo local es una solución útil. En otros casos, el valor de error es inaceptable, y el mínimo local no es una solución útil. Estos casos suelen denominarse mínimos locales *buenos y malos*, respectivamente (Figura 2.36). En general, los problemas de optimización con mínimos locales son mucho más difíciles de resolver con descenso del gradiente. Este algoritmo depende de la derivada del error, que tiene valor cero en los mínimos. Si la derivada del error es cercana a cero, entonces el descenso de gradiente realizará pasos muy chicos, y por ende el algoritmo puede *atascarse* en un mínimo local *malo*.

La función de error de las redes neuronales casi nunca es convexa. Afortunadamente, al utilizar redes con miles o millones de parámetros, un diseño de red apro-

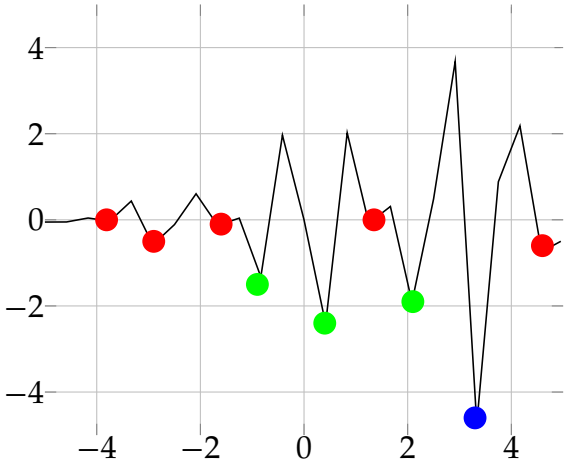


Figura 2.36: Función de error con varios mínimos. El mínimo global (azul) es el mejor error posible. No obstante, podemos considerar que otros mínimos locales también son soluciones aceptables (verde), y mientras otros tienen un valor de error demasiado alto (rojo).

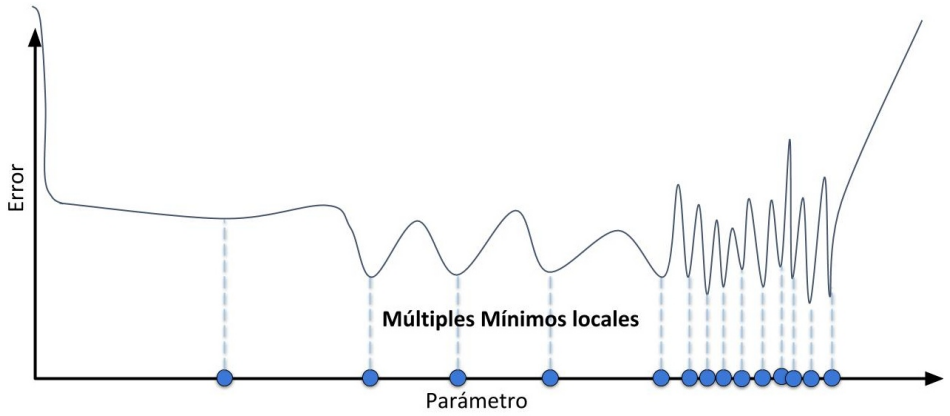


Figura 2.37: Una función de error típica de una red neuronal. Hay varios mínimos locales cuyo valor de error es similar, por ende desde el punto de vista del error son equivalentes.

piado para el problema y una buena inicialización de los parámetros, la mayoría de los mínimos locales que se encuentran durante el entrenamiento son mínimos locales *buenos* (Figura 2.37). Es decir, las funciones de error de las redes neuronales típicamente contienen una gran cantidad de mínimos locales cuyo error es similar, y por ende representan soluciones equivalentes.

Convergencia del descenso de gradiente Si bien las redes neuronales son modelos de aprendizaje automático y por ende rara vez se puede lograr que su error sea cero, se considera que una red converge cuando su error deja de cambiar significativamente.

En el Algoritmo 1 se utiliza una cantidad máxima de iteraciones como criterio de terminación del algoritmo de descenso de gradiente. En varios dominios, la cantidad

iteraciones que requiere el algoritmo para converger se conoce de forma aproximada, y por ende este esquema es suficiente.

En otros casos, no se puede determinar a priori la cantidad aproximada de iteraciones necesarias. Así, otro criterio de terminación posible del algoritmo de descenso de gradiente consiste en monitorear el error de cada iteración, y finalizar el entrenamiento cuando la diferencia entre el error de una iteración y la anterior sea menor a un umbral, que generalmente es un valor pequeño como 0.001.

No obstante, estos criterios son complementarios, y generalmente se combina el criterio del error con una cantidad máxima iteraciones, de modo de limitar el costo computacional máximo del algoritmo. En ambos casos, no obstante, es una buena práctica registrar la evolución del error de la red para cada iteración para comprender su dinámica de aprendizaje y la cantidad de iteraciones que precisa para converger.

Descenso de gradiente estocástico

El algoritmo descenso de gradiente nos permite optimizar la red para un lote de ejemplos (X, Y) fijos. No obstante, si tenemos un conjunto de datos de millones de ejemplos, no será posible propagar por la red todos los ejemplos al mismo tiempo, por lo mencionado en la sección 2.2.10.

Es posible modificar el algoritmo de propagación hacia atrás para funcionar por lotes, de modo que podamos calcular las derivadas de la red respecto al error promedio de millones de ejemplos. Este esquema se conoce como entrenamiento *batch*. No obstante, de esa forma se desperdiciaría mucho poder de cómputo, ya que para realizar el cálculo de las derivadas no es necesario utilizar tantos ejemplos para obtener significancia estadística.

Un esquema más balanceado consiste en realizar descenso de gradiente de forma *estocástica*. En este esquema el entrenamiento se realiza por lotes, de la misma forma que en la evaluación de las redes. El entrenamiento se organiza en épocas; en cada *época*, la red se entrena con todos los ejemplos del conjunto de entrenamiento. Para ello, el conjunto de entrenamiento se divide en lotes, ordenados de alguna forma. Cada lote corresponde a una iteración dentro de la época. En cada iteración entonces se realiza una actualización de los pesos de la red utilizando una aplicación de descenso de gradiente. Entre dos iteraciones consecutivas los pesos se modifican, con lo cual la propagación hacia atrás se realiza nuevamente en cada iteración/lote. El algoritmo se denomina estocástico porque el lote varía en cada actualización de pesos, a diferencia del caso anterior en el cual se realiza una actualización por cada época.

La elección del tamaño de lote permite encontrar un balance entre significancia estadística y eficiencia computacional. Por un lado, un lote podría tener un solo ejemplo. Entrenar la red con este lote sería un ejemplo de entrenamiento *online*, ya que en cada época actualizaríamos los pesos una vez por cada ejemplo. En este caso el valor de las derivadas sería casi aleatorio y muy probablemente inútil, ya que optimizar la red para un ejemplo a la vez dificulta la generalización, dado que los pesos de un ejemplo particular puede que no sean de utilidad para otro ejemplo. Por otro lado, el lote podría consistir en todos los ejemplos del conjunto entrenamiento, con lo cual volvemos al esquema anterior de entrenamiento *batch*.

En general, los tamaños de lotes usuales para el descenso de gradiente estocástico varían entre 8 y 512, dependiendo de la aplicación y la cantidad de memoria disponible.

Regularización

Hay dos esquemas básicos de regularización que se suelen utilizar en las redes neuronales: regularización L2, y decadencia de pesos.

La regularización L2 penaliza a la redes con pesos cuyos valores son muy altos en magnitud. De esta forma, se logra que la red no sobre-ajuste al conjunto de entrenamiento, ya que la penalización impone una restricción de magnitud sobre la salida de las capas lineales. Para ello, en una red con pesos p_i con $i = 1, \dots, L$, se establece la función de error regularizado E_r según la Ecuación [2.13], donde $|p_i|^2$ denota la norma euclidiana al cuadrado del parámetro p_i.

$$E_r(X, Y, e, p_1, \dots, p_l) = E(X, Y, e) + \sum_{i=1} L|p_i|^2 \qquad [2.13]$$

El segundo término de E_r es la penalización. Con esta función de error promedio, el entrenamiento de la red a través de las derivadas naturalmente llevará a la misma a buscar una solución con parámetros de baja magnitud.

Por otro lado, la decadencia de pesos es un algoritmo simple cuyo funcionamiento básico es multiplicar a los pesos por un valor γ entre 0 y 1 en cada iteración. Los valores de γ generalmente está muy cercanos a uno, como 0.99 o 0.999 [9]. Es decir, los pesos se modifican en cada iteración de acuerdo a la Ecuación [2.14]:

$$p_i \leftarrow p_i * \gamma \qquad [2.14]$$

[9] En ocasiones, en lugar de especificar el valor γ, se especifica el complementario $1 - \gamma$, de modo que los valores de decadencia suelen ser cercanos a 0.

De esta forma, obtenemos un resultado similar al de la regularización L2, pero con un proceso de modificación de pesos paralelo al del entrenamiento. No obstante, en varios casos ambos esquemas de regularización son equivalentes [GBC16].

2.2.12. Librerías de Redes Neuronales

Existen diversas librerías que facilitan la implementación de redes neuronales. Las más conocidas y utilizadas actualmente son Tensorflow [Aba+16] y Pytorch [Pas+19]. Estas librerías permiten especificar un grafo de computación mediante código del lenguaje Python, a distintos niveles de abstracción, ya sea con operaciones de bajo nivel entre tensores, o definiendo directamente capas lineales y funciones de activación mediante clases específicas.

La ventaja de utilizar librerías como estas radica en que establecen un lenguaje común para toda la comunidad que utiliza modelos de redes neuronales. Además, las operaciones básicas se encuentran optimizadas, y permiten ejecutar el mismo modelo en distintos dispositivos, como CPUs o GPUs. Por último, para la mayoría de operaciones las librerías permiten sólo especificar el cálculo de la etapa de preparación hacia delante, es decir, diseñar la salida de la red, y proveen una implementación automatizada de la etapa de propagación hacia atrás para calcular las derivadas del error respecto de los parámetros. Por eso mismo también permiten utilizar varios algoritmos de optimización de forma intercambiable.

2.2.13. Resumen

En esta sección detallamos los aspectos básicos de las redes neuronales, en términos de su diseño, considerando su topología y los tipos de capas más usuales: lineales y funciones de activación.

Además, desarrollamos el algoritmo de entrenamiento más utilizado por las redes, compuesto por la propagación hacia atrás y el descenso de gradiente. Por último, comentamos sobre la evaluación por lotes, esquemas de regularización y librerías para implementar las redes.

A continuación, describimos capas específicas para trabajar con imágenes, que permiten construir Redes Neuronales Convolucionales.

2.3. Redes Convolucionales

Las redes neuronales presentadas en la sección sección 2.2 que utilizan capas lineales y funciones de activación permiten crear modelos para todo tipo de datos que puede codificarse como un vector de entrada.

No obstante, los datos de tipo imagen se codifican como un tensor t de tres dimensiones H, W y F ($t \in R^{H \times W \times F}$), donde H y W representan el alto y el ancho, y F la cantidad de canales. Para imágenes en escala de grises, la dimensión F suele tener tamaño uno; para imágenes a color en RGB, F suele tener tamaño tres. Entonces, una imagen a color de 4 pixeles de alto y 8 de ancho se codificará como un tensor con dimensiones $(4, 8, 3)$. En general, una imagen puede tener una cantidad de canales F arbitrarios. Cada una de las F matrices de tamaño (H, W) también se conocen como mapas de características o *feature maps* (FMs) (Figura 2.38).

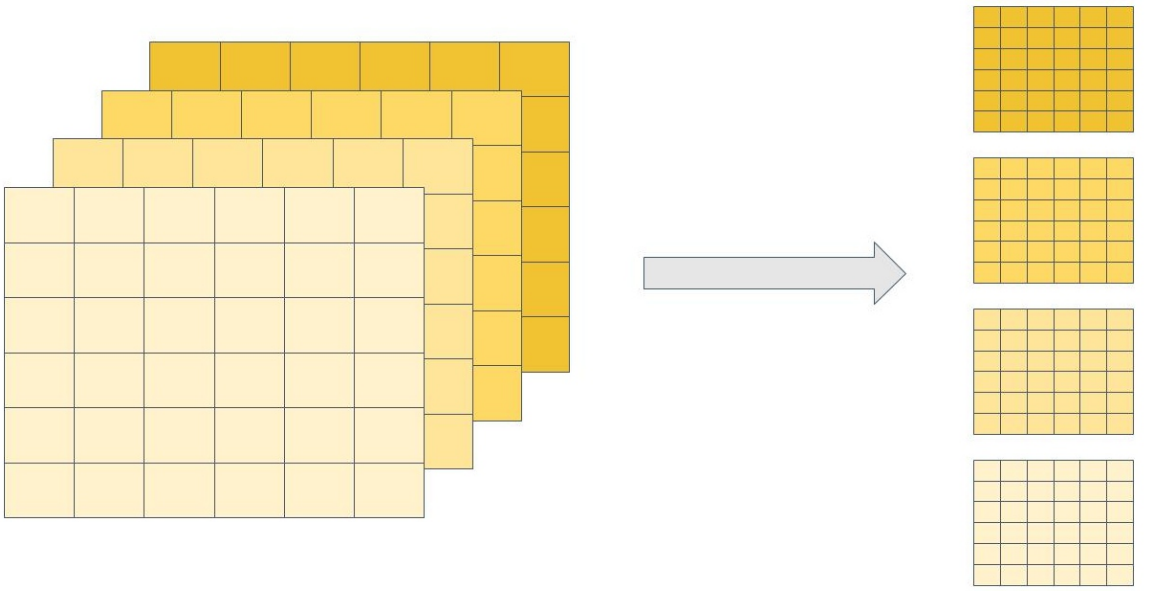
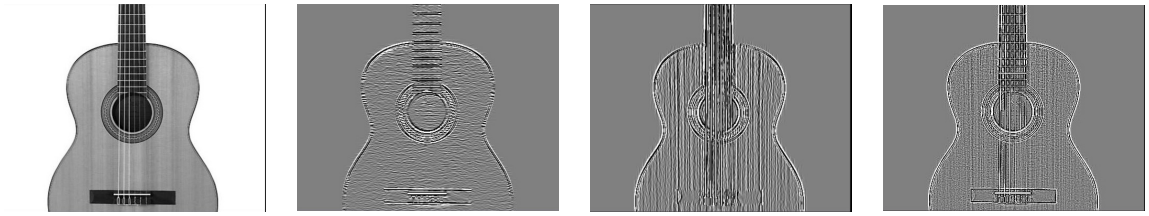


Figura 2.38: Una imagen formada por varios mapas de características. Cada mapa de características tiene tamaño (H, W); en este caso en particular, la imagen tiene 4 mapas de características (FMs) cada uno de tamaño $(6, 6)$

La estructura espacial y canales de imagen impide su fácil procesamiento utilizando las capas lineales que vimos anteriormente. Para procesar una imagen con estas capas debemos convertirla previamente a un vector, es decir un tensor con una sola dimensión. Para ello, existe la operación o capa de aplanamiento, que describimos a continuación.



(a) (b) (c) (d)

Figura 2.39: Imagen original (a) procesada con un filtro de paso alto horizontal (b), vertical (c) y de ambos lados (d).

2.3.1. Capa de aplanamiento (*Flatten*)

La capa de vectorización o aplanamiento (*Flatten*), convierte un tensor de dimensiones arbitrarias $(D_1, D_2, \dots, D_k)$ en un tensor con una sola dimensión, es decir, un vector. Para ello, simplemente se calcula la cantidad de valores individuales del tensor $D = D_1 \times D_2 \times \cdots \times D_k$, y se modifica el tipo del tensor a un vector de tamaño D. En la mayoría de los casos, no es necesario ningún tipo de copia de memoria, ya que es una cuestión de considerar a los mismos elementos del tensor original con otra organización. El orden de los elementos en el nuevo vector generalmente depende de la implementación, de acuerdo a como se almacenaba el tensor original.

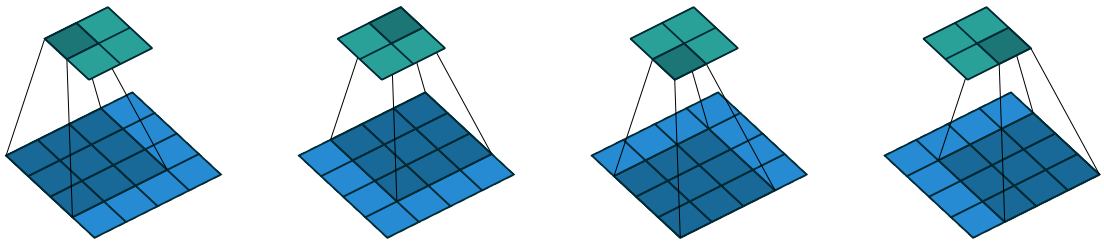
Por ejemplo, en el caso de la imagen de dimensiones $(4, 8, 3)$, el aplanamiento da como resultado un vector de dimensiones $(96) = (4 \times 8 \times 3)$. De esta forma, se puede utilizar una capa lineal para procesar una imagen.

2.3.2. Capas Convolucionales

Si bien la combinación de capas de aplanamiento y lineales permite procesar imágenes, dicho procesamiento será poco eficiente y eficaz, debido a que ignora la estructura espacial de la imagen. Dicha estructura permite utilizar operaciones especiales para imágenes que aprovechan la localidad espacial y la estructura en canales.

En este caso, la operación que utilizaremos es la convolución 2D, que permite aplicar un mismo filtro en distintas ubicaciones espaciales de la imagen. Las salidas de la aplicación de filtro se organizan espacialmente a partir de la imagen de entrada para generar una imagen resultante. La convolución permite detectar características de las imágenes de entrada y resaltarlas en la imagen resultante (Figura 2.39). Los filtros también se conocen con el nombre de *núcleos* o *kernels*.

La convolución 2D está definida en general para dos señales x y f como:



(a) (b) (c) (d)

Figura 2.40: Esquema del cálculo de la operación de convolución 2D con un filtro de 3×3 y una imagen de 4×4 [DV16].

$$y[i,j] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[m,n] \cdot x[i+n, j+m] \qquad [2.15]$$

En la Ecuación [2.15], la señal y es la imagen de salida, la señal x es la imagen de entrada y la señal f es el filtro.

Para cada posición de salida $[i,j]$, la convolución combina los valores de la entrada y el filtro, para producir una salida transformada.

Esta definición asume que las señales son infinitas, pero en la práctica lidiamos con imágenes y filtros digitales con soporte finito, es decir, con matrices de tamaño (H,W). Como mencionamos antes, las imágenes tienen varios canales, por lo cual el tensor que representa una imagen tiene dimensiones (H,W,C); no obstante, dado que sólo dos dimensiones son espaciales, las seguimos llamando convoluciones $2D$. Por ende, a continuación definiremos las convoluciones en términos de estos tensores.

Una convolución recibe c_x mapas de características, es decir, una imagen de dimensiones (h_x, w_x, c_x). Además, recibe un filtro de tamaño (h,w,c_x) y genera como resultado un solo FM y de tamaño (h_y, w_y), donde $h_y = h_x - h + 1$ y $w_y = w_x - w + 1$). La Figura 2.40 muestra un esquema de esta operación, y el Listado 2.1 muestra un ejemplo de su implementación en Python.

```
import numpy as np
def conv2d(f, x):
    # 'f': el vector de pesos o coeficientes del filtro
    # 'x': imagen de entrada

    h_x,w_x,c_x = x.shape #alto, ancho y canales
    h,w,c = w.shape #alto, ancho y canales
    assert c == c_x
    assert impar(h) and impar(w)

```

```
    # Calcular dimensiones de salida
    h_y = h_x - h + 1 #Alto de la matriz de salida (imagen)
    w_y = w_x - w + 1 #Ancho de la matriz de salida (imagen)
    #Vector de salida (un sólo feature map)
    y = np.zeros( (h_y,w_y) )

    for i in range(w_y):
        for j in range(w_y):
            x_window=x[i:i+h,j:j+w,:]
            y[i,j] = (w*x_window).sum()
    return y
```

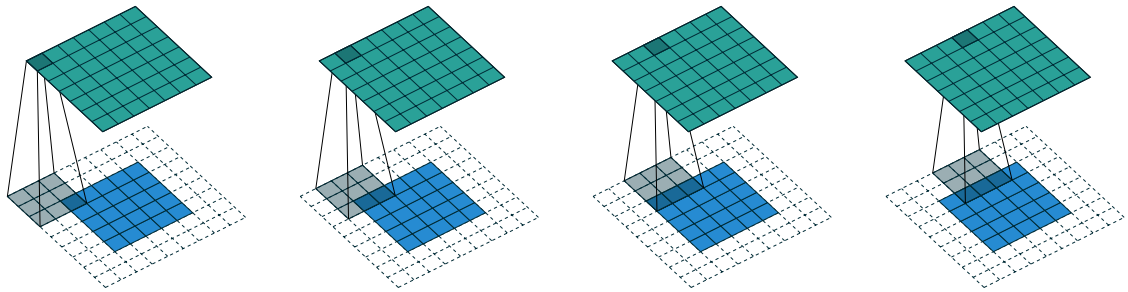
Algoritmo 2.1: Operación de convolución

Las capas convolucionales tienen f_y filtros. Por cada filtro se realiza una operación de convolución distinta. La capa convolucional recibe la imagen de tamaño (h_x, w_x, c_x), y aplica cada uno de los f_y filtros por separado a dicha entrada. Así se obtienen f_y imágenes de tamaño (h_y, w_y), que se juntan para formar la imagen de salida con tamaño (h_y, w_y, f_y) (Listado 2.2).

```
import numpy as np
def conv2d_capa(fs, x):
    # 'ws': Tensor de pesos o coeficientes del filtro
    # 'x': imagen de entrada

    h_x,w_x,c_x = x.shape # alto, ancho y canales
    f_y,h,w,c = f.shape #cantidad de filtros, y su alto, ancho y canales

    # Calcular dimensiones de salida
    h_y = h_x - h + 1 #Alto de salida
    w_y = w_x - w + 1 #Ancho de la salida
    #Salida (f_y feature maps)
    y = np.zeros( (h_y,w_y,f_y) )

    # Calcular convoluciones de cada filtro
    for i in range(F):
        y[:,:,i]= conv2d(ws[i],x)
    return y
```

Algoritmo 2.2: Operación de la capa de convolución

Las capas convolucionales tienen una gran ventaja frente a los filtros diseñados por un experto. Los coeficientes o pesos de sus filtros son parámetros de la red, y se aprenden también mediante propagación hacia atrás y descenso del gradiente.



(a) (b) (c) (d)

Figura 2.41: Esquema del cálculo de la operación de convolución 2D con un filtro de 3×3 y una imagen de 5×5 con relleno de 2 pixeles de cada lado. [DV16]

Convoluciones con relleno (*Padding*)

El tamaño espacial de la salida de las convoluciones generalmente es menor al de la entrada. Como las imágenes son de tamaño finito, los filtros convolucionales no pueden ser aplicados en los bordes de las mismas, y la salida pierde píxeles en los bordes. Dado que en general preferimos que los tamaños de las imágenes sean potencias de 2, por temas de eficiencia y simplicidad, este fenómeno genera inconveniencias.

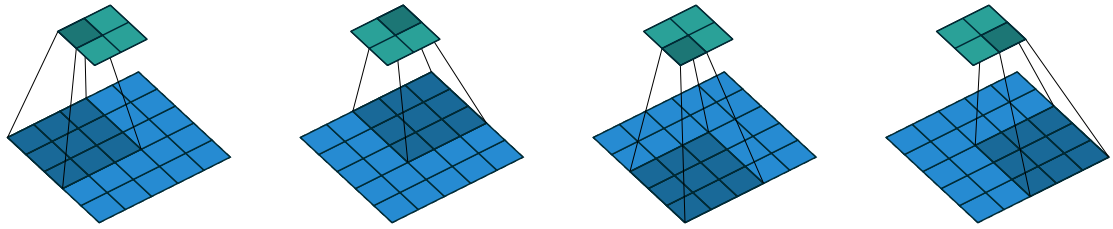
Las convoluciones con relleno intentan paliar esta inconveniencia agrandando la imagen previo a la convolución. Para ello se *rellenan* los bordes, agregando píxeles arriba, abajo, a la izquierda y a la derecha de la imagen (Figura 2.41). La cantidad que se agrega se expresa como la cantidad de filas o columnas que se agrega en un lado en particular, sino en términos de píxeles o de filas o columnas totales. Los píxeles agregados generalmente tienen un valor fijo que suele ser cero, pero también pueden utilizarse los píxeles del borde de la imagen original duplicados en forma de espejo.

En cualquier caso, dada una imagen de tamaño (H, W), si se rellena con r_h y y_w píxeles, se obtiene una imagen de tamaño $(H + 2r_h, W + 2r_w)$.

Generalmente, el tamaño de relleno R se elige en conjunto con el tamaño de los filtros convolucionales. Si es filtro tiene tamaño (h, w), entonces quitará a la imagen $(h-1)/2$ pixeles arriba y abajo, y $(w-1)/2$ a la izquierda y a la derecha. Por ende, si el relleno se elige como $r_h = (h-1)/2$ y $r_w = (w-1)/2$, el tamaño de la imagen de salida sea igual que el de la imagen de entrada.

Convoluciones espaciadas (*Strided Convolutions*)

Las convoluciones anteriores se realizan de forma densa, es decir, excepto por los bordes, por cada dimensión espacial de la imagen de entrada se genera una dimen-



(a) (b) (c) (d)

Figura 2.42: Esquema del cálculo de la operación de convolución 2D con un filtro de 3×3 y una imagen de 5×5 con *paso* $= 2$ [DV16].

sión espacial correspondiente en la imagen de salida.

Las convoluciones espaciadas o con *paso* $\neq 1$ permiten que los filtros no se apliquen con cada par de índices espaciales de la entrada, sino que se salteen algunos. En general, se suele utilizar *paso* $= 2$, de forma que los FMs se escalan a la mitad del tamaño (Figura 2.42), pero también es posible utilizar valores de *paso* mayores a 2. El Listado 2.3 muestra una implementación posible de esta operación sobre un solo FM.

```
import numpy as np
def conv2d_paso(f, x, h_paso,w_paso):
    # 'f': el vector de pesos o coeficientes del filtro
    # 'x': imagen de entrada
    # 'h_paso': el tamaño de paso vertical para las convoluciones espaciadas
    # 'w_paso': el tamaño de paso horizontal para las convoluciones espaciadas

    h_x,w_x,c_x = x.shape #alto, ancho y canales
    h,w,c = w.shape #alto, ancho y canales
    assert c == c_x
    assert impar(h) and impar(w)

    # Calcular dimensiones de salida
    h_y = ((h_x - h)//h_paso) + 1 #Alto de la matriz de salida (imagen)
    w_y = ((w_x - w)//w_paso) + 1 #Ancho de la matriz de salida (imagen)
    #Salida (un sólo feature map)
    y = np.zeros( (h_y,w_y) )

    for i in range(h_y):
        i_paso = i*h_paso
        for j in range(w_y):
            j_paso = j*w_paso
            x_window=x[i_paso:i_paso+h,j_paso:j_paso+w,:]
            y[i,j] = (w*x_window).sum()
```

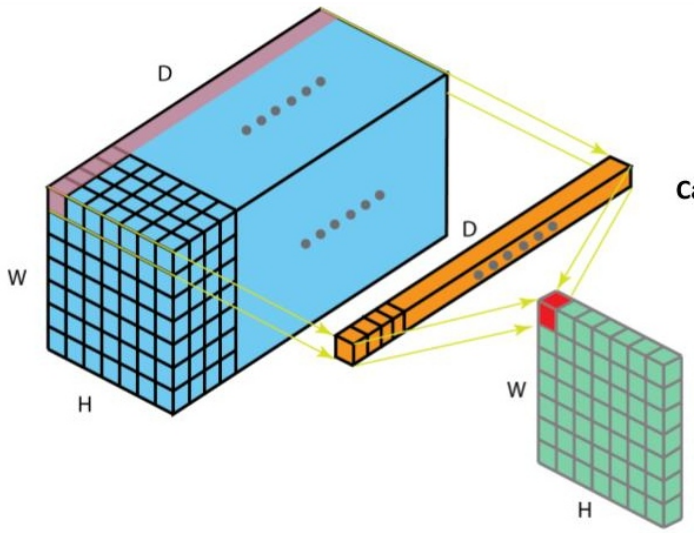


Figura 2.43: Cálculo de la operación de convolución 2D con filtros de tamaño 1×1.

```
25      return y
```

Algoritmo 2.3: Operación de convolución

Convoluciones 1×1

Las convoluciones 1×1 son un caso especial de las convoluciones, que utilizan filtros cuyo tamaño espacial no les permite filtrar un pixel junto con sus vecinos. No obstante, estas convoluciones si actúan en la dimensión de canales, ya que el filtro tiene tamaño $1 \times 1 \times c_x$ si hay c_x canales en el FM de entrada. Como las capas convolucionales utilizan c_y filtros para generar c_y FMs de salida, entonces dichas capas con convoluciones 1×1 permiten generar combinaciones lineales de los filtros existentes para generar nuevos. Esta combinación lineal es aprendida, ya que los coeficientes del filtro son parámetros de la red.

La Figura 2.43 ejemplifica el funcionamiento de estas convoluciones. En ocasiones, este tipo de convoluciones se utiliza para obtener más FMs de salida que los de entrada, o viceversa según el problema.

2.3.3. Capas de Agrupamiento (*Pooling*)

Las capas de agrupamiento permiten reducir la dimensionalidad de las representaciones internas de la red, y generalmente se utilizan sobre los FMs.

La capa de agrupamiento más conocida es la de Agrupamiento Espacial, que permite reducir el tamaño espacial de los FMs. Por otro lado, las capas de Agrupamiento Global permiten eliminar completamente las dimensiones espaciales de los

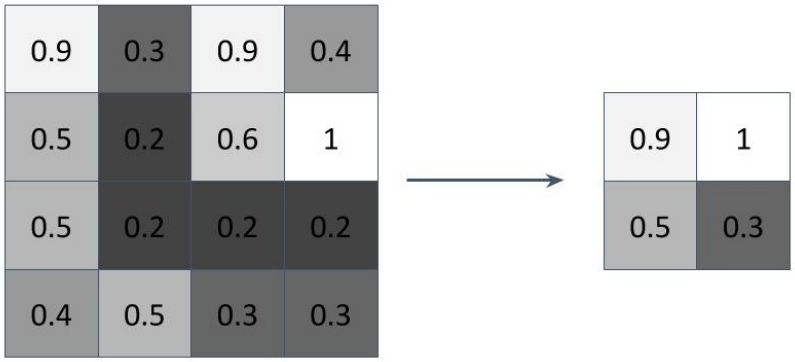


Figura 2.44: Cálculo de la operación de MaxPooling con ventanas de tamaño 2×2.

FMs, y obtener un vector resultante de manera similar a la operación de las capas de Aplanado. A continuación, describimos en más detalle ambas operaciones.

Agrupamiento espacial (*Spatial Pooling*)

Las capas de Agrupamiento Espacial permiten reducir el tamaño espacial de los feature maps, de modo de reducir la dimensionalidad. Esta operación suele ser necesaria por dos motivos. Por un lado, la aplicación de varios filtros convolucionales sobre los FMs de entrada suele tener un efecto similar al de la aplicación de varios filtros gaussianos; la resolución efectiva de los objetos decrece, mientras que la resolución de la imagen se mantiene constante. Al mismo tiempo, para mantener constante el coste computacional y de memoria pero incrementar la eficacia del modelo, es preferible achicar el tamaño de cada FM pero incrementar su cantidad.

De este modo, las capas de Agrupamiento Espacial simplemente son una versión derivable de operaciones de escalado de imágenes. En particular, una variante popular es el Agrupamiento por Máximo o *MaxPooling*. Esta operación trabaja de forma separada con cada FM. MaxPooling achica el tamaño espacial del FM de entrada dividiéndolo en ventanas, que son reemplazadas por el máximo de esa ventana el FM de salida (Figura 2.44). El Listado 2.4 muestra una posible implementación de esta operación. También, existen otras variantes que utilizan otras funciones de agregación en lugar del máximo, como la suma, producto, mínimo, media, etc. Su implementación es similar, excepto por la línea 23 del Listado 2.4

```
1 import numpy as np
2 def max_pooling(x, h_paso,w_paso,h,w):
3     # 'h_paso': tamaño de paso vertical
4     # 'w_paso': tamaño de paso horizontal
5     # 'h': tamaño de la ventana vertical
6     # 'w': tamaño de la ventana horizontal
7     # 'x': imagen de entrada (c feature maps)
```

```
8
9     h_x,w_x,c = x.shape #alto, ancho y canales
10
11    # Calcular dimensiones de salida
12    h_y = ((h_x - h)//h_paso) + 1 #Alto de la matriz de salida (imagen)
13    w_y = ((w_x - w)//w_paso) + 1 #Ancho de la matriz de salida (imagen)
14    #Salida (c feature map)
15    y = np.zeros( (h_y,w_y,c) )
16
17    for k in range(c):
18      for i in range(h_y):
19          i_paso = i*h_paso
20          for j in range(w_y):
21              j_paso = j*w_paso
22              x_window=x[i_paso:i_paso+h,j_paso:j_paso+w,:]
23              y[i,j,k] = x_window.max()
24    return y
```

Algoritmo 2.4: Operación de la capa de Agrupamiento por Máximo (MaxPooling)

Agrupamiento global (*Global Pooling*)

El Agrupamiento Global es un caso extremo del espacial en el cual todas las dimensiones espaciales desaparecen. Su objetivo es similar al de las capas de Flatten, pero en lugar utilizar todos los elementos originales de los FMs de entrada, para cada FM calcula una función de agregación de los valores en sus dimensiones espaciales (alto y ancho).

De esta forma, si la entrada es un conjunto de c FMs, la salida es un vector de c elementos, donde cada elemento tiene, por ejemplo, el promedio de los valores de cada FM (Figura 2.45). El Listado 2.5 muestra una posible implementación de este caso, que se conoce como Agrupamiento Global con Promedio o *Global Average Pooling* (GAP).

```
1 import numpy as np
2 def global_average_pooling(x):
3     # 'x': imagen de entrada
4     h,w,c = x.shape #alto, ancho y canales
5     y = np.zeros( c ) # Vector de salida
6
7     for i in range(c):
8         # promedio de las dimensiones espaciales
9         y[i]= x[:,:,c].mean()
```

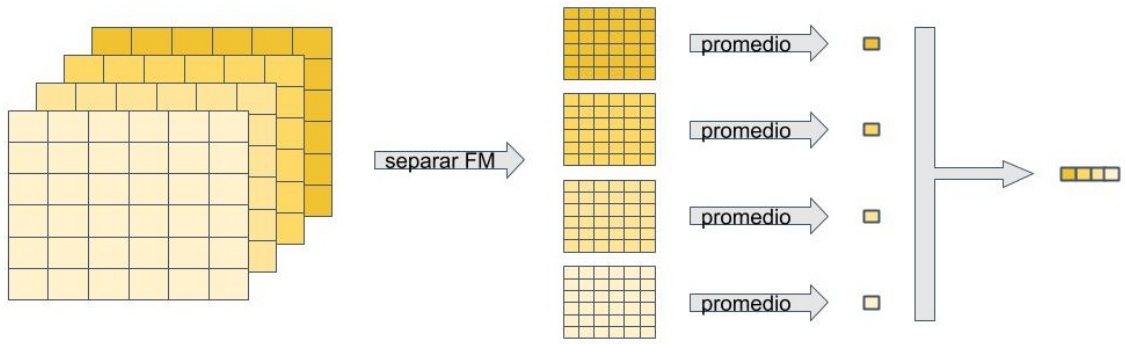


Figura 2.45: Cálculo de la operación de GlobalAveragePooling.

```
    return y
```

Algoritmo 2.5: Operación de la capa de Agrupamiento Promedio Global (GlobalAveragePooling)

El GAP se suele utilizar en combinación con una convolución 1×1 para generar puntajes por clases. Para un problema con C clases, la convolucion 1×1 primero cambia la cantidad de FMs a C, obteniendo un conjunto de featuremaps con dimensiones (H, W, C). Luego, el GAP elimina las dimensiónes (H, W), obteniendo un vector con C elementos de puntajes por clase.

2.3.4. Resumen

En esta sección detallamos los aspectos básicos de las redes neuronales convolucionales. Introducimos las capas convolucionales, que aplican un filtro aprendido a su entrada para determinar automáticamente sus características. Consideramos algunas variaciones de dicha operación, como la convolución con relleno, espaciada o 1×1.

También comentamos el funcionamiento de las capas de Agrupamiento o Pooling, tanto las espaciales, de las cuales la de MaxPooling es la más conocida, y las globales, ejemplificadas por Global Average Pooling.

A continuación, describimos algunos modelos de redes convolucionales conocidos que serán utilizados luego en las distintas contribuciones de esta tesis.

2.4. Modelos de Redes Convolucionales

En esta sección realizamos un resumen de las arquitecturas de redes convolucionales utilizadas en los análisis y experimentos de esta tesis.

Describiremos un modelo convolucional simple llamado *SimpleConv*, que es una variación de *LeNet*. Por otro lado, también resumimos las características principales de otros modelos más complejos que utilizamos en los experimentos: AllConvolutional, VGG, Inception-v3 y ResNet. Estos modelos han demostrado su desempeño en problemas de procesamiento de imágenes y visión por computadora en conjuntos de datos conocidos, como MNIST, CIFAR10 e ImageNet. Además, cada uno ha incorporado alguna propiedad particular que lo ha distinguido de modelos anteriores.

2.4.1. LeNet y SimpleConv

El modelo LeNet [LeC+98] fue propuesto en 1998, para resolver problemas de reconocimiento óptico de caracteres, así como letras escritas a mano en documentos. Es uno de los primeros modelos convolucionales y una línea de base común para el campo de las CNNs.

Su arquitectura consiste simplemente en capas convolucionales 2D estándar, con max-pooling para reducir la dimensión espacial y una cabeza de clasificación con dos capas lineales (Figura 2.46).

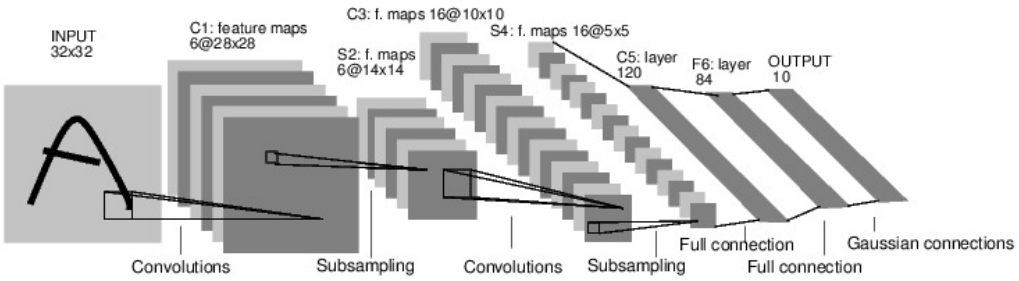


Figura 2.46: Diagrama de la arquitectura LeNet.

SimpleConv es una variante de LeNet que tiene un buen desempeño en MNIST y CIFAR10 (sección 2.5), pero al mismo tiempo es una red simple y eficiente, ya que como LeNet sólo está compuesta de capas convolucionales y max-pooling (Figura 2.47). Todos los filtros convolucionales son de tamaño 3×3. F representa la cantidad de filtros o feature maps de las capas convolucionales. F es un parámetro que se ajusta para las distintas bases de datos, pero sus valores usuales son 32, 64 o 128. L es la cantidad de salidas de la capa lineal. C es la cantidad de clases de la salida. La función de activación por defecto es ELU.

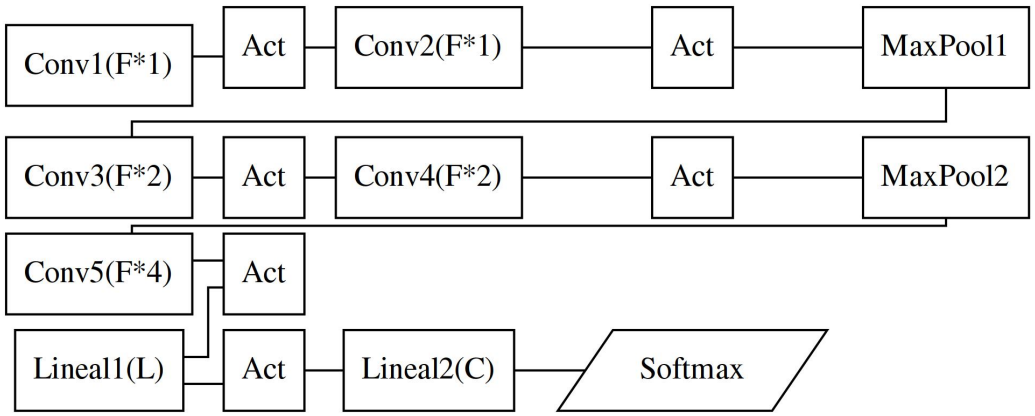


Figura 2.47: Diagrama de la arquitectura SimpleConv.

2.4.2. AllConvolutional

La arquitectura AllConvolutional [Spr+15] está basada en utilizar solo capas convolucionales. Por ende, se remueven las capas de max-pooling, y se reemplazan con capas convolucionales con *paso* $= 2$ para achicar el tamaño espacial de los feature maps. Además, las capas lineales se reemplazan por una operación llamada Global Average Pooling (sección 2.3) que reemplaza las dimensiones espaciales de cada feature map por su promedio, y de esa forma puede convertir feature maps en puntuaciones de clase.

La Figura 2.48 muestra un diagrama de la arquitectura. Como en el caso de SimpleConv, F es la cantidad de filtros convolucionales, L la dimensión de la capa lineal, y C la cantidad de clases. El hiperparámetro k de las capas convolucionales indica el tamaño del kernel.

2.4.3. VGG

La red VGGfue desarrollada en 2013 para la competencia LSVRC [SZ14]. Su principal contribución fue proveer evidencia a favor de apilar varias capas convolucionales con pequeños filtros de 3×3 en lugar de los diseños con filtros de tamaños mayores (5×5 o 7×7). Al apilar varios filtros pequeños se puede aproximar filtros grandes de forma más eficiente; este razonamiento es similar a la aplicación sucesiva de filtros gaussianos.

Por ende, la red consiste en una gran cantidad de filtros convolucionales de 3×3, apilados en grupos de 2 o 3 capas, con capas max-pooling entre ellos. La cabeza de clasificación consiste en dos capas lineales de 4096 salidas, y una capa lineal final de clasificación.

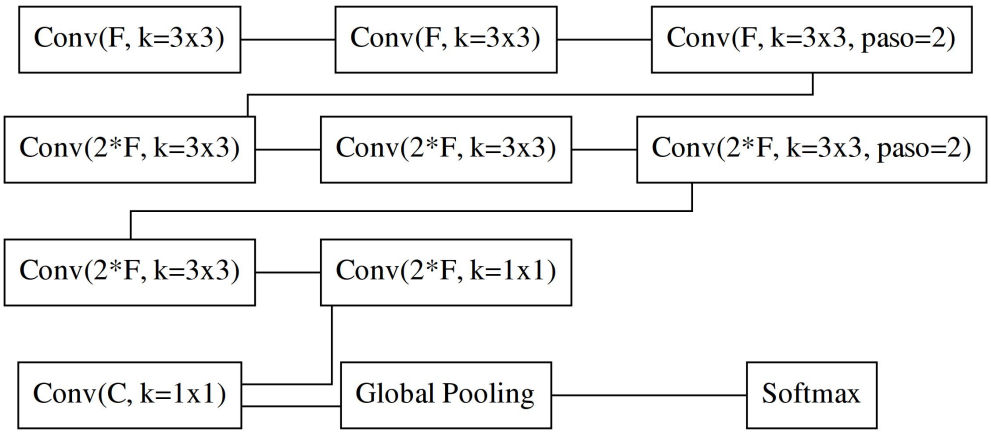


Figura 2.48: Diagrama de la arquitectura AllConvolutional [Spr+15]. En lugar de las capas max-pooling, se encuentran convoluciones espaciadas con *paso* = 2. La cabeza de clasificación consiste en una capa de Global Average Pooling seguida por la tradicional Softmax.

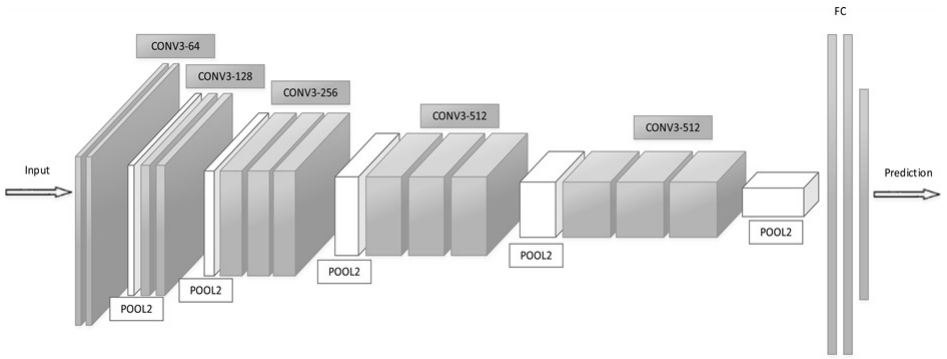


Figura 2.49: Diagrama de la arquitectura VGG, con los filtros apilados y capas max-pooling entre ellas. [SZ14]

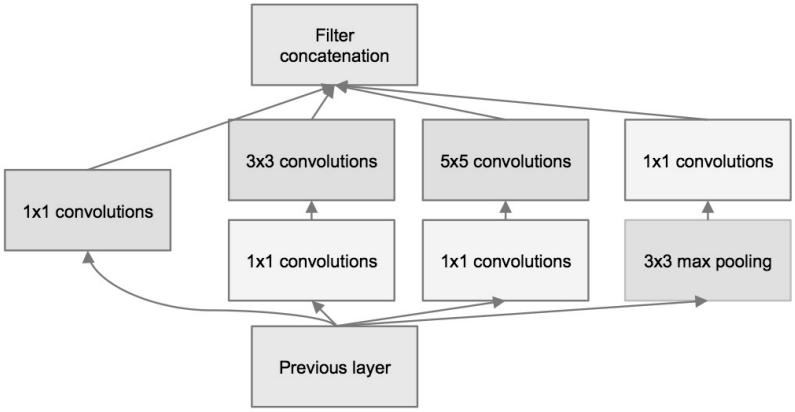


Figura 2.50: Diagrama de un bloque Inception. Los 4 caminos distintos promueven la diversidad de representaciones y filtros. Las convoluciones 1×1 antes de las convoluciones 3×3 y 5×5 son el cuello de botella [Sze+15a].

La gran cantidad de parámetros le otorga al modelo mucha potencialidad. No obstante, tanto su entrenamiento como su evaluación son muy costosos computacionalmente, además de utilizar una gran cantidad de memoria.

2.4.4. Inception

La arquitectura Inception [Sze+15a; Sze+15b] fue introducida en 2014. Originalmente se llamó GooLeNet pero las nuevas versiones se denominaron Inception-vN, donde N es la versión específica.

La contribución principal del modelo es el diseño de bloques Inception. Dichos bloques incluyen una capa que funciona como cuello de botella. El cuello de botella se aplica mediante convoluciones de tamaño de kernel 1×1, que reducen la cantidad de canales antes de aplicar kernels de tamaños espaciales mayores. Además, se utilizan distintos *caminos* en el bloque, de modo de promover la diversidad de representaciones. Finalmente, los resultados de cada camino se concatenan en la dimensión de canales para formar la salida del bloque.

De esta forma, se logran extraer características a varios niveles de escala en el mismo bloque del modelo. Los múltiples caminos, a través de cada bloque, dan posibilidad a la red de lograr representaciones muy diversas a través del aprendizaje, ya que los filtros 1×1 efectivamente funcionan como selectores o combinadores de canales. La diversidad además ayuda a reducir el coste computacional de la red, ya que se pueden codificar funciones más complejas usando la misma cantidad de parámetros.

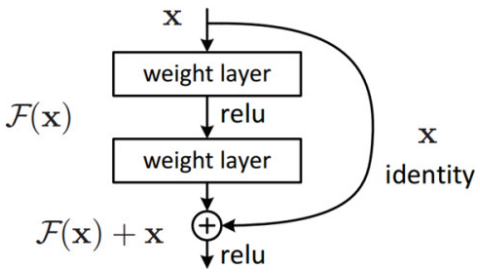


Figura 2.51: Diagrama de un bloque residual [He+16]. La entrada se combina con la salida de la transformación mediante la función suma.

2.4.5. ResNet

Los modelos de redes con conexiones residuales (ResNet) [He+16] surgieron en el año 2016. Las técnicas de bloques residuales que introdujeron permitieron el entrenamiento y uso de redes mucho más profundas que lo que era posible en ese entonces. Los bloques residuales transforman su entrada, pero tienen como salida final la suma de la entrada y la transformación de la misma (Figura 2.51). Los bloques se inicializan de forma tal que al comienzo del entrenamiento calculen la función identidad. Luego, el entrenamiento entonces aprende a generar transformaciones a partir de la entrada. De esta forma, si una capa no tuviese información para agregar, puede simplemente retener los pesos correspondientes a la función identidad. Además, las conexiones residuales permiten el flujo de gradientes hacia la capa anterior directamente. De esta forma, se logró entrenar modelos de redes neuronales con una mayor cantidad de capas que las reportadas hasta ese entonces en el estado del arte [He+16].

Por ende, la contribución más importante de esta arquitectura fue demostrar que pueden entrenarse redes con cientos de capas utilizando técnicas conocidas y estándar como el descenso de gradiente estocástico.

Esta arquitectura también utiliza filtros convolucionales de 1×1 en los módulos residuales para formar capas de cuello de botella como las vistas en la arquitectura Inception. Esto ayuda a reducir la dimensionalidad en el módulo residual y luego restaurarla. De este modo se puede igualar la cantidad de canales de la entrada con la de la salida, un requisito para poder aplicar la función suma entre las dos salidas. ResNet también utiliza Global Average Pooling, pero adiciona una capa lineal antes de la capa final de clasificación Softmax.

2.4.6. Resumen

El auge de los modelos de Redes Neuronales Profundas ha generado una revolución en el desempeño de los modelos de aprendizaje automático. En el caso de las CNNs, existe una gran variedad de modelos propuestos en los últimos años [Liu+17; Kha+19].

No obstante, se destacan algunos modelos emblemáticos como *LeNet*, AllConvolutional, Inception, VGG y ResNet. Cada uno ha contribuido algún tipo de mejora a los modelos de CNN, que han elevado significativamente el estado del arte anterior en cada ocasión[LeC+98; Spr+15; Sze+15a; SZ14; He+16; GBC16].

2.5. Conjuntos de datos para clasificación de imágenes

En esta sección, presentamos algunas de las BDs utilizadas en los trabajos experimentales de esta tesis. En particular, utilizamos las bases de datos MNIST y CIFAR10 para experimentos generales, y las BDs LSA16 y RWTH-PHOENIX-Weather específicas para la clasificación de formas de mano.

2.5.1. Conjuntos de datos conocidos para clasificación de imágenes

Las bases de datos MNIST y CIFAR10 son estándar en el área de clasificación de objetos en imágenes y se han utilizado en numerosos experimentos y comparaciones [GBC16]. Como estas BDs tienen baja resolución, permiten realizar experimentos con mayor rapidez. Al ser bien conocidas y contener ejemplos reales, los resultados que se obtengan sobre ellas son relevantes para la comunidad de aprendizaje automático.

MNIST

La base de datos MNIST [10] está formada por imágenes de dígitos manuscritos (Figura 2.52). Es un subconjunto de la base de datos Formularios y Caracteres Impresos Manualmente de NIST que publica el Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés) [LeC+98].

Contiene dos subconjuntos: uno de entrenamiento de 50 000 ejemplos y otro de prueba de 10 000 ejemplos. La BD contiene 10 clases, una por cada dígito. Es una BD balanceada, con 5000 ejemplos para cada clase para el entrenamiento, y 1000 para prueba.

Los dígitos tienen un tamaño normalizado y están centrados en una imagen de tamaño fijo (28×28). Las imágenes están codificadas en escala de grises. Los dígitos están segmentados, con un fondo mayormente negro y el trazo en grises y blancos.

La tarea de clasificación que propone es relativamente sencilla actualmente, ya que casi todos los métodos modernos obtienen una tasa de acierto del 99 % o superior en el conjunto de prueba [GBC16].

[10] Sitio oficial de la BD MNIST: `http://yann.lecun.com/exdb/mnist`

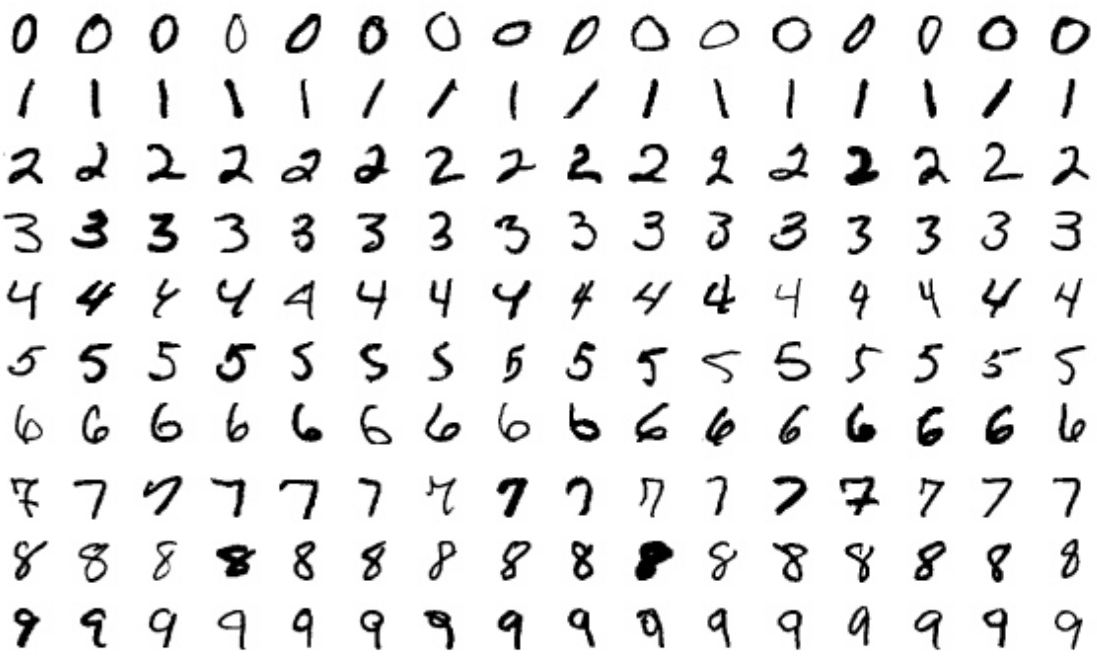


Figura 2.52: Ejemplos aleatorios de la base de datos MNIST.

Figura 2.53: Ejemplos aleatorios de la base de datos CIFAR10.

CIFAR10

La base de datos CIFAR-10 [11] consta de 60.000 fotos divididas en 10 clases (Figura 2.53). Fue creada por el *Canadian Institute For Advanced Research* (CIFAR). CIFAR-10 es un subconjunto etiquetado de la BD *80 million tiny images* [Kri+09]. El etiquetado fue realizado por estudiantes como un trabajo asalariado.

Las clases incluyen objetos comunes como aviones, automóviles, aves, gatos, etc. El conjunto de datos se divide de la misma forma que MNIST, con 50.000 imágenes en el conjunto de entrenamiento y los 10.000 restantes para el de prueba. Las imágenes son a color, codificadas en RGB y su tamaño es de 32 x 32 píxeles. Los objetos no están segmentados y tienen mucha diversidad, tanto en el fondo como en el objeto en sí.

[11] Sitio oficial de CIFAR10: https://www.cs.toronto.edu/~kriz/cifar.html

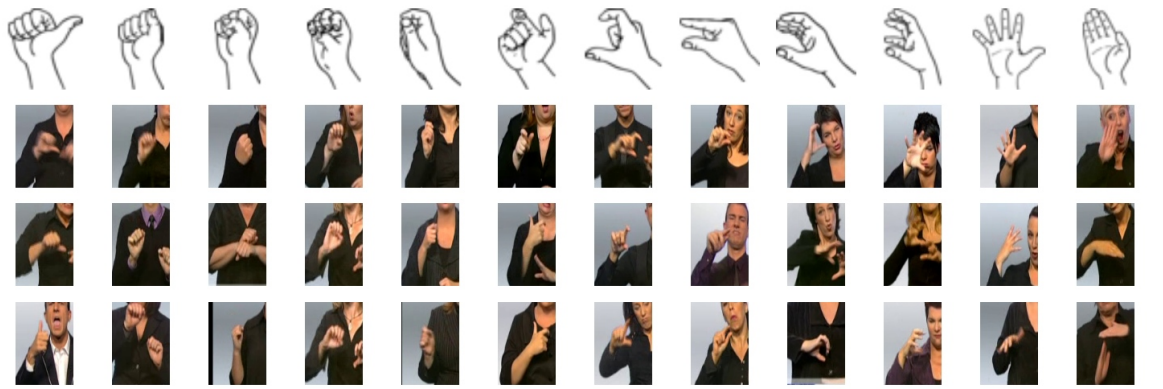


Figura 2.54: Imágenes del conjunto de datos RWTH-PHOENIX-Weather de formas de mano.

La tarea de clasificación que propone es de dificultad mediana actualmente, ya que la mayoría de los modelos modernos alcanzan una tasa de acierto de al menos 80 %. Los modelos propuestos en los últimos años incrementan esa tasa hasta llegar al 99 % [TL19].

2.5.2. Conjuntos de datos para la clasificación de formas de manos

Existen diversos conjuntos de datos para la clasificación de formas de mano. A continuación, describimos el conjunto RWTH-PHOENIX-Weather, generado a partir de recortes de pronósticos climáticos, y LSA16, un conjunto de imágenes de formas de mano utilizadas en la Lengua de Señas Argentina [Qui+16a].

RWTH-PHOENIX-Weather (RWTH)

El conjunto de datos de formas de mano RWTH-PHOENIX-Weather [12] de la universidad RWTH AACHEN contiene 3359 imágenes (Figura 2.54) de formas de mano de 45 clases distintas [KNB16]. Las imágenes fueron extraídas de otro conjunto de datos del mismo nombre pero con videos de señas dinámicas [For+12]. Este último conjunto fue presentado en 2012 y contiene recortes de programas de televisión en donde intérpretes de la lengua de seña traducen reportes del estado del clima.

En 2014, los autores incrementaron el tamaño del conjunto con nuevos videos, anotaciones de las posiciones de las manos y otros puntos de interés en las caras de los intérpretes. Adicionalmente se agregaron notas sobre las distintas posturas y formas de las manos que pueden ser utilizadas para tareas de clasificación de formas de mano.

[12] Sitio oficial de RWTH-PHOENIX-Weather: https://www-i6.informatik.rwth-aachen.de/~koller/1miohands-data/

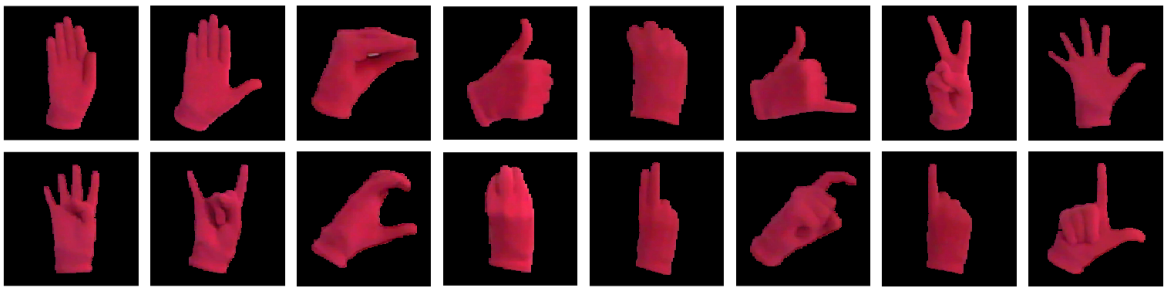


Figura 2.55: Ejemplos de cada clase de la base de datos LSA16.

Es importante destacar que este conjunto de datos tiene un grave desbalance de clases. Algunas de ellas tienen más de 200 ejemplos, mientras que otras menos que 5. Por ende, un preprocesamiento usual consiste en quitar las clases con 7 o menos ejemplos, de modo que las clases resultantes son 30, y la nueva cantidad de ejemplos se reduce muy ligeramente a unos 3289.

Base de datos de formas de mano de la Lengua de Señas Argentina (LSA16)

La base de datos de configuraciones de Lengua de Señas Argentina [13], creada con el propósito de producir un diccionario de LSA y entrenar un traductor automático de señas, contiene 800 imágenes en donde 10 sujetos realizaron 5 repeticiones de 16 tipos distintos de configuraciones de mano utilizadas en distintas señas de dicho lenguaje [Qui+16a]. Las configuraciones fueron elegidas dentro de las más utilizadas en el léxico, y se pueden observar en la Figura 2.55. Cada configuración fue realizada repetidamente en diferentes posiciones y diferentes rotaciones en el plano perpendicular a la cámara, para generar mayor diversidad y realismo en la base de datos.

Los sujetos vistieron ropa negra, sobre un fondo blanco con iluminación controlada, como se observa en la figura 2.56. Para simplificar el problema de segmentación de la mano dentro de una imagen, los sujetos utilizaron guantes de tela con colores fluorescentes en sus manos. Esto resuelve parcialmente pero de un modo muy eficaz el reconocimiento de la posición de la mano y carece de los problemas existentes en los modelos de piel. Por otro lado, propone un artefacto simple y económico al momento de realizar pruebas o confeccionar una aplicación real.

2.5.3. Resumen

Existen múltiples bases de datos para clasificación de imágenes [GBC16]. MNIST y CIFAR10 son BDs sumamente conocidas y utilizadas en miles de trabajos [Kha+19].

[13] Sitio oficial de LSA16: http://facundoq.github.io/unlp/lsa16/

Figura 2.56: Imágenes no segmentadas de la base de datos LSA16.

Ambas tienen una resolución de imagen pequeña, lo que permite realizar experimentos complejos. La simplicidad del problema controlado planteado por MNIST y el hecho de que CIFAR10 contenga imágenes naturales, así como el hecho de que ambas bases de datos tienen la misma distribución de ejemplos facilita la comparación de modelos y comprensión de resultados.

Por otro lado, LS16 y RWTH-PHOENIX-Weather son bases de datos de imágenes de formas de mano reconocidas en el área [Oli18; Mas+18; KNB16]. Nuevamente, se complementan debido a la naturaleza controlada de LSA16 y al realismo de RWTH-PHOENIX-Weather, lo cual permite realizar diversos análisis.

2.6. Invarianza y Equivarianza

Los modelos de aprendizaje automático y redes neuronales se utilizan en dominios muy diversos. En algunos de ellos, se conocen a priori ciertos requerimientos del modelo.

En ocasiones buscamos que un modelo se comporte de una manera determinada ante ciertos cambios de su entrada [DDFK16; Xie+17]. Por ejemplo, que ignore ciertas características de un objeto, o que cuantifique ciertos cambios del mismo. Como casos particulares, podemos mencionar el reconocimiento de texturas en imágenes, en el cual las transformaciones afines no afectan el tipo de textura. En la clasificación de formas de mano, todas las transformaciones afines de cuerpo rígido (sin deformar el objeto) tampoco deben afectar el reconocimiento. Por otro lado, dada una red que calcule la orientación de un objeto, deseamos que las transformaciones de orientación del objeto se reflejen de forma controlada en la salida de la red.

Estos requerimientos pueden formalizarse mediante los conceptos de invarianza, auto-equivarianza y equivarianza a las transformaciones. Las propiedades de invarianza o equivarianza nos permiten especificar como un modelo f reacciona ante transformaciones de la entrada x.

Coloquialmente, una red es invariante a un conjunto de transformaciones si su salida no cambia cuando una de esas transformaciones se aplica a su entrada. Por ejemplo, dada una red que clasifica imágenes, si el conjunto de transformaciones T son las rotaciones de 0°, 90°, 180° y 270°, la red es invariante a T si su clasificación (es decir, su salida) siempre es la misma aún al aplicar cualquiera de estas rotaciones a la imagen de entrada [DDFK16].

Formalmente, dada una transformación t, f es invariante a t si $f(x)$ es igual que $f(t(x))$ para todo x del dominio de f [DDFK16]. Esta propiedad debe cumplirse entonces para toda posible entrada de f (Ecuación [2.16]).

$$f(t(x)) = f(x) \qquad \forall x \in Dom(f) \tag{2.16}$$

Podemos generalizar esta noción a un conjunto $T = [\, t_1 \cdots t_m \,]$ de m transformaciones, de forma de que si f es invariante a T, entonces es invariante a toda transformación de T (Ecuación [2.17]).

$$f(t_1(x)) = f(t_2(x)) = \cdots = f(t_m(x)) \qquad \forall x \in Dom(f) \tag{2.17}$$

Asumiendo que la función identidad se encuentra dentro del conjunto T, lo cual

es muy usual, la Ecuación [2.18] provee una definición equivalente de invarianza que nos servirá mejor más adelante:

$$f(t(x)) = f(x) \qquad \forall t \in T, \forall x \in Dom(f) \tag{2.18}$$

Como se observa en la Figura 2.57 (a), una función invariante produce la misma salida para todas las transformaciones T de su entrada.

La *auto-equivarianza* es una propiedad complementaria a la invarianza. Una función es auto-equivariante si la *misma* transformación t puede ser aplicada a la entrada o la salida de la función con el mismo resultado [DDFK16], como se muestra en la Figura 2.57 (b).

$$f(t(x)) = t(f(x)) \qquad \forall t \in T, \forall x \in Dom(f) \tag{2.19}$$

La auto-equivarianza es una propiedad bastante rígida, ya que requiere la transformación t también esté definida para la salida de f, es decir, que $Im(f) \subset Dom(t)$.

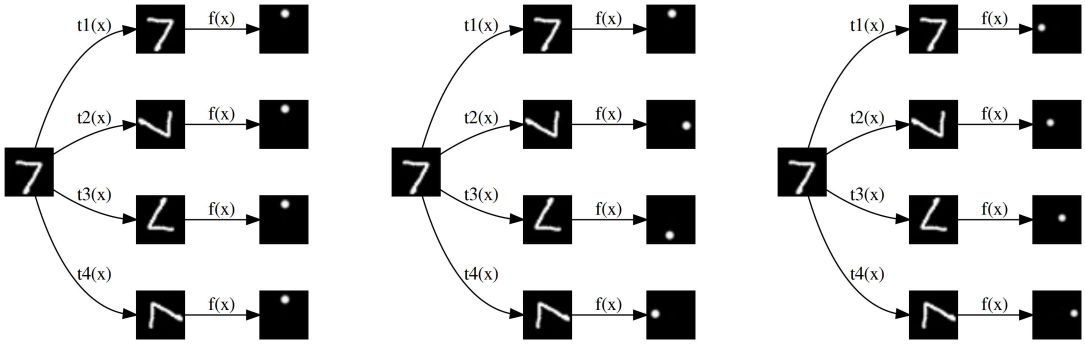
La generalización de ambas propiedades es la *equivarianza*. Una función es equivariante a t si su salida cambia de forma *predecible* cuando x es transformado por t [DDFK16]. Formalmente, es equivariante si existe una función t' que es la contrapartida de t, y que actua en la salida de la función f (Ecuación [2.20]). En esta definición, t' actúa sobre la salida $f(x)$, mientras que t actúa sobre la entrada x. Por ende, la función t' no necesita tener ningún tipo de similitud con t', como en el caso de la auto-equivarianza, ni está atada a ningúna restricción como en el caso de la invarianza.

$$f(t(x)) = t(f(x)) \qquad \forall t \in T, \quad \forall x \in Dom(f) \tag{2.20}$$

La Figura 2.57 (c) muestra como transformaciones de rotación t de la entrada pueden corresponder a transformaciones de traslación t' en la salida; en este caso, la función es equivariante.

Analizar si una función f es equivariante a una transformación t que opera sobre x requiere *determinar* la función t' correspondiente que actúa en las salidas [LV15]. Una condición suficiente para ello es que la función f sea invertible. Las CNNs son aproximadamente invertibles [Gil+17; LV15], por ende la existencia de t' es muy probable, al menos de forma aproximada. Por ende, analizar empíricamente la equivarianza de una CNN puede ser difícil [LV15]

Por otro lado, podemos ver que la invarianza es un caso particular de equivarian-



(a) Invarianza (b) Auto-equivarianza (c) Equivarianza

Figura 2.57: Ejemplos gráficos de invarianza, auto-equivarianza, y equivarianza de f al conjunto de transformaciones $T = [t_1, t_2, t_3, t_4]$ correspondientes a rotaciones de $[0°, 90°, 180°, 270°]$ respectivamente.

za en donde la función t' es simplemente la función identidad, es decir $t'(x) = x$. La auto-equivarianza también es un caso particular de la equivarianza, en donde la función que actúa sobre las entradas es la misma que la que actúa sobre las salidas, es decir, $t = t'$. Medir invarianza o auto-equivarianza no requiere estimar o determinar t', y por ende estas propiedades son más factibles de ser medidas [LV15; Goo+09].

2.6.1. Resumen

Las propiedades de invarianza, equivarianza y auto-equivarianza nos permiten especificar el comportamiento de un modelo en respuesta a transformaciones t de su entrada. La invarianza indica que un modelo no cambia ante la transformación de la entrada; la auto-equivarianza nos dice que la salida cambia de igual forma que la entrada. La equivarianza, un concepto que generaliza a los dos anteriores, requiere especificar una función t' que relaciona transformaciones t de la entrada con transformaciones t' de la salida. Por ende, la invarianza y la auto-equivarianza de un modelo son propiedades más accesibles para ser evaluadas empíricamente.

En el caso de las redes neuronales, otorgarle estas propiedades a un modelo puede ser difícil debido a su naturaleza de caja negra [CW16b; SG18]. La sección 2.7 analiza los modelos específicos que pueden dotar a las redes de invarianza o equivarianza.

2.7. Modelos de Redes Convolucionales con Invarianza y Equivarianza

En esta sección hacemos una revisión de modelos de CNN modificados para obtener invarianza a la rotación y a otras transformaciones. Hacemos foco en modelos de CNN que trabajan con imágenes.

Las capas convolucionales de una red aprenden un conjunto de filtros. La aplicación de uno de estos filtros puede considerarse una versión particular de una capa lineal. Para ello, en la capa lineal se comparten pesos en cada ubicación espacial, de modo que la operación de la capa lineal corresponde exactamente a una convolución [GBC16]. Por ende, podemos considerar a las convoluciones como casos especiales de las capas lineales. Las capas convolucionales se benefician de este esquema al obtener una mayor eficiencia por parámetro, así como un sesgo útil para procesar datos espaciales o temporales [GBC16].

Las redes feedforward tradicionales compuestas sólo por capas lineales y funciones de activación no lineales son muy expresivas y pueden aproximar cualquier función suave con una precisión arbitraria, si tienen la cantidad suficiente de parámetros [Hay94]. Por otro lado, las capas convolucionales pierden parte de esa expresividad, de forma de que individualmente no pueden codificar invarianzas o equivarianzas a conjuntos de funciones arbitrarias [DWD15]. Si bien las capas convolucionales son equivariantes a la traslación por diseño [DWD15], no está claro si una *serie* de capas convolucionales con funciones de activación no lineales pueden adquirir invarianza o equivarianza a transformaciones arbitrarias, pero hay poca evidencia que indica que pueden hacerlo al menos parcialmente con aumentación de datos [SG18].

Dependiendo de la aplicación la equivarianza puede requerirse en distintas capas de la red. Además, en algunos casos puede requerirse equivarianza. Por ejemplo, en una imagen de una persona si la mano de una persona puede rotar o trasladarse, pero no el cuerpo, el primero requiere invarianza y el segundo no [DDFK16]. En cualquier caso, las redes compuestas por capas convolucionales y/o lineales no pueden lidiar directamente con transformaciones arbitrarias en escenarios de entrenamiento e inferencia usuales. Hay dos enfoques principales para otorgar equivarianza a la red: *aumentación de datos*, o *modificar el modelo*.

La aumentación de datos es una forma de otorgar equivarianza a transformaciones de la entrada a una red. La aumentación de datos consiste en transformar los ejemplos de entrenamiento. Dicha transformación puede realizarse eligiendo una transformación aleatoria para un ejemplo cada vez que la red se entrena con el mismo. Alternativamente, se pueden pre-generar todas las transformaciones posibles

de cada ejemplo para obtener un conjunto de datos aumentado. Para cualquiera de los casos, si se utiliza el conjunto de transformaciones al que se busca lograr la invarianza, y se proveen suficientes ejemplos, dicha aumentación induce al modelo a obtener la invarianza. Un proceso similar puede realizarse para la equivarianza, al transformar tanto la entrada como la salida esperada de la red.

La equivarianza a las transformaciones mediante aumentación de datos ha sido estudiada ligeramente para las Máquinas de Boltzmann Restrictas [Lar+07] así como para los HOGs y las CNNs [LV14; SG18]. Estos resultados proveen evidencia de que las redes tradicionales pueden aprender automáticamente representaciones equivariantes con aumentación de datos. No obstante, en general esto implica utilizar un mayor presupuesto computacional, ya que deben explorar el espacio de transformaciones de la entrada.

El otro enfoque posible consiste en modificar el modelo o la arquitectura de CNN en lugar de la entrada para adquirir la equivarianza. Por ejemplo, algunos modelos emplean filtros invariantes a la transformaciones, o combinan múltiples predicciones realizadas con versiones transformadas de la entrada. En estas líneas, se han propuestos varias modificaciones para obtener equivarianza a las transformaciones [Lap+16; Zha+17; Lua+17; CW16b; CW16a; LB15; Dai+17; Zho+17; DWD15; Jad+15]. En varios casos, estos modelos requieren de todas formas un entrenamiento con aumentación de datos [CW16a], o incluyen un proceso interno similar [Lap+16].

Dentro del enfoque de modificar el modelo, hay varias opciones de diseño.

Algunos trabajos concluyen que para tareas de clasificación donde se requiere invarianza usualmente es preferible que las primeras capas de la red codifiquen equivarianzas, de modo que múltiples representaciones de la entrada puedan coexistir, y que las últimas capas de la red colapsen esas representaciones equivariantes a representaciones invariantes para realizar la clasificación final[LV15]. De esta forma, la red puede aprender las distintas versiones del objeto como entidades separadas, y luego convertir dichas representaciones en una única etiqueta de clase.

Alternativamente, podemos añadir un esquema explícito de transformación inversa que se aplica a la entrada antes de utilizarla como entrada a la red. Este esquema lleva a la entrada a una representación canónica, de modo que la red puede ignorar las transformaciones y aprender una única representación del objeto. Si bien esto simplifica el diseño de la red, el esquema de transformación inversa generalmente requiere un modelo extra que pueda predecir la transformación original.

El primer enfoque pone la invarianza cerca de la salida, mientras que el segundo lo pone cerca de la entrada. No obstante, como mencionamos anteriormente, en algunos dominios requerimos invarianza a sólo una parte de la entrada. Por ende,

para este tipo de problemas una representación canónica de toda entrada que sea invariante a las transformaciones puede no ser el mejor enfoque.

A continuación, realizamos un resumen de los modelos de CNN modificados para obtener equivarianza a las transformaciones. Los clasificamos en dos grupos:

- Modelos que lidian con el problema de forma global transformando la entrada a una representación canónica.
- Modelos que proponen modificar la capa convolucional de alguna manera para producir características equivariantes y, opcionalmente, convertir esas características en invariantes luego si es necesario.

2.7.1. Transformation de la imagen de entrada o feature map

Redes de Transformación Espacial (STN)

Las Redes de Transformación Espacial (STN, por Spatial Transformer Network) [Jad+15] son redes que utilizan un nuevo tipo de capa llamada Capa de Transformación Espacial (STL, por Spatial Transformer Layer) que puede aprender a transformar feature maps a una representación canónica de modo que las subsiguientes capas pueden enfocarse en la representación canónica. Las STLs pueden corregir transformaciones afines arbitrarias. Entonces, la capa STL contiene una sub-red que recibe los feature maps y genera un vector 6-dimensional que codifica la matriz de transformación afin. Utilizan muestreo bilineal para realizar las transformaciones, una operación que permite calcular el gradiente del error con respecto a los parámetros de la transformación. De esta forma, la transformación puede aprenderse con el mismo mecanismo de optimización que el resto de la red. Si bien las capas STL son modulares se pueden aplicar en cualquier parte de la red, generalmente se utilizan como primer capa.

Agrupamiento Invariante a las Transformaciones (TIP)

El Agrupamiento Invariante a las Transformaciones (TIP, por Transformation-Invariant Pooling) [Lap+16] propone definir un conjunto de transformaciones $= t_1, \dots, t_m$ a los cuales se desea que la red sea invariante. Luego, se entrena una red siamesa, que contiene una subred N_i por cada transformación t_i. Dada una entrada x a la red general, la entrada a la subred N_i es $t_i(x)$, es decir, cada subred recibe una versión transformada distinta de la entrada x. Las subredes forman parte de una red

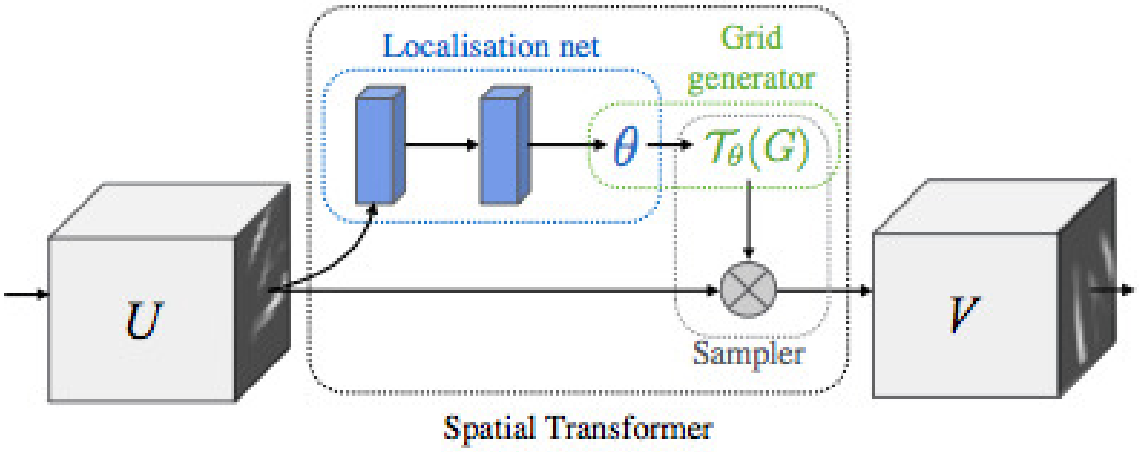


Figura 2.58: Arquitectura de una STN [Jad+15]. La capa STL transforma los feature maps U en feature maps V aplicando una transformación afín T. Los parámetros de la transformación afín θ los predice una sub-red de localización.

siamesa, es decir, comparten sus parámetros entre todas. En otras palabras, cada capa correspondiente de cada subred usa los mismos pesos.

El conjunto de transformaciones T es definido por el usuario; los autores demuestran que siempre que T forma un grupo algebraico entonces la salida del pooling será invariante a T [Lap+16].

Cada subred genera como salida feature maps, posiblemente distintos. Si el conjunto de transformaciones forma un grupo algebraico, entonces estos feature maps, considerados como conjunto, son equivariantes a la transformación, debido a que una transformación de la entrada corresponde a una permutación de las subredes y por ende de sus salidas.

Si bien las salidas de las subredes son distintas en valores, son iguales en estructura debido a la arquitectura siamesa. Por ende, es posible aplicar una operación de pooling entre los feature maps de salida, de modo de que quede uno solo. Esta operación colapsa la dimensión de los canales o feature maps, y no las dimensiones espaciales. De esta forma, se colapsa la equivarianza, obteniendo una representación invariante. Se puede utilizar una capa max-pooling para agrupar, que siempre dejará los valores máximos de todos los feature maps, o una avg-pooling, que calculará una representación promedio [Lap+16].

TIP puede ser considerado entonces como un modelo que embebe un procedimiento de aumentación de datos internamente tanto para el entrenamiento como para la inferencia.

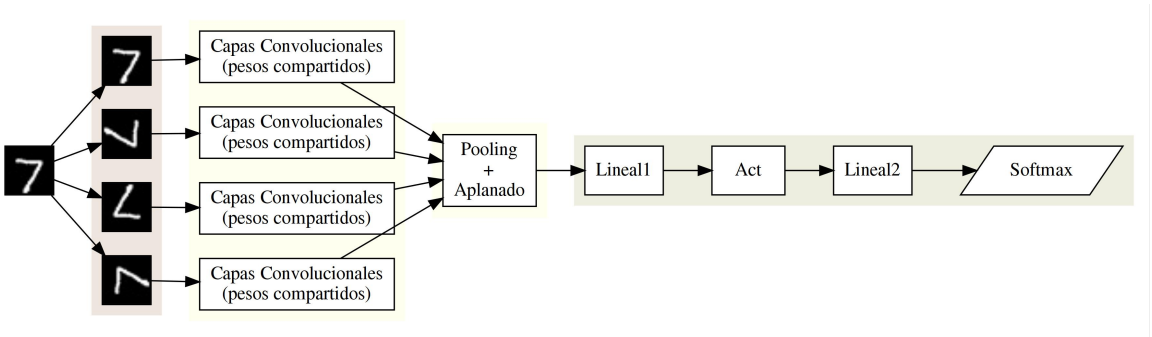


Figura 2.59: Ejemplo de un modelo TI-Pooling para un conjunto de transformaciones T con 4 rotaciones.

Redes Profundas de Simetría (DSN)

Las Redes Profundas de Simetría (DSN, por Deep Symmetry Networks) [GD14] también transforman la imagen antes de aplicar capas convolucionales. No obstante, para ello agregan un procedimiento de optimización iterativo sobre el espacio 6-dimensional de transformaciones afines, de modo de encontrar una transformación que activa de forma máxima a los filtros. Este procedimiento se realiza para cada nuevo ejemplar que pasa por la red, ya sea al entrenar o evaluar. De esta forma, DSN combina ideas de TIP y STN. En espíritu, su enfoque es más similar al de STN, pero el procedimiento de optimización es menos elegante que el entrenamiento end-to-end de la subred de localización, y es más costoso computacionalmente. También podría verse como una forma dinámica de aumentación de datos, de modo que el procedimiento de buscar una transformación es distinto al enfoque de STN, debido a que se realiza en tiempo de ejecución. Comparado contra aumentación de datos en las BDs MNIST y NORB, DSN tiene mejor desempeño al entrenar con menos de 10000 muestras. Con más datos de entrenamiento (ambos datasets tiene alrededor de 50000), la aumentación de datos regular obtiene un desempeño similar.

2.7.2. Modificaciones de la capa convolucional

Convoluciones Invertir-Rotar-Juntar (FRPC)

Las Convoluciones Invertir-Rotar-Juntar (FRPC, por Flip-Rotate-Pooling Convolutions) [WHK15] extienden las capas convolucionales comunes para obtener invarianza a la rotación. Una FRPC de F filtros o feature maps genera $F * R$ filtros dinámicamente al rotarlos con R ángulos distintos. Rotar el filtro en lugar de la imagen resulta más eficiente, y tiene el mismo efecto. Estos R nuevos filtros por cada filtro original se organizan en una nueva dimensión denominada la dimensión de la rotación, obteniendo feature maps con formato N, H, W, F, R, donde R es la nueva

dimensión. El sesgo b es el mismo para las R rotaciones. Luego, de forma similar a TIPooling, se utiliza pooling para eliminar la dimensión de rotación R, obteniendo entonces F filtros invariantes a la rotación. A mayor valor de R, mejor es la aproximación. No obstante, a diferencia de TIPooling, FRPC se plantea como una nueva capa convolucional, no como un modelo entero.

El hecho de que FRPC utilice filtros rotados en lugar de imágenes rotadas puede verse simplemente como una optimización de la idea general de utilizar distintas transformaciones de la entrada y luego utilizar una operación de pooling para unificarlas. Estas transformaciones, como en el caso de TIPooling, no tienen por qué restringirse a rotaciones, si bien FRPC utiliza un truco de optimización para mejorar la eficiencia computacional en este caso.

De hecho, las R versiones de cada filtro convolucional comparten sus pesos. Por ende, podemos ver a FRPC como una versión de TIPooling a nivel de capa. Esto permite ubicar capas FRPC en varios lugares distintos de la red de acuerdo a los requerimientos del dominio.

FRPC no ha sido comparado contra la técnica de aumentación de datos. De forma similar a TIPooling, si bien el número de parámetros de FRPC no se incrementa respecto de una CNN comparable, la cantidad de memoria en tiempo de ejecución y requisitos computaciones se ve multiplicada por R. No obstante, el número de feature maps F generalmente puede reducirse respecto de un modelo tradicional dado que cada filtro es más expresivo.

El mismo enfoque que FRPC se utiliza en [MVT16] para la clasificación de texturas. En tal caso, si compara la técnica contra aumentación de datos, pero sólo para clases con 20 muestras. Dado que el sesgo de rotación es muy fuerte en conjuntos de datos de textura, entendemos que es una comparación poco justa.

CNNs con Simetría Cíclica

Las CNNs con Simetría Cíclica (Exploiting Cyclic Symmetry in CNNs) introducen un método similar a FRPC, pero de forma similar que con TIPooling prueban que si el conjunto de transformaciones forma un grupo algebraico, entonces la salida de la capa antes de aplicar pooling es equivariante. Por ello, sugieren utilizar varias capas convolucionales modificadas pero sólo realizar el pooling de la dimensión de rotación en la última capa convolucional. La diferencia entonces entre una red TIPooling y una con simetría cíclica se encuentra en que en el primer modelo la transformación se aplica solo a la entrada, mientras que en el segundo también se aplica a los feature maps intermedios, pudiendo entonces aprender invarianzas y

equivarianzas por partes del objeto a clasificar.

Redes de Respuesta Orientada

Las redes de respuesta orientada (ORN, por Oriented response networks) [Zho+17] también son similares a FRPC, pero en lugar de una operación de pooling introducen una capa ORAlign que reordena los feature maps según su nivel de activación. Esta estrategia, basada en la utilizada por el descriptor SIFT [Low+99] le da invarianza al resultado final. Podemos considerarla una generalización del pooling con el máximo, ya que no solo se queda con el feature map más activado. El hecho de conservar todas las activaciones puede ser una ventaja en términos de información, pero también involucra un mayor coste computacional.

Por ende introducen además un parámetro p que indica qué porcentaje de los feature maps no se calcularán con la nueva estrategia y en lugar de eso mantienen la forma de cómputo tradicional, de manera de mantener un balance entre desempeño y coste computacional.

CNNs Equivariantes a un Grupo y CNNs Direccionables

Las CNNs Equivariantes a un Grupo (GCNN, por Group Equivariant Convolutional Networks GCNN) y CNNs Direccionables (Steerable CNNs) [CW16a; CW16b; WHS18] usan los mismos métodos básicos que FRPC, pero proveen una teoría formal para garantizar la equivarianza o invarianza de las representaciones en el caso en que las transformaciones formen un grupo algebraico. No obstante, se enfocan en un conjunto de rotaciones muy pequeño, el grupo p4m de 4 rotaciones y volteo horizontal de la imagen. También aplican la optimización de FRPC de transformar los filtros en lugar de los feature maps. Las CNNs esféricas generalizan este enfoque al caso de convoluciones 3D [CGW19].

Notamos que varios de los modelos anteriores siguen una estrategia similar a FRPC. Definen una nueva versión de la capa convolucional que 1) aplica varias transformaciones a la entrada 2) calcula la convolución con las versiones transformadas, generando una representación equivariante y 3) luego colapsa de alguna manera la representación equivariante a una invariante si es necesario.

Redes con Convoluciones Deformables

Las operaciones de convolución tradicionales utilizan filtros de tamaño rectangular $(h \times w)$. Las Redes con Convoluciones Deformables (Deformable Convolutional

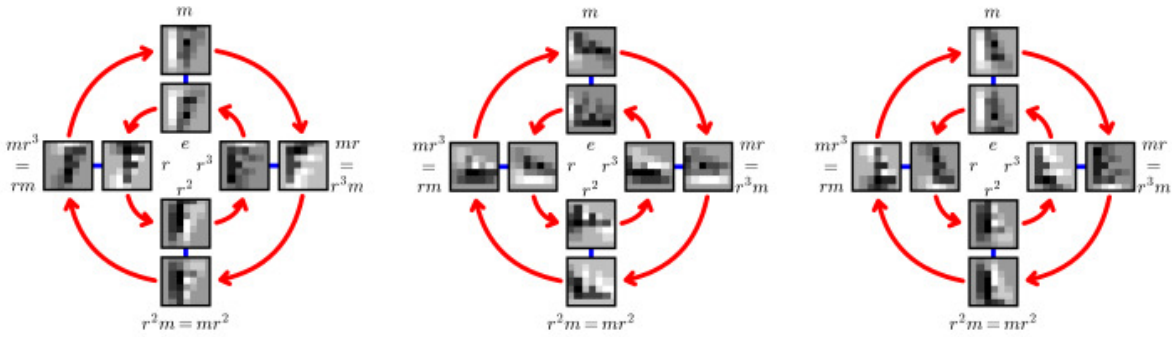


Figura 2.60: Transformaciones de un filtro de una CNN Equivariante a un Grupo [CW16a]. Los filtros se rotan y voltean, y luego se aplican a los feature maps. Estas operaciones forman un grupo.

Networks) [Dai+17] modifican la operación de convolución para aprender filtros de forma arbitraria. Para ello, utilizan convoluciones de tamaño rectangular, para cada una de sus posiciones se aprende una traslación durante el entrenamiento, efectivamente permitiendo referenciar cualquier pixel del feature map de entrada. De forma similar a STN, utiliza un muestreo bilineal para realizar las transformaciones. Como este muestreo es diferenciable, se pueden calcular los gradientes de los parámetros de traslación. Si bien este enfoque no está restringido a la rotación, es más general hasta que STN dado que no está limitado a transformaciones afines.

2.7.3. Resumen

En los últimos años se han propuesto diversos modelos de CNN para lidiar con invarianza y equivarianza. Varios de estos modelos son similares, con distintos enfoques sobre la misma idea: crear representaciones equivariantes, y luego colapsarlas a representaciones invariantes.

No obstante, para determinar cuáles de estas mejoras son más eficientes y relevantes, hace falta una comparación exhaustiva entre los modelos, así como un análisis más profundo sobre su funcionamiento.

La sección 2.8 analiza las métricas existentes de equivarianza para redes neuronales, que permiten el análisis de la estructura de la equivarianza de los modelos.

2.8. Métricas de Invarianza y Equivarianza

El uso de redes neuronales para problemas de clasificación típicamente utiliza arquitecturas feedforward. En particular, las CNNs utilizan una serie de capas convolucionales seguidas por una o dos capas lineales. Estos modelos, entrenados normalmente con descenso de gradiente estocástico y si aumentación de datos, no puede aprender invarianzas o equivarianzas a las transformaciones [AW18]. Las redes feedforward compuestas exclusivamente por capas lineales pueden aproximar cualquier función suave dada la suficiente cantidad de parámetros. Por otro lado, algunos modelos de CNN evitan completamente las capas lineales y al mismo tiempo logran un desempeño muy cercano al estado del arte [Spr+15], pero no han sido evaluados en problemas que requieran invarianza o equivarianza.

Como se menciona en la sección 2.7, hay dos mecanismos básicos para adquirir equivarianzas: aumentación de datos, o modificación del modelo. En ambos casos, los mecanismos que emplea la red para aprender la equivarianza no se entienden bien. Dado que la mayoría de los modelos requiere aumentación de datos durante el entrenamiento, o emplea algún esquema similar a la aumentación de datos en la modificación del modelo, no está claro cuales son las contribuciones relativas entre el modelo o los datos aumentados para adquirir las equivarianzas [SG18; AW18]. Además, muchas de las capas propuestas son equivariantes individualmente, pero no se han realizado análisis sobre la interacción entre las representaciones de distintas capas ni se ha evaluado si realmente esta equivarianza es útil para la red.

Las propiedades de equivarianza explican como un modelo reacciona a las transformaciones. Entender mejor la equivarianza de un modelo [Sha+16] o cualquier sistema [Buc+19] puede ayudarnos a mejorar su desempeño y robustez.

Por ende, es necesario contar con herramientas para poder analizar los modelos, de modo de comprender mejor de donde proviene la equivarianza. Dichas herramientas son las métricas de equivarianza. En esta sección presentamos las métricas preexistentes de equivarianza. Pueden distinguirse dos tipos principales de métricas: las basadas en el desempeño o tasa de acierto del modelo, y aquellas basadas en las activaciones o valores intermedios del modelo, es decir, las que utilizan las características internas del modelo.

2.8.1. Métricas basadas en el desempeño o tasa de aciertos de un modelo

Existen varios métodos propuestos para medir la equivarianza de una red neuronal de forma empírica a través del desempeño o tasa de acierto [Lar+07; Pen+14; FF15; Amo+18; BRW18; Kan17; Qui+18; Kau18]. La mayoría de las métricas se basan en comparar el desempeño de modelos invariantes contra aquellos que no lo son, o de analizar como varia la tasa de aciertos del modelo con cada transformación.

Los trabajos de [Kau18] y [SG18] analizan el efecto de utilizar distintos esquemas de aumentación de datos y arquitecturas CNNs. Específicamente, el mapa de sensitividad a la traslación desarrollado en [Kau18] relaciona la tasa de acierto de un clasificador con la posición central de un objeto en una imagen; [SG18] utiliza gráficos 3D para evaluar la equivarianza a varios tipos de transformaciones. [Kan17] y [FF15] se enfocan en algoritmos para determinar el conjunto de transformaciones al cual es invariante la red, pero también utilizan la tasa de aciertos como métrica indirecta de invarianza.

Por último, la invarianza a las transformaciones también fue investigada desde una perspectiva adversarial [Eng+17], cuantificando la intensidad de las perturbaciones requeridas para decrementar el desempeño de un clasificador.

Todos estos métodos tienen en común que solo analizan la equivarianza en relación a la tasa de acierto final, sin tomar en cuenta la representación interna de la red.

2.8.2. Métricas basadas en activaciones o valores intermedios

Métrica de invarianza de Goodfellow y otros

Goodfellow y otros [Goo+09] fueron los primeros en proponer una métrica de invarianza para las activaciones de una red en lugar de las salidas solamente. Recreamos aquí la definición de su métrica dado que es conceptualmente similar a las que desarrollaremos en el 4.

La métrica está definida en términos de la *tasa de disparo* de la s activaciones. Para ello, asumen que por cada activación a existe un umbral asociado t, de modo que si $a(x) > t$ entonces esa unidad está *disparando*. Su métrica $\mathrm{GF}(a)$ (Ecuación [2.21]) se define entonces como la proporción entre la tasa de disparo *local* denotada como $L(a)$ y la tasa de disparo *global* denotada como $G(a)$ (Ecuaciones [2.22] y [2.23]).

$$\mathrm{GF}(a) = \frac{L(a)}{G(a)} \tag{2.21}$$

$$L(a) = \frac{1}{|Z(a)|} \sum_{z \in Z} \frac{1}{|T(z)|} \tag{2.22}$$

$$G(a) = \mathop{E}_{x \sim P(x)} [f(a, t, x)] \tag{2.23}$$

Donde:

- $f(a, t, x) = \begin{cases} 1 & \text{if } a(x) > t \\ 0 & \text{otherwise} \end{cases}$.
- $P(x)$ representa la distribución sobre los ejemplos.
- $Z(a) = [x \mid a(x) > t]$ representa el conjunto de ejemplos que hacen disparar a a.
- $T(x) = [t(x, \gamma_0) \mid \gamma \in \Gamma]$ es el conjunto de versiones transformadas de x, donde Γ es el conjunto de parámetros de la transformación t.

Si bien la métrica está definida sobre un conjunto arbitrario X, en práctica la esperanza en Ecuación [2.23] se calcula sobre un conjunto finito de n ejemplos.

Entonces, $\mathrm{GF}(a)$ es la puntuación de invarianza de una sola activación. Para evaluar la invarianza de toda una red N, definen $Invp(N)$ como el promedio de las p activaciones con mayor puntuación de invarianza en la última capa de N. Las $1 - p$ activaciones menos invariantes se descartan bajo la hipótesis de que «las diferentes sub-poblaciones de activaciones pueden ser invariantes a transformaciones distintas». Si bien puede que esto sea cierto, no se utiliza ningún método para determinar el valor de p y ninguna evidencia que soporte esa hipótesis.

El umbral t, cuyo valor puede ser distinto para cada activación, se selecciona de modo que $G(a) = \alpha$, donde $\alpha = 0.01$ en sus experimentos. Por un lado, el criterio para para la selección de t presenta un problema de desempeño, ya que requiere calcular un percentil. El cálculo del percentil no puede hacerse a medida que se calculan las activaciones para cada ejemplo de forma exacta, sin utilizar una aproximación. Por ende, para n ejemplos y t transformaciones requiere tener los $O(n \times m)$ valores de la activación en memoria y $O(nlog(n))$ operaciones. Por otro lado, tampoco hay justificación en el uso del valor $\alpha = 0.01$ para determinar el valor de t. Aun más, el uso de un umbral y la noción de la *tasa de disparo* de una activación, aunque popular para modelos de neuronas biológicas [GH92], tiene valor límitado para arquitecturas

de redes neuronales modernas. Por último, dado que su trabajo fue presentado en 2009, con dicha métrica solo evaluaron arquitecturas de la época que utilizan pre-entrenamiento no supervisado y redes profundas pero de creencias. Además, los conjuntos de datos utilizados no son estándar en el área. A nuestro leal conocimiento, a la fecha solo [Sha+16] utilizó su método para evaluar la invarianza de modelos de forma limitada en los conjuntos de datos CIFAR10 e ImageNet para justificar la proposición de una nueva función de activación.

Métrica de equivarianza de Lenc y otros

Lenc y otros[LV14] evaluaron la equivarianza de los feature maps internos de una CNN con respecto a un conjunto de transformaciones T de la entrada. El método que proponen asume que dada una transformación t que actúa sobre las entradas, la transformación t' correspondiente que actúa sobre las salidas (Ecuación [2.20]) es afín, es decir, $t'(x) = A_t x + b_t$.

La métrica se calcula luego de entrenar el modelo. Para cada capa se asume una función t' distinta. Para determinar t' para una capa, se optimizan los pesos A_t y b_t de la transformación afín respecto del error total de la red, pero al calcular la salida de la red reemplazan $f(t(x)$ por $t'(f(x))$. El grado de equivarianza se mide comparando la función de error de la red original con esta nueva red con el reemplazo.

Para evaluar el grado de invarianza, utilizan A_t como indicador, Si A_t es la matriz identidad, entonces la representación debe ser invariante. Caso contrario, utilizan una función de distancia que determina que tan lejos está A_t de la matriz identidad; cuanto más grande es, consideran que es menos invariante.

Desde su publicación, y según nuestro conocimiento, dicha métrica solo ha sido utilizada una vez por [CT17], los cuales en base al análisis de la métrica modificación la función de error del modelo para mejorar su equivarianza e invarianza. No obstante, los autores solo estimaron el impacto del error en la última capa, es decir, con la tasa de error.

Esto sugiere que las dificultades de este método radican en que *i*) sólo lidia con transformaciones afines, con lo cual no puede modelar funciones t' no lineales *ii*) requiere un proceso de optimización *iii*) no es simple de interpretar y calcular.

2.8.3. Otras técnicas relacionadas

Análisis teóricos Algunos trabajos realizan también un análisis teórico de la equivarianza como en el caso de [AW18], donde estudian la falta de equivarianza en

algunas CNNs aplicando relaciones entre el teorema de muestreo de Shannon con las convoluciones espaciadas.

Evaluación cualitativa En muchos trabajos, se utiliza un enfoque cualitativo para determinar la equivarianza de las características o representaciones internas de un modelo. Por ejemplo, se han usado visualizaciones para entender la invarianza y otras propiedades como diversidad en los feature maps de CNNs [ZD19; ZF14; Cad+18]. No obstante, este enfoque está limitado en que se debe crear una visualización útil para cada tipo de característica.

Medida de la invarianza También conocida como **Invarianza Factorial** [VL00; SK08], es un campo bien establecido dentro de la estadística que busca proveer a los modelos estadísticos la propiedad de medir el mismo objeto en distintos grupos. Por ejemplo, sus técnicas pueden ser utilizadas para determinar si una cierta métrica es invariante a distintos grupos establecidos de acuerdo a la raza y el género. Para ello, analiza el comportamiento de sus variables latentes, tanto analítica como empíricamente. No obstante, los métodos de la Medida de la Invarianza están enfocados en técnicas para modelos estadísticos estándar como el Análisis de Factores Confirmatorios y no puede ser aplicado directamente a las redes neuronales [SK08].

Métricas de equivarianza ad-hoc En diversas áreas, se ha determinado o establecido como hipótesis que ciertos modelos o cantidades son equivariantes a ciertas variables [Buc+19]. A su vez esto incentiva a que se creen técnicas ad-hoc para verificar dichas hipótesis de equivarianza [Buc+19].

Otras métricas Si bien la equivarianza es una propiedad muy importante, otros autores han definido métricas para otras propiedades como la complejidad de las características, invertibilidad, selectividad, capacidad y atención [TC16; KWT17].

2.8.4. Resumen

En esta sección, presentamos varias métricas y técnicas relacionadas para medir la equivarianza de un modelo.

Con la excepción de los métodos de Goodfellow [Goo+09] y Lenc [LV14], todos los métodos mencionados se enfocan en medir como la tasa de acierto de la red varía de acuerdo a las transformaciones, arquitecturas, o esquemas de aumentación de datos utilizados, sin tomar en cuenta la representación interna. Además, sólo un

autor comparó la aumentación de datos con los modelos específicos de redes para invarianza [SG18], pero sólo en términos de la tasa de aciertos.

A nuestro leal saber, existen solo dos métricas propuestas de equivarianza interna de la red [Goo+09; LV14], las cuales sólo han sido usadas dos veces [Sha+16; CT17] luego de su publicación, entendemos que debido a su dificultad para computar e interpretar, y a la falta de implementaciones de público acceso.

2.9. Clasificación de formas de mano para el reconocimiento de Lengua de Señas

El reconocimiento automático de señas es un caso especial del reconocimiento de acciones o gestos. Es un problema multidisciplinar sumamente complejo que hoy en día sigue sin ser resuelto en forma total. Si bien en el último tiempo han habido avances en el reconocimiento de gestos, impulsados principalmente por el desarrollo de nuevas tecnologías, aún queda un largo camino por recorrer para construir aplicaciones precisas y robustas que permitan la traducción e interpretación de los gestos realizados por un intérprete [VA+08; CHB11; Qui+16a; Qui+16c; Bra+19].

La compleja naturaleza de los gestos motivan esfuerzos de diversas áreas de investigación como interacción hombre-máquina, visión por computador, análisis de movimientos, aprendizaje automático y reconocimiento de patrones. La Lengua de Señas, y particularmente la Lengua de Señas Argentina (LSA), es una temática muy impulsada actualmente por gobiernos y universidades para incluir a personas sordas en diferentes entornos.

Una particularidad de la lengua de señas es que cada región a nivel mundial tiene su propio léxico y grupo de señas que lo representan. Esto lo hace un problema diverso, y diferente de abordar en cada región, ya que nuevos gestos o configuraciones de manos involucran nuevos desafíos no contemplados con anterioridad. En particular, para la Lengua de Señas Argentina (LSA) prácticamente no existen sistemas y bases de datos que representen los gestos que posee.

Hay numerosas publicaciones sobre el reconocimiento automático de lengua de señas, un campo que comenzó mayormente en los años 90. Von Agris [VA+08], Cooper [CHB11] y Braggs [Bra+19] presentan revisiones completas sobre la problemática y el estado del arte del área.

La tarea completa de reconocer un gesto de la lengua de señas involucra diferentes pasos: la ubicación de las manos del intérprete, el reconocimiento de las formas de las manos (configuraciones), y el seguimiento de las manos para detectar el movimiento realizado, interpretación semántica y traducción al lenguaje escrito [CHB11].

La tarea completa de reconocer un gesto de la lengua de señas involucra diferentes pasos[CHB11; Bra+19], que pueden simplificarse como:

1. Localización del intérprete.
2. Localización y seguimiento de las manos y cabeza del intérprete.
3. Segmentación de las manos y creación de modelos de su forma.

4. Clasificación de las formas de las manos.
5. Clasificación de la expresión del rostro
6. Clasificación de una secuencia de expresiones de rostro y formas de mano en una seña.
7. Asignar semántica a una secuencia de señas.
8. Traducir dicha semántica a un lenguaje escrito.

Estas tareas pueden ser desarrolladas y evaluadas en forma separada ya que cada una tiene su complejidad particular.

El reconocimiento de lengua de señas utiliza distintos tipos de características, usualmente clasificadas como manuales y no manuales [CHB11]. Las características no manuales como la pose y la lectura de los labios o expresiones faciales muchas veces se incluyen en el proceso de reconocimiento dado que algunas señas no pueden diferenciarse solamente con información manual [VA+08]. No obstante, la información manual conlleva la mayor parte de la información de la seña, especialmente en la forma [Bra+19].

La información que provee la mano a una seña está compuesta por una secuencia de formas de mano. Si bien en la mayoría de las ocasiones dicha secuencia tiene longitud, es decir, una sola forma de mano, muchas señas utilizan secuencias de dos o tres formas de mano, las cuales deben modelarse con sus transiciones. No obstante, la conversión de una forma de mano en otra requiere una transformación no-rígida de la mano, lo cual también debe ser modelado. Capturar transformaciones 3D de la mano en la presencia de oclusiones utilizando una cámara RGB 2D normal es difícil [PB11]. El escenario óptimo para el reconocimiento entonces requiere múltiples cámaras, sensores especiales de profundidad u otros, o marcadores corporales [Bra+19]. No obstante, el uso de dichos dispositivos limita la aplicabilidad de un sistema de reconocimiento de lengua de señas.

En la literatura existen numerosos trabajos desarrollados que abordan el reconocimiento automático de lengua de señas. No obstante, cada trabajo presenta un escenario particular, a veces difícil de replicar completamente, o con ciertas limitaciones. Por ejemplo, diferente trabajos utilizan sensores de profundidad como el MS Kinect, o similares para capturar imágenes 3D. En [PB11],[ZYT13] y [RMG14] se utilizan imágenes de profundidad para clasificar configuraciones de la lengua de señas norteamericano (ASL). Estos enfoques en general presentan dos problemas: por un lado la necesidad de contar con un equipo de similares características con el que fue probado, y por otro lado la alta tasa de error que todavía tienen estos dispositivos (al menos los de un costo bajo) para calcular las imágenes de profundidad.

La mayoría de los enfoques utilizan sólo imágenes RGB para el reconocimiento. En [Rou+10a] se crea un modelo probabilístico de color de piel para detectar y seguir las manos del intérprete en un video. En [Coo+12] se utiliza este modelo para segmentar las manos y aplicar un clasificador basado en Modelos de Markov. En general los sistemas basados únicamente en color de piel no son robustos a la variabilidad en el fondo o la vestimenta del intérprete, y en las oclusiones mano-mano o mano-cara. Para realizar un reconocimiento de la posición de la mano suele ser necesario adicionar información morfológica al filtrado de color.

2.9.1. Clasificación de formas de mano

Los análisis de sistemas de reconocimiento de lengua de señas muestran evidencias que la clasificación de la forma de la mano es el paso más importante para obtener un buen desempeño [KNB16; Qui+16b; Bra+19]. La mayoría de la literatura sobre este subproblema está enfocada en encontrar descriptores adecuados para formas de mano, que luego serán clasificados con modelos como las máquinas de vectores de soporte o redes neuronales tradicionales (no profundas). Por ende, las siguientes secciones ofrecen un resumen de los distintos tipos de descriptores, haciendo énfasis en su desempeño y su aptitud para lidiar con transformaciones.

Descriptores geométricos de la mano

Los descriptores geométricos son simples de definir y computar y se utilizan en diversas investigaciones. Generalmente se calculan sobre el contorno de la mano, y requieren algún tipo de algoritmo de filtrado de imágenes previo que permita identificar el contorno. Esto es una tarea relativamente sencilla hoy en día. Existen numerosos filtros llamados "filtros de detección de bordes" que realizan esta tarea eficientemente. Luego, diversas características de la forma pueden calcularse. Las más comunes son: el área, centro de coordenadas, orientación del eje principal, excentricidad, compacidad, etc. Un ejemplo de estos descriptores puede encontrarse en [VA+08; Ron18]. La figura 2.61 muestra un esquema gráfico de este enfoque.

Descriptores de Fourier

Los descriptores de Fourier han sido aplicados exitosamente a muchas tareas de visión por computador [Gra72], principalmente como representación de formas en trabajos de reconocimiento óptico de caracteres y otros problemas de clasificación de imágenes. Al ser robustos al ruido, además de ser fáciles de derivar y simple de

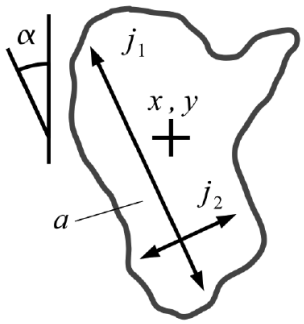


Figura 2.61: Descriptores geométricos, tomados de [VA+08].

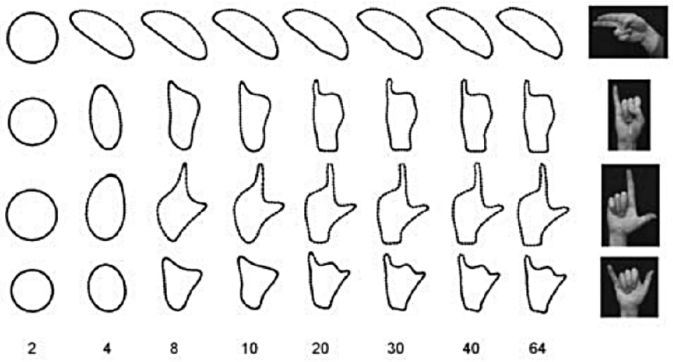


Figura 2.62: Descriptores de Fourier para la forma de mano, tomados de [CBM07].

normalizar, han sido utilizados en numerosas aplicaciones. En 1972, Granlund los utilizó para reconocer huellas digitales debido a su simplicidad describiendo contornos [Gra72]. Para el caso de configuraciones de manos, la idea es estimar los coeficientes de Fourier dado el contorno. Al aplicar la transformada de Fourier a una imagen particular del contorno de una mano, se generan una serie de coeficientes, que describen la forma que posee el contorno en el dominio de la frecuencia.

La Figura 2.62 muestra esto con un ejemplo. Las frecuencias más bajas codifican la curvas más suaves, la idea general del contorno, mientras que las frecuencias más altas codifican los detalles. Una particularidad de estos coeficientes para describir contornos de manos, es que resultan invariantes a la rotación, escala y traslación [Tan+14]. No obstante esta invarianza, no otorgan un buen desempeño para el reconocimiento de formas de mano [GAeS13; Tan+14] en general debido al poco poder discriminante que poseen.

SIFT (Scale-invariant feature transform)

Un descriptor SIFT es un histograma espacial 3D de los gradientes de una imagen, que caracteriza la apariencia de un punto de interés. Para ello, con el gradiente

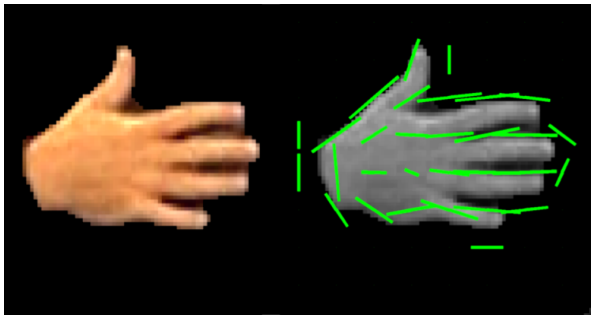


Figura 2.63: Descriptores de HOG para la forma de mano, tomados de [BZE09]

de cada pixel se calcula un descriptor más elemental formado por la ubicación del pixel y la orientación del gradiente. La técnica fue propuesta en 1999 [Low+99] para detectar regiones descriptivas en una imagen y así poder realizar segmentación de objetos, o seguimiento en video. Dado un posible punto de interés, estos descriptores elementales son pesados por la norma del gradiente y acumulados en un histograma 3D que representa el descriptor SIFT de la región alrededor del punto de interés. Al formar el histograma, se le aplica a los descriptores elementales una función de peso gaussiana para darle menos importancia a los gradientes que están más lejos del centro del punto de interés.

Los descriptores SIFT han sido aplicados a varias tareas de visión por computadoras, incluyendo el reconocimiento de configuraciones de mano [ZW12] y reconocimiento de rostros [Lan+13].

Histogramas de gradientes orientados. (Histogram of Oriented Gradients - HOG)

En relación con los vectores SIFT, debido a su búsqueda de contexto de formas, los vectores HOGs caracterizan una imagen por la distribución de los gradientes de una serie de regiones. El descriptor divide la imagen en pequeñas regiones cuadradas llamadas celdas, donde a cada una le calcula las directrices de los gradientes para los pixeles dentro de ella. El resultado es una serie de vectores que indican la magnitud de los gradientes para cada celda. Esto da la posibilidad de describir los diferentes contornos que hay en una imagen. La cantidad de orientaciones que se calculan es un parámetro del algoritmo, siendo lo normal 9 segmentos de orientación, aunque existen trabajos con menos y con más, dependiendo lo que se desee obtener. Para evitar disparidades debido a variaciones en iluminación, los histogramas de las celdas se normalizan en bloques más grandes que contienen varias de ellas.

[BZE09] utiliza descriptores HOGs para describir cada mano segmentada por separado en un entorno de reconocimiento de lengua de señas. Cada imagen tiene una resolución de 80x80 pixeles y para calcular los HOGs utilizaron una grilla de

10x10 celdas con 4 orientaciones (Figura 2.61). [CHB11] utiliza estos descriptores pero con una resolución de 8x8 celdas de 32x32 pixeles cada una, con 9 segmentos de orientación.

Si bien los descriptores HOG pueden encontrarse en numerosos trabajos logrados eficazmente, traen aparejada la desventaja de tener una fuerte dependencia a la traslación y rotación, debido a su naturaleza espacial dentro de la imagen. Esto, generalmente trata de evitarse rotando con anterioridad la imagen, para llevarla a una orientación canónica. En ocasiones, estos descriptores se utilizan para detectar una objeto conocido, como una persona dentro de una imagen, pero su aplicación para clasificar diferentes configuraciones de manos no es trivial.

Transformada de Radon

La transformada de Radon definida en el espacio R2 para aplicar a imágenes digitales, se define como una integral de linea sobre la imagen. La idea se basa en integrar diferentes lineas con una ordenada al origen y un ángulo determinado. El resultado es un descriptor con información de frecuencia de puntos para diferentes ángulos. Para reconocimiento de configuraciones, resulta muy útil, sobre todo si se utiliza el contorno de la mano, ya que es posible encontrar rápidamente las zonas donde hay mucha frecuencia de lineas, es decir, donde están los dedos de las manos. Ha sido muy utilizada en reconstrucción de imágenes, con particular aplicación a reconstrucción de imágenes médicas [NW01]. Ha sido utilizada también para reconocer objetos y para identificar a personas en base a las características de su mano. Por ejemplo, en [GCC13] se aplica la transformada de Radón al contorno de la mano para realizar un proceso de autenticación biométrica. En [Mos+09] se utiliza un enfoque similar, pero se calcula el ángulo óptimo y solo se computa la transformada con ese ángulo, dejando un descriptor en forma de vector, en lugar de ser una matriz. En [Qui+16a] se utiliza para la clasificación de formas de mano con resultados al nivel del estado del arte. No obstante, la transformada de radon no es invariante a las transformaciones afines.

Otros descriptores para configuraciones

Existen diversos trabajos donde se utilizan otros descriptores a los presentados anteriormente. En [OB04] presentan una combinación de detección de manos junto con un clasificador de configuraciones, basado en un clasificador en cascada tipo boosting. Los niveles más bajos del árbol de clasificadores permiten separar la imagen de la mano entre varios conjuntos posibles utilizando distancia basada en

contexto de formas. En [Yan10] se propone una combinación de cinco descriptores distintos para caracterizar aspectos geométricos y visuales: histogramas de color, momentos Hu, wavlet de Gabor, descriptores de Fourier y descriptores SIFT. El sistema luego es utilizado para reconocer configuraciones de la lengua de señas china. En [KTN15] proponen usar los autovalores de los momentos HU, luego de segmentar y calcular el contorno de las manos, en un entorno de reconocimiento de lengua de señas irlandesa. En [Rou+10b] proponen un complejo proceso de caracterización de la forma de la mano a través de un novedoso método que los autores proponen como un balance entre "forma" y "apariencia". El método combina una modificación de Modelos de Apariencia Activa [CET01] con un modelado explícito de variación de poses de las manos incorporando transformaciones afines de las imágenes. El método, luego de realizar una aproximación de la imagen segmentada a los diferentes modelos de configuraciones que posee, realiza una reducción de dimensionalidad utilizando Análisis de Componentes Principales (PCA). No obstante, el desempeño de este modelo no es óptimo.

2.9.2. Resumen

El reconocimiento de lengua de señas, un caso particular del reconocimiento de gestos o acciones, permite traducir un video donde un intérprete se comunica en lengua de señas a un lenguaje escrito. Para crear un sistema automático de reconocimiento de señas, el paso más importante es la clasificación de las formas de las manos.

Previo al uso de redes convolucionales, se han desarrollado varios descriptores o características de formas de mano para reconocerlas con mayor tasa de acierto. Con el desarrollo de las CNNs, se han comenzado a aplicar para clasificar imágenes de formas de mano, reemplazando paulatinamente a los descriptores diseñados explícitamente.

No obstante, no existen comparaciones exhaustivas que determinen qué tipo de arquitectura convolucional se adapta mejor a la clasificación de formas de mano. Más aún, en el contexto de reconocer dichas formas con invarianza a varias transformaciones afines, no hay experiencias previas con CNNs.

Capítulo 3

Modelos Invariantes vs Aumentación de Datos

Se han propuesto varias modificaciones de CNNs para lidiar con equivarianza a la rotación y a otras transformaciones, como se estableció en la sección sección 2.7. No obstante, no está claro si éstas modificaciones representan una mejora ante las CNNs tradicionales entrenadas con aumentación de datos, tanto en términos de eficiencia como de poder de representación. Aún más, los mecanismos mediante los cuales las CNNs tradicionales adquieren equivarianzas no se conocen, así como cuál es la mejor estrategia para aumentar los datos para adquirir equivarianza.

Este capítulo compara modelos de CNN modificados con aumentación de datos para obtener invarianza rotacional para clasificación de imágenes, con el objetivo de determinar cual de estas estrategias es más conveniente para lograr la invarianza, y comprender mejor sus características en el caso de aumentación de datos.

3.1. Metodología

Realizamos dos tipos de experimentos, en ambos casos utilizando las bien conocidas BDs MNIST y CIFAR10 (sección 2.5). En el primero, comparamos el desempeño de los modelos específicos contra la aumentación de datos. En el segundo, evaluamos estrategias para re-utilizar modelos pre-entrenados y otorgarles invarianza a la rotación.

A continuación, describimos las transformaciones y modelos utilizados.

3.1.1. Transformaciones

El esquema de aumentación de datos que usamos consiste en rotar la imagen de entrada con un ángulo aleatorio en el rango $[0°, 360°)$, discretizado en 32 valores posibles de manera uniforme. Para la BD MNIST, varios trabajos previos utilizan la versión MNISTrot en donde las imágenes han sido rotadas en 8 ángulos fijos, pero en este caso decidimos aumentar la cantidad de transformaciones para proveer un conjunto de prueba más real. Sólo usamos rotaciones globales, es decir, rotaciones de toda la imagen alrededor del centro.

En todos los experimentos, entrenamos los modelos hasta la convergencia monitoreando la tasa de acierto promedio del conjunto de prueba. Utilizamos el algoritmo de optimización ADAM con una tasa de aprendizaje base de 1e−4 y decadencia de pesos 1e−9 para todos los parámetros excepto los sesgos.

3.1.2. Modelos

Como línea de base, utilizamos un modelo simple de CNN llamado SimpleConv (sección 2.4). Para MNIST la cantidad de filtros base es $F = 32$, y la dimensionalidad de la capa lineal es $D = 64$. Para CIFAR10, $F = 64$ y $D = 128$, debido a la complejidad extra de esta BD.

Para evaluar la importancia de las capas lineales en proveer invarianza a la rotación, también experimentamos con el modelo AllConvolutional, que sólo utiliza capas convolucionales (sección 2.4). Para MNIST, la cantidad de filtros convolucionales de base es $F = 16$, mientras que para CIFAR10 es $F = 96$.

Como modelos invariantes a la rotación, elegimos dos de cada uno de los grupos descriptos en la sección 2.7.

Como representante del enfoque de transformar la imagen de entrada, que corresponde a poner la invarianza al principio de la red, utilizamos una capa STL del modelo STN (sección 2.7.1) para re-orientar la imagen antes de que la red la clasifique. Dicha capa se coloca como primer capa de los modelos SimpleConv y AllConvolutional, para generar las versiones SimpleConvSTN y AllConvolutionalSTN. Dado que STL puede calcular transformaciones afines arbitrarias, para que la comparación sea justa restringimos los coeficientes a estimar de modo que la matriz de transformación afín resultante solo represente rotaciones. En ambos casos, la red de localización que estima el ángulo de rotación de la capa STL es una CNN simple con la siguiente topología: $Conv1(16, 7 \times 7)$–ELU–$MaxPool1(2 \times 2)$–$ELU - Conv2(16, 5 \times 5) - MaxPool2(2 \times 2)$–$ELU - Lineal(32) - ELU - Lineal(1)$.

Como representante del enfoque de modificar la capa convolucional, que corresponde a poner la invarianza al final de la red, elegimos el modelo GCNN (sección 2.7.2). Las redes resultantes se denominan SimpleConvGCNN y AllConvolutionalGCNN, donde simplemente reemplazamos la convoluciones normales por capas de convolución de grupo, y agregamos una operación de pooling de la dimensión de transformación antes de las capas de clasificación para proveer la invarianza a la rotación requerida. Las capas convolucionales GCNN generan 4 veces más feature maps que las convoluciones normales. Para compensar esta ventaja, redujimos la cantidad de filtros a la mitad. Si bien en términos de número de feature maps la red GCNN sigue generando el doble que su versión común, la diversidad posible de GCNN es menor. Entonces, usar la mitad de los filtros originales es un punto de compromiso.

A continuación, presentamos los resultados de los experimentos.

3.2. Desempeño con aumentación de datos

Para comenzar, evaluamos la tasa de acierto promedio de los modelos SimpleConv y AllConvolutional con y sin aumentación de datos para obtener una línea de base de ambos modelos.

Entonces, entrenamos dos instancias de cada modelo: una con la BD original, y otra con una versión aumentada mediante rotaciones. Luego, evaluamos cada instancia de cada modelo con el conjunto de prueba de las dos versiones de la BD. La Figura 3.1 muestra los resultados del experimento para cada combinación de modelo/BD.

En MNIST, podemos observar que si bien las redes entrenadas con la BD rotada tienen una tasa de acierto promedio menor, el decremento es menor que 2 %. Es sorprendente que en el primer caso el desempeño sea tan parecido, sobre todo dado que el número de parámetros es el mismo para las dos redes. Esto puede indicar una redundancia en los filtros del modelo sin rotar.

Por otro lado, la pérdida de desempeño para las redes entrenadas con la BD sin rotar y probadas con la BD rotada son mucho mayores (decremento del 55 %). Es interesante notar, no obstante que el modelo tiene una tasa de acierto promedio de aproximadamente 40 %, lo cual es mucho mayor que lo esperado de forma aleatoria (10 %, ya que las BDs tienen 10 clases). Esto se debe parcialmente a que algunas de las muestras de MNIST son naturalmente invariantes a las rotaciones, como el número 0, o parcialmente invariantes a las rotaciones como los números 1 y 8. Además, algunas de las características aprendidas podrían ser naturalmente invariantes a la rotación,

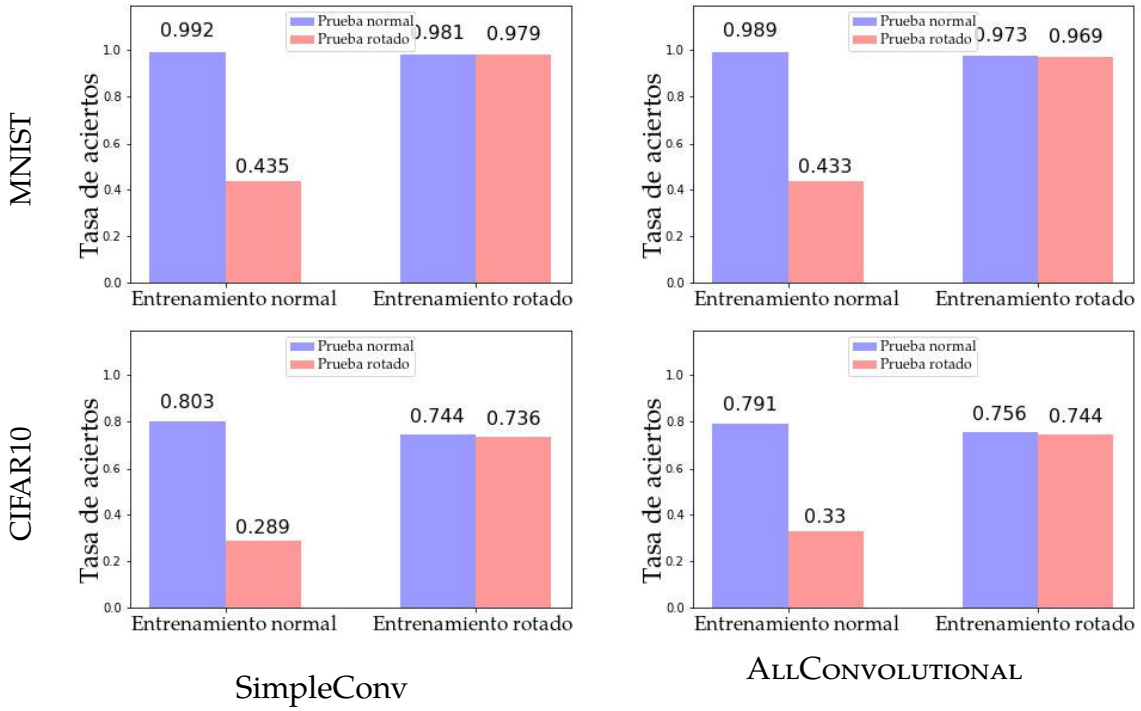


Figura 3.1: Tasas de acierto para el conjunto de prueba, para las dos versiones de la BD (normal o sin rotar, y rotada), y los modelos convolucionales tradicionales.

aún cuando no haya un sesgo específico en el conjunto de datos o entrenamiento para eso.

En el caso de CIFAR10, los resultados muestran una situación similar, aunque la pérdida de desempeño al pasar de una BD sin rotar a una rotada es aún mayor. Una posibilidad es que este fenómeno se deba a que el conjunto de datos posee menos invarianzas naturales que MNIST. Otra posibilidad es que CIFAR10 es un conjunto de datos más complicado y las redes no obtienen un desempeño tan bueno como en MNIST.

Debemos notar que para realizar una comparación justa redujimos el número de épocas de entrenamiento de AllConvolutional, alcanzando entonces una tasa de acierto promedio de 80 % en lugar del 91 % reportado en [Jad+15]. No obstante, es sorprendente que la red AllConvolutional pueda aprender con tanto éxito la versión rotada de las BDs, dado que las convoluciones individualmente no son ni invariantes ni equivariantes a la rotación. Esto sugiere que el conjunto de filtros de una red puede auto-organizarse durante el entrenamiento para representar todas las variaciones rotadas de un objeto.

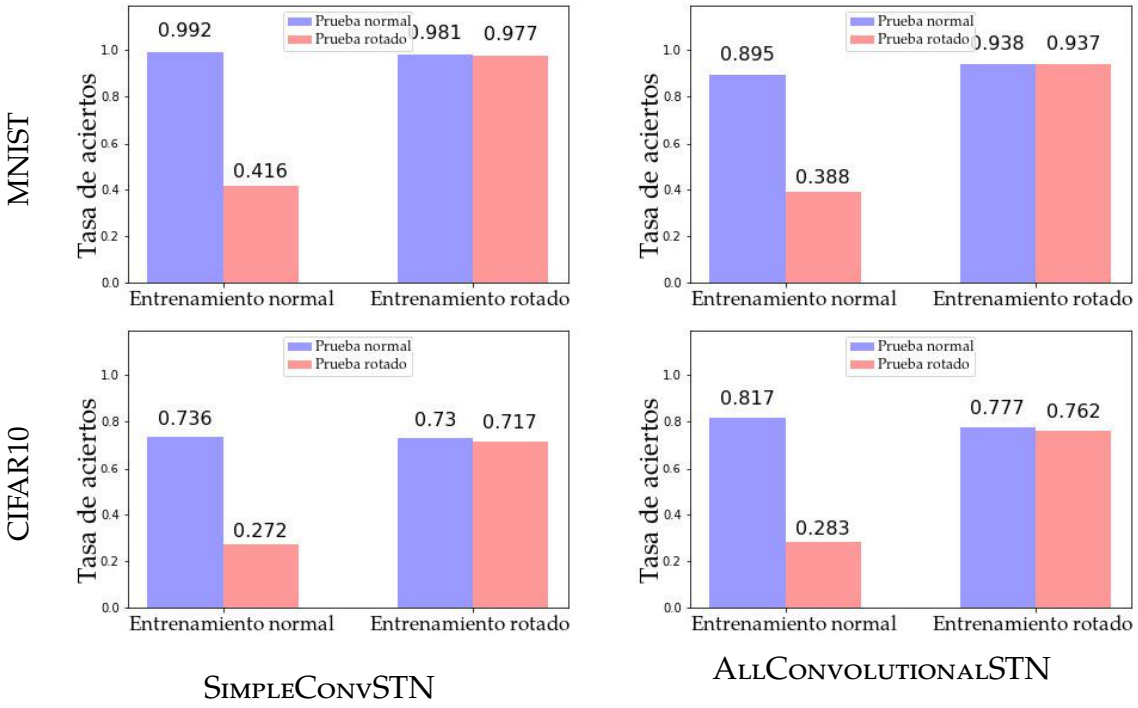


Figura 3.2: Tasas de acierto para el conjunto de prueba, para las dos versiones de la BD (normal o sin rotar, y rotada), y los modelos convolucionales con una capa STL.

3.3. Comparación con STN y GCNN

Utilizando las versiones modificadas de las redes con las arquitecturas STN y GCNN, repetimos los experimentos anteriores. La Figura 3.2 muestra los resultados de los modelos con capas STN.

Podemos observar que en todos los casos el desempeño del modelo sin rotar en la BD rotada es mucho peor que en el original. Esto es esperable dado que la capa STL requiere aumentación de datos para aprender la orientación canónica. De hecho, el desempeño del modelo AllConvolutionalSTN es ligeramente inferior en el conjunto de datos MNIST respecto de los otros modelos Además, el modelo no tiene un desempeño notablemente superior al de los modelos con aumentación de datos (Figura 3.1).

La Figura 3.3 muestra los mismos resultados para los modelos basados en GCNN. Al igual que con las redes STN, podemos ver que el modelo entrenado sin ejemplos rotados tiene un desempeño muy inferior al modelo entrenado con aumentación de datos. No obstante, vemos un aumento significativo (+10 %) en la tasa de acierto promedio para este caso en particular, pero sólo en la arquitectura AllConvolutional. Es posible que esto se deba a que la capacidad superior de representación de las capas lineales en el modelo SimpleConv compensen la falta de calidad de las capas convolucionales, ignorando las ventajas de las convoluciones GCNN. En el caso de

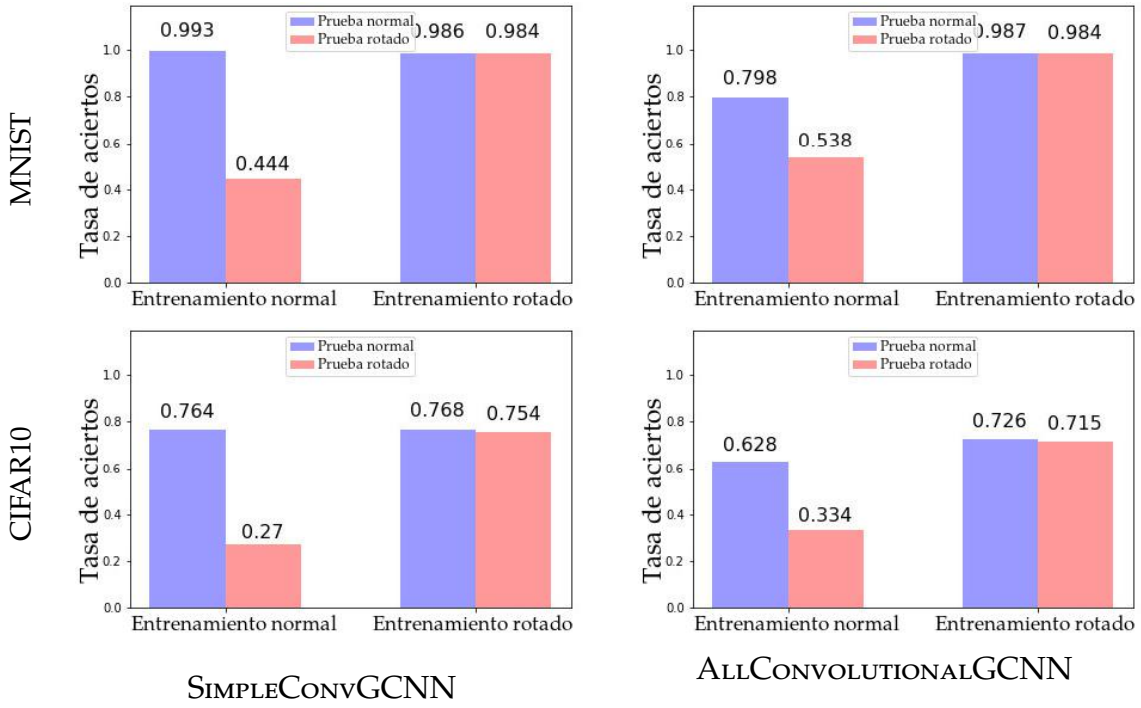


Figura 3.3: Tasas de acierto para el conjunto de prueba, para las dos versiones de la BD (normal o sin rotar, y rotada), y los modelos con convoluciones GCNN.

AllConvolutional, hay más presión para que las capas convolucionales junto con GCNN aprendan buenas representaciones.

Es evidente entonces que el mecanismo de GCNN puede otorgar un sesgo hacia la invarianza a la red, pero no le permite ser invariante sin un esquema de aumentación de datos. De todas formas, aún utilizando aumentación de datos, y de forma análoga al caso de STN, el uso de convoluciones GCNN con aumentación de datos no mejora el desempeño con respecto a simplemente utilizar aumentación de datos.

3.4. Evaluación de aumentación de datos para invarianza con distintas transformaciones

En las secciones anteriores establecemos que en el caso de la invarianza a la rotación, la aumentación de datos puede ser una alternativa útil a los modelos específicos. En esta sección, estudiamos la invarianza de modelos entrenados con aumentación de datos utilizando además otras transformaciones afines, como los escalados, las traslaciones, y combinaciones de las tres. De esta forma, podemos determinar si el efecto encontrado se restringe a las rotaciones o es válido para otras transformaciones también.

3.4.1. Metodología

Para simplificar, restringimos los modelos a evaluar a SimpleConv, AllConvolutional, VGG16D y ResNet18. En estos experimentos, utilizamos una configuración de entrenamiento similar a la de la sección anterior, donde entrenamos los modelos utilizando aumentación de datos con varios conjuntos de transformaciones.

Primero entrenamos cada modelo sin aumentación de datos. Luego, por cada conjunto de transformaciones, entrenamos los modelos con aumentación de datos a esas transformaciones. Finalmente, los evaluamos a todos utilizando datos sin transformar y datos transformados.

Además, en todos los casos utilizamos el optimizador AdamW [LH19], una tasa de aprendizaje de 0.001, y elegimos una cantidad de iteraciones base para cada modelo de forma que siempre sea suficiente para que converja. Para los modelos entrenados con aumentación de datos dicha cantidad de iteraciones base se multiplica por $log(m)$, donde m es el tamaño del conjunto de transformaciones. Dado que nuestro objetivo es obtener una idea del comportamiento general de los modelos ante las transformaciones y no nos interesa obtener un desempeño del estado del arte, buscamos un conjunto de hiperparámetros que funcionen bien para todo el conjunto de modelos entrenados. En este caso, buscamos que la tasa de aciertos mínima sea de 0.9 para MNIST y 0.75 para CIFAR10.

Utilizamos cuatro conjuntos de transformaciones: rotaciones, escalados, traslaciones y una combinación de las anteriores.

1. **Rotación** (16 transformaciones). Rotaciones desde 0° a 360°, discretizadas en 16 ángulos. Las rotaciones se realizan siempre utilizando el centro de la imagen como origen.

2. **Escalado** (13 transformaciones). Escalamos las imágenes con los coeficientes de escala: 0.40, 0.50, ...1.0, 1.05, ..., 1.30.

3. **Traslaciones** (48 transformaciones): Generamos todas las traslaciones de $d = 2^i$ píxeles con el siguiente esquema: $(-d,-d)$, $(-d,d)$, $(d,-d)$, (d,d), $(0,d)$, $(d,0)$, $(0,-d)$, $(-d,0)$. Los valores de i utilizados son $0,1,2,3,4,5$, para un total de $8 \times 6 = 48$ transformaciones.

4. **Combinadas** (9984 transformaciones): Generamos todas las posibles combinaciones de las transformaciones anteriores, para un total de $16 \times 13 \times 48 = 9984$ transformaciones.

La Figura 3.4 muestra ejemplos de ambas bases de datos a lo cuales se les aplican estas transformaciones.

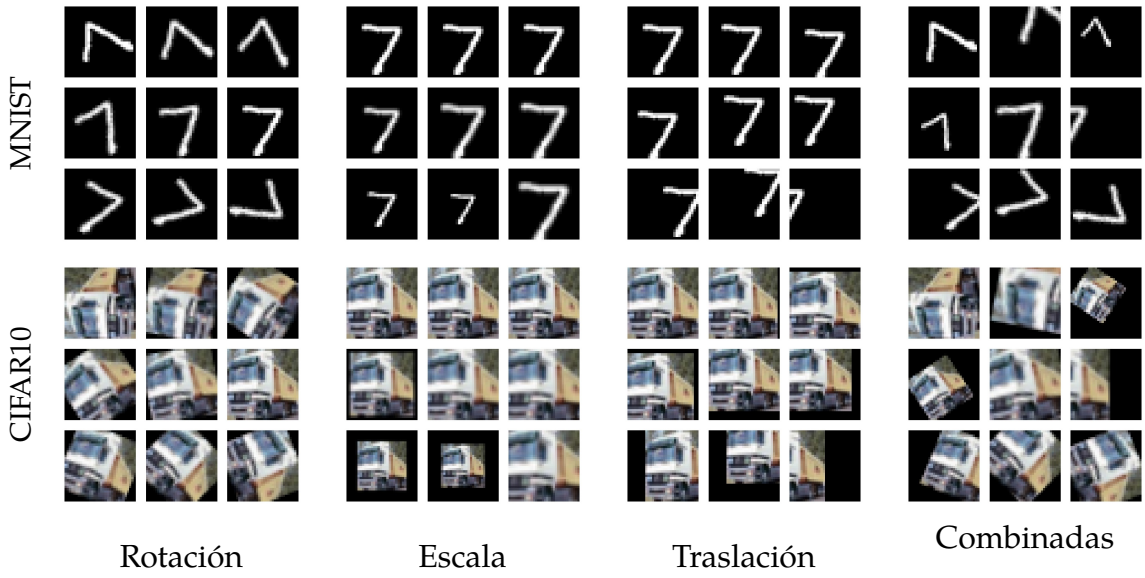


Figura 3.4: Muestras transformadas de las bases de datos CIFAR10 y MNIST.

Para evaluar la tasa de aciertos, utilizamos el conjunto de prueba, donde cada ejemplo es transformado por una de las transformaciones elegida de forma aleatoria. De este modo, si el conjunto de prueba tiene n ejemplos, entonces se evalúa al modelo con n ejemplos, donde cada ejemplo es evaluado con una transformación aleatoria (posiblemente) distinta.

3.4.2. Resultados

Para estas transformaciones podemos observar (Figura 3.5) también como los modelos entrenados sin aumentación de datos tienen tasas de acierto mucho menores al ser evaluados en conjuntos de datos transformados. Este efecto es mayor para las rotaciones, y menor para otras transformaciones, lo cual parece proporcional a la intensidad de las transformaciones (Figura 3.5). En todos los casos podemos observar que la aumentación de datos recupera el desempeño de los modelos normales tanto en el conjunto de prueba normal como en el transformado. Notablemente, este resultado también se aplica al caso de las transformaciones combinadas, que representan un conjunto de transformaciones muy complejas. Esto indica que los resultados obtenidos en la sección 3.2 son generalizables a distintas transformaciones.

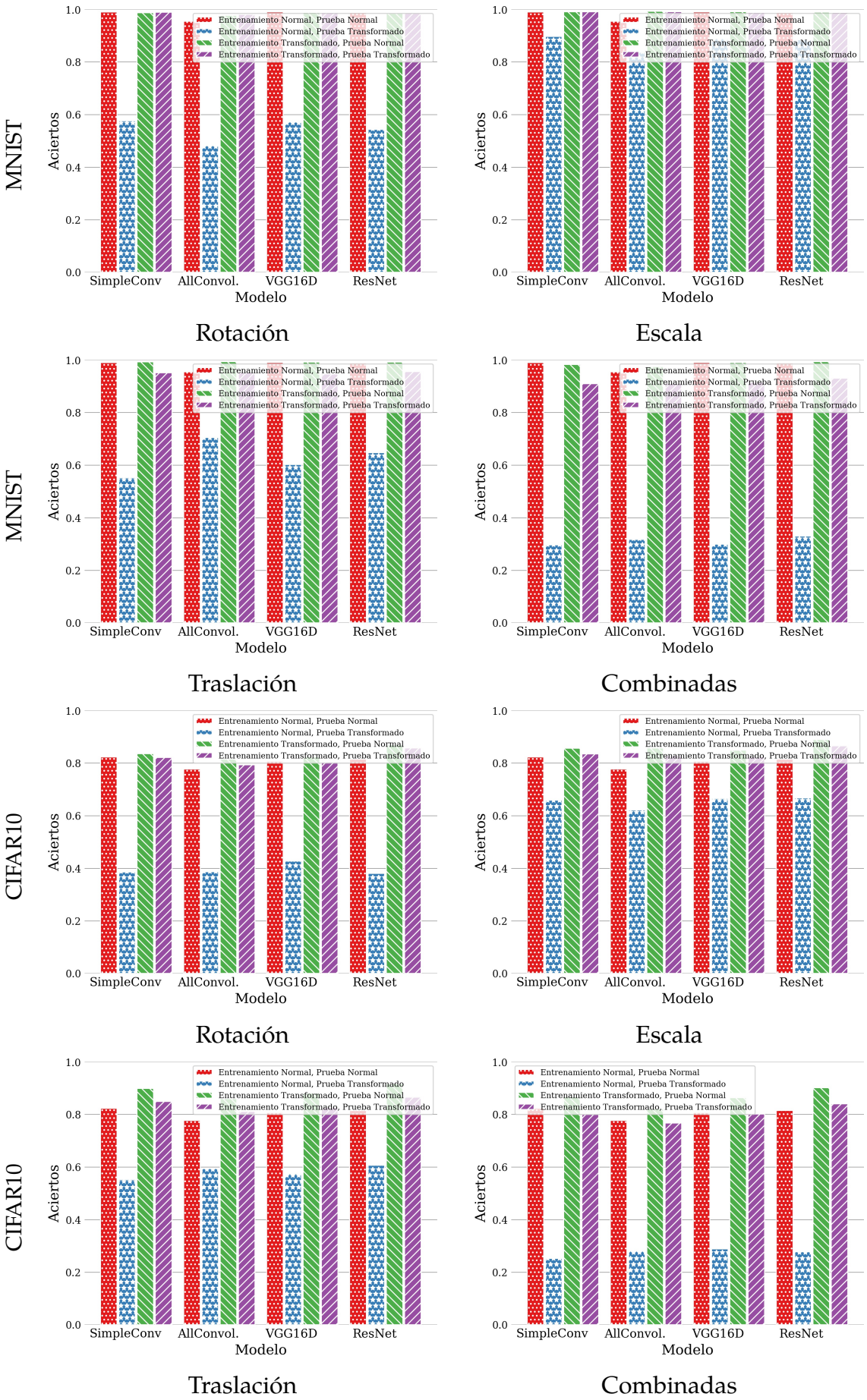


Figura 3.5: Tasas de acierto para varios tipos de transformaciones de aumentación de datos en las bases de datos MNIST y CIFAR10. Cada barra corresponde a la tasa de aciertos de un modelo, entrenado y evaluado con distintos conjuntos de transformaciones.

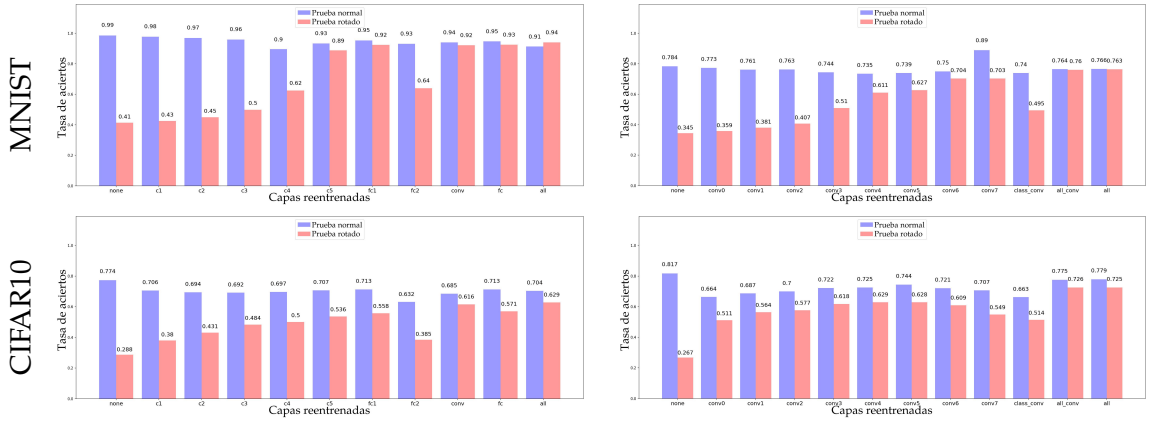


Figura 3.6: Resultado de los experimentos de re-entrenamiento. La figura muestra la tasa de acierto promedio del modelo luego de re-entrenar una capa o conjunto de capas. La tasa de acierto promedio se indica para la versión rotada y la versión original de la BD. El eje x indica que capas re-entrenamos. Las etiquetas *conv* y *allconv* denotan todas las capas convolucionales; la etiqueta *fc* denota todas las capas lineales.

3.5. Re-entrenamiento de modelos para obtener invarianza

Las secciones 3.2 y 3.3 indican que hay poca diferencia en desempeño entre utilizar modelos especializados y modelos con aumentación de datos. No obstante, podría argumentarse que los modelos especializados son más eficientes para entrenarse. Si el tiempo de entrenamiento efectivamente es un factor limitante para el método de aumentación de datos, entonces una alternativa posible sería utilizar una red preentrenada y re-entrenar algunas de sus capas para adquirir invarianza. No obstante, a priori no está claro si toda la red puede o necesita ser re-entrenada, o si solo algunas capas requieren adaptarse a los ejemplos rotados.

Para analizar qué capas son las más indicadas para este fin, entrenamos un modelo base con una BD sin rotar, y luego hacemos copias de la red. En cada copia re-entrenamos algún subconjunto de sus capas con el conjunto de entrenamiento y aumentación de datos. Luego, evaluamos el desempeño de la red con alguna de sus capas re-entrenadas.

Esta estrategia no solo nos permite identificar que capas son mejores para re-entrenar, sino que también pueden ofrecer indicios de cómo se codifica la invarianza en la red. Notamos que este no es un procedimiento de transferencia de aprendizaje, debido a que no cambiamos el dominio ni la BD utilizada para entrenar el modelo original cuando lo re-entrenamos.

La Figura 3.6 muestra las tasas de aciertos obtenidas al re-entrenar algunos subconjuntos de capas. En términos generales, podemos observar que re-entrenar para obtener invarianza muestra una dinámica similar a re-entrenar para hacer transferencia de aprendizaje: las capas finales, con características de alto nivel, son una mejor elección para re-entrenar de forma individual dado que están más cerca de la salida y pueden afectarla más.

Notamos que re-entrenar las ultimas capas restaura el desempeño de las redes casi a niveles originales. Esto indica que hay una redundancia de información en las capas anteriores, debido a que las versiones rotadas de las imágenes pueden codificarse con éxito a partir de la salida de dichas capas. Dado que la red de base fue entrenada con la BD original, esto indica que o bien la red aprende naturalmente filtros equivariantes, o que la equivarianza en los filtros no es tan importante para clasificar objetos rotados.

Por otro lado, re-entrenar cualquier capa aumenta la invarianza, aunque sea levemente. Esto indica que la codificación de la invarianza no tiene por qué codificarse en alguna capa en particular, es decir, puede hacerse en forma global.

No obstante, en el caso de la red SimpleConv es sorprendente que re-entrenar solamente *la última capa* lleva a un desempeño inferior que al re-entrenar otras capas. Como la última capa *fc2* es lineal, esto posiblemente se deba a la acción de la capa lineal anterior *fc1* que colapsa la representación convolucional (posiblemente equivariante) antes que *fc2* la convierta en puntajes de clase. Es decir, la capa *fc1* debe estar perdiendo cierta información relevante a la invarianza.

En el caso de la red AllConvolutional, recordemos que la penúltima capa *class_conv* realiza una simple convolución 1×1 para forzar la cantidad de feature maps a 10. Por algún motivo, en MNIST re-entrenar esta capa no permite recuperar el desempeño original, pero sí en CIFAR10. Tomando este último caso, podemos concluir que al utilizar una sola capa para generar puntajes de clase, la red puede recuperar el desempeño original entrenando sólo esta clase.

3.6. Conclusiones

La invarianza rotacional es una propiedad deseable para varias aplicaciones en el campo de la clasificación de imágenes. La aumentación de datos es una forma simple de entrenar modelos de CNN, que representan el estado del arte en clasificación, para adquirir invarianza. Por otro lado, hay varios modelos CNN con diseños especializados para generar redes invariantes.

En este capítulo, comparamos estas dos estrategias para generar redes invariantes. Observamos que si bien la aumentación de datos puede requerir mayor tiempo de aprendizaje, alcanza desempeños muy similares a otros métodos. Aún más, la red generada es una red estándar, lo cual facilita su utilización en entornos de producción y su optimización.

Además, diseñamos y ejecutamos un experimento para determinar la mejor forma de re-entrenar un modelo no invariante para que adquiera invarianza. Al re-entrenar algunas capas o conjuntos de capas de forma aislada, observamos que se puede aumentar el grado de invarianza re-entrenando cualquier capa. No obstante, encontramos que las capas superiores son más efectivas para recuperar un nivel de desempeño en la BD rotada similar al de la BD original. Estos resultados refuerzan la noción de que las primeras capas convolucionales de una CNN aprenden un conjunto de filtros redundantes y por ende pueden ser reutilizados para otras tareas (aún tareas invariantes a la rotación, como en este caso). Por otro lado, también encontramos casos en donde es posible re-entrenar solo las capas convolucionales y no las lineales de la cabeza de clasificación, y aún así recuperar un nivel de desempeño similar al de una red entrenada para la invarianza.

En base a esto, creemos que se puede aprender más acerca de las CNNs estudiando sus invarianzas y equivarianzas. En este caso, lo hicimos utilizando aumentación de datos, y estrategias de entrenamiento, con la tasa de acierto promedio de la red como métrica. El Capítulo 4 define un conjunto de métricas de equivarianza para evaluar modelos, no solo en base a su tasa de acierto promedio, sino a su representación interna. De esa forma, podemos analizar la invarianza en las CNNs desde otra perspectiva.

Capítulo 4

Métricas de Equivarianza

En este capítulo, presentamos las métricas que forman la contribución principal de esta tesis

Estas métricas permiten evaluar la equivarianza de cualquier modelo basado en redes neuronales con una alta granularidad, es decir, en cada *activación* o valor intermedio de la red.

Presentamos tres tipos de métricas:

- Métricas de invarianza basadas en la varianza.
- Métricas de invarianza basadas en distancia (generalización de las métricas basadas en la varianza).
- Métricas de auto-equivarianza, también basadas en varianza o distancia.

A continuación, desarrollamos los conceptos generales de cualquier métrica que lidie con muestras y transformaciones, y la forma óptima de evaluar el modelo e iterar sobre las activaciones. Luego, introducimos cada una de las métricas, así como versiones específicas para algunos tipos de capas o para hacer un análisis por clases.

4.1. Definiciones generales

Nuestro objetivo es computar una métrica de las activaciones de un modelo f con respecto a un conjunto de transformaciones T de su entrada x. En este caso, estas métricas son de invarianza o equivarianza, pero podrían también corresponder a otras propiedades relacionadas con las transformaciones.

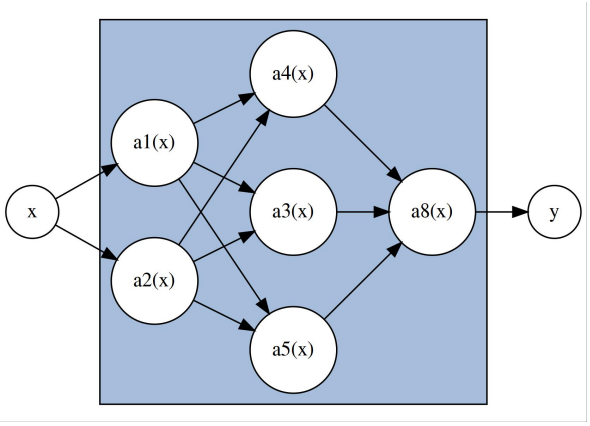


Figura 4.1: Diagrama de una red f con sus activaciones. Dada una entrada x, la red calcula su salida $y = f(x)$, para lo cual requiere calcular las activaciones $a_1(x), \dots, a_8(x)$. El valor final de salida y es simplemente el valor $a_8(x)$. En lugar de considerar cada activación a_i como una función de la salida de las activaciones que están conectadas a ella, vemos la activación como una función de la entrada original x.

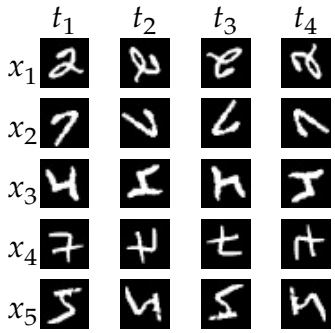
Dada una entrada x, un modelo de redes neuronales computa varios valores intermedios u ocultos, denotados como $a_1(x), \dots, a_k(x)$. Por simplicidad, llamaremos a estos valores $a_i(x)$ *activaciones* de la red f. No debe confundirse este término con *funciones de activación* como *ReLu* o *TanH*. Una activación *puede* ser el resultado de aplicar una función de activación a un vector o tensor, o puede ser simplemente la salida de una capa convolucional o lineal. Notamos que cuando aparece libre, x siempre se refiere a la entrada de toda la red, y no a la entrada de una capa intermedia, como se muestra en la Figura 4.1.

Por ejemplo, sea f una red con una capa convolucional, seguida de una función de activación *ReLU* y luego de una capa lineal final. La salida de la capa convolucional contiene $H \times W \times C$ valores escalares o *activaciones*. Luego de aplicar la función *ReLU*, obtenemos otro conjunto de $H \times W \times C$ *activaciones*. Si aplicamos una función de aplanado para obtener un vector y luego una capa lineal con dimensionalidad D, obtenemos otras D *activaciones*. Por ende, el modelo tiene $k = H*W*C+H*W*C+D$ activaciones.

En situaciones particulares, se pueden ignorar algunas activaciones, por ejemplo si solo nos interesa la salida de una capa convolucional luego de aplicar la función *ReLU*, o si no nos interesa el resultado de la operación de aplanado.

Para medir la equivarianza de un modelo f, medimos la equivarianza de las activaciones individuales $a_1(x) \dots a_k(x)$. Dado que la métrica puede ser definida para una activación de forma independiente del resto, en lo siguiente nos referiremos a una activación simplemente como $a(x)$, sin su índice.

A continuación, definimos la matriz de activaciones Muestra-Transformación (**MT**),



(a) Muestras y transformaciones

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{bmatrix} a(t_1(x_1)) & a(t_2(x_1)) & a(t_3(x_1)) & a(t_4(x_1)) \\ a(t_1(x_2)) & a(t_2(x_2)) & a(t_3(x_2)) & a(t_4(x_2)) \\ a(t_1(x_3)) & a(t_2(x_3)) & a(t_3(x_3)) & a(t_4(x_3)) \\ a(t_1(x_4)) & a(t_2(x_4)) & a(t_3(x_4)) & a(t_4(x_4)) \\ a(t_1(x_5)) & a(t_2(x_5)) & a(t_3(x_5)) & a(t_4(x_5)) \end{bmatrix} \end{array}$$

(b) Matriz **MT**

Figura 4.2: (a) Muestras y sus correspondientes transformaciones $(t_j(x_i))$. (b) Matriz **MT** conteniendo los valores de activación correspondientes a cada entrada para la activación a, para $n = 5$ muestras y $m = 4$ transformaciones.

la cual provee el contexto principal y la notación para la definición de varias de las métricas transformacionales.

4.2. Matriz Muestra-Transformación de Activaciones (MT)

La matriz **MT** de una activación a contiene toda la información necesaria para computar una métrica sobre a. Nos indica como se comporta a para cada combinación posible de muestra y transformación.

Dado un conjunto de n muestras $X = [\, x_1 \cdots x_n \,]$ y un conjunto de m transformaciones $T = [\, t_1 \cdots t_m \,]$ definidas sobre X, podemos calcular todas las posibles transformaciones de todas las muestras, y organizarlas en una matriz.

Dada una activación a, podemos definir la matriz de muestras-transformaciones $\mathbf{MT}(a, X, T)$ de tamaño $n \times m$ como:

$$\mathbf{MT}(a, X, T)_{i,j} = a(t_j(x_i)) \qquad [4.1]$$

Por simplicidad, nos referiremos a $\mathbf{MT}(a, X, T)$ como $\mathbf{MT}(a)$ o simplemente **MT** siempre que el contexto determine claramente a X, T o a.

La Figura 4.2 muestra la correspondencia entre las muestras y sus transformaciones, así como su relación a **MT**. Notamos que **MT** recuerda a la matriz de observaciones utilizada en ANOVA de una vía, donde cada transformación puede ser considerada como un *tratamiento* distinto.

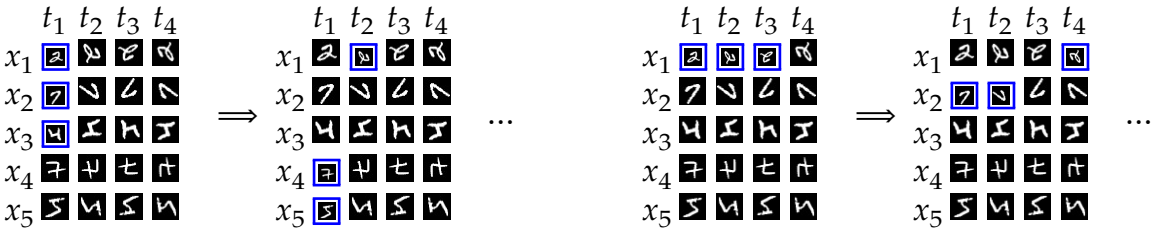
4.2.1. Computación eficiente de MT

El cómputo eficiente de **MT** es crucial para el uso práctico de las métricas. Para obtener las activaciones, solo se requieren evaluaciones de la red, sin cálculo de gradientes. Recordemos que las redes neuronales generalmente se evalúan utilizando lotes de ejemplos, es decir, no permiten evaluar todo un conjunto de datos al mismo tiempo por limitaciones de memoria. Dado un lote de entradas, podemos obtener las k activaciones de la red para dicho lote. No obstante, estas activaciones representan un pequeño subconjunto de los k valores de la matriz **MT** de cada activación a, dado que dichas limitaciones de memoria hacen imposible que se evalúen todas las combinaciones de muestras y transformaciones al mismo tiempo. Por ende, hay una falta de correspondencia entre el orden natural de evaluación de las redes (todas las activaciones para un conjunto pequeño de muestras y transformaciones) y el orden de evaluación de las matrices **MT** (todas las combinaciones de muestra/transformación para una activación particular).

Sería posible computar la matriz **MT** correspondiente a una *sola* activación a calculando la salida de las $n \times m$ combinaciones de muestras transformadas y descartando el valor de todas las activaciones que no son a. El problema con este enfoque es que es prohibitivo computacionalmente. Siendo k el número de activaciones, hay k matrices **MT** correspondientes. Por ende, el proceso se debe repetir k veces. Ahora, k puede ser un número muy grande para redes neuronales modernas. Por ejemplo, ResNet18, la versión más chica de la familia de modelos ResNet (sección 2.4), produce en el orden de 3 millones de activaciones cuando utiliza imágenes con resolución de 224×224 [He+16]. Si asumimos que una red neuronal con k activaciones tiene orden $O(k)$ para ejecutarse, esta estrategia de cómputo de **MT** tiene orden $O(k \times k \times m \times n)$.

Otra posibilidad consiste en generar y almacenar las k matrices **MT** sin repetir evaluaciones de la red. Si bien esto es computacionalmente factible, no lo es en términos de almacenamiento. Si bien m, el número de transformaciones, suele ser chico, el número de muestras n tiende a ser grande para darle relevancia estadística a la métrica. Por ende, almacenar las k matrices **MT** puede ser limitante o aún imposible, ya que requiere $n \times m \times k$ números escalares en punto flotante. Por ejemplo, utilizando el mismo modelo ResNet18 mencionado antes, $n = 1000$ muestras [1], $m = 16$ transformaciones y números flotantes de precisión simple (32 bits) para almacenar las activaciones, se requieren $\frac{k \times m \times n \times 4}{2^{20}} = 178$ GB de almacenamiento para todas las activaciones. Tal cantidad es muy difícil de almacenar en RAM con sistemas usuales. Almacenar las activaciones en disco y luego analizarlas también es posible, pero muy

[1] La 5.4.1 justifica la necesidad de tener esta cantidad de muestras.



(a) Iteración por columnas (varía la fila primero)

(b) Iteración por filas (varía la columna primero)

Figura 4.3: Entradas para la iteración por lotes de la matriz **MT**, utilizando un tamaño de lote 3, $n = 5$ muestras y $m = 4$ transformaciones.

demandante en términos de almacenamiento, en especial al realizar varios experimentos.

Por ende, toda métrica que use matrices **MT** debe implementarse de manera *online* o *móvil*, es decir, calculando la métrica para todas las activaciones al mismo tiempo, a medida que van cambiando las muestras y transformaciones. Esto implica que la matriz **MT** de todas las activaciones deben iterarse al mismo tiempo. Dichas iteraciones puede realizarse por filas (variando primero las transformaciones y luego las muestras) o por columnas (variando primero las muestras y luego las transformaciones), como muestra la Figura 4.3, y siempre por lotes.

Dada la naturaleza de la mayoría de los frameworks y librerías de redes neuronales y cómputo tensorial, no hay una forma directa de iterar sobre las matrices **MT** directamente para implementar una métrica. Además, el cómputo de la evaluación de a lotes implica una complicación extra. Esto dificulta la implementación correcta, rápida y legible de las métricas. Por ello, hemos desarrollado una librería de código abierto [2] que dado un modelo, un conjunto de muestras (una base de datos) y un conjunto de transformaciones, permite iterar sobre la matriz **MT** de forma individual para cada activación y con eficiencia. Esta librería simplifica entonces la implementación de las métricas, e incluye también implementaciones de las métricas propuestas en las secciones 4.3 a 4.6, así como la métrica de Goodfellow [Goo+09] descripta en la sección 2.8.

En las siguientes secciones, desarrollamos varias métricas en base a la matriz **MT**. Primero, definimos tres tipos de métricas de invarianza, cada una basada en distintos conceptos:

- La métrica ANOVA (sección 4.3) utiliza dicho procedimiento de análisis de varianza para determinar si una activación es invariante o no, asumiendo que las transformaciones representan tratamientos en el esquema ANOVA.

[2] Librería de Métricas Transformacionales.

- Las métricas basadas en varianza (sección 4.4) utilizan la varianza muestral o desviación estándar de cada activación para cuantificar su invarianza.
- Las métricas basadas en distancia (sección 4.5) generalizan las métricas basadas en varianza. Para ello, comparan la distancia entre conjuntos de activaciones para cuantificar cuanto cambian ante la transformaciones, y por ende su invarianza.

Finalmente, proponemos también una métrica de auto-equivarianza, para cuantificar este tipo específico de equivarianza(sección 4.6).

4.3. Métrica de invarianza basada en ANOVA

El análisis de la varianza (ANOVA) es un método estadístico para realizar pruebas de hipótesis [Kir12]. Se utiliza para analizar muestras de distintos grupos. ANOVA puede establecer si las medias para distintos grupos de muestras, llamados tratamientos, son las mismas. Si bien ANOVA es un método paramétrico, sus pre-condiciones son laxas y es robusto a violaciones de la normalidad, especialmente con muestras de gran tamaño como aquellas disponibles en las bases de datos de aprendizaje automático [Kir12].

La matriz de observaciones utilizadas en ANOVA de una vía contiene n filas y m columnas; cada fila corresponde a una muestra a la cual se le han aplicado m tratamientos de forma independiente. Dado que la matriz **MT** tiene la misma estructura a la matriz de observaciones de ANOVA de una vía, podemos adaptar la interpretación del método para evaluar la invarianza (Ecuación [2.16]). La hipótesis nula de ANOVA es que todas las medias sean las mismas para distintos grupos. Si los distintos grupos corresponden a diferentes transformaciones, entonces la hipótesis nula es equivalente a la invarianza en esta interpretación estadística. Si la hipótesis nula se rechaza, entonces la activación no es invariante.

Definimos entonces la métrica ANOVA (AM) simplemente como la aplicación del procedimiento ANOVA a la matriz **MT** de cada activación de forma independiente. Por ende, el único parámetro de la métrica es α, el valor de significancia de cada prueba. Como mencionamos en la sección 4.2, el número de activaciones en una red neuronal suele ser enorme. Por ende, debemos también aplicar una corrección de Bonferroni [Kir12] a α para tomar en cuenta la gran cantidad de pruebas de hipótesis correspondientes.

La elección de la invarianza como la hipótesis nula puede resultar extraña, dado que en general es una propiedad que a priori no se asume en los modelos. No obs-

tante, en varios casos los modelos que se vayan a medir habrán sido entrenados o diseñados para ser invariantes. Por ende, si bien el enfoque contrario (que la hipótesis nula se identifique con la no invarianza) tiene sus méritos, usar la invarianza como hipótesis nula puede ser más apropiado en varios casos, y además corresponde mejor con la formulación original del procedimiento ANOVA.

Una limitación importante de la métrica ANOVA asume que para cada transformación t, la distribución de las activaciones para cada una de las muestras transformadas por t es unimodal. Por ende, espera que cada activación responda en una forma similar para distintas muestras, aún cuando estas sean muy distintas, o incluso de distintas clases. Es decir, asume que existe una *activación media* que es la misma para todas las muestras de cada transformación.

4.3.1. Cómputo de la métrica ANOVA

Una prueba ANOVA requiere calcular el valor F de la prueba. El valor F es simplemente la relación entre las diferencias cuadráticas *entre grupos* e *intra grupos*, ajustadas con sus correspondientes grados de libertad. Por ende, el cómputo de la métrica ANOVA requiere dos iteraciones sobre la matriz **MT**, ambas en el orden de columnas primero. La primera iteración calcula las medias para cada tratamiento/transformación. Luego, la segunda computa las diferencias cuadráticas entre grupos e intra grupos, utilizando las medias anteriores. Finalmente, se realiza la división entre ambas cantidades. Por ende, el orden de la métrica es $O(k \times n \times m)$, donde k es la cantidad de activaciones.

4.4. Métricas de Invarianza basadas en la Varianza

Las métricas de invarianza basadas en la varianza calculan la varianza de cada activación. La varianza Var es una función con rango $[0, \infty]$; por ende, una activación es invariante si su varianza es 0. Valores mayores a 0 indican diferentes grados de falta de invarianza. Al medir invarianza en este contexto, podemos tener dos fuentes de variación, una debido a las muestras, y otra debido a las transformaciones. Por ello, desarrollamos dos métricas de invarianzas distintas, Varianza Transformacional y Varianza Muestral. Luego, estas dos métricas se combinan para formar la métrica de invarianza Varianza Normalizada. La siguiente sección describe las tres métricas.

$$\begin{array}{c} \\ \text{(2) Media} \downarrow \end{array} \overset{\rightarrow \text{(1) Var}}{\begin{bmatrix} a(t_1(x_1)) & a(t_2(x_1)) & a(t_3(x_1)) & a(t_4(x_1)) \\ a(t_1(x_2)) & a(t_2(x_2)) & a(t_3(x_2)) & a(t_4(x_2)) \\ a(t_1(x_3)) & a(t_2(x_3)) & a(t_3(x_3)) & a(t_4(x_3)) \\ a(t_1(x_4)) & a(t_2(x_4)) & a(t_3(x_4)) & a(t_4(x_4)) \\ a(t_1(x_5)) & a(t_2(x_5)) & a(t_3(x_5)) & a(t_4(x_5)) \end{bmatrix}} \implies Media \left(\begin{bmatrix} Var([\, a(t_1(x_1))\ a(t_2(x_1))\ a(t_3(x_1))\ a(t_4(x_1)\,]) \\ Var([\, a(t_1(x_2))\ a(t_2(x_2))\ a(t_3(x_2))\ a(t_4(x_2))\,]) \\ Var([\, a(t_1(x_3))\ a(t_2(x_3))\ a(t_3(x_3))\ a(t_4(x_3)\,]) \\ Var([\, a(t_1(x_4))\ a(t_2(x_4))\ a(t_3(x_4))\ a(t_4(x_4)\,]) \\ Var([\, a(t_1(x_5))\ a(t_2(x_5))\ a(t_3(x_5))\ a(t_4(x_5)\,]) \end{bmatrix} \right)$$

Figura 4.4: Cálculo de la métrica Varianza Transformacional para $n = 5$ muestras y $m = 4$ transformaciones. Primero, se calcula la varianza de cada fila (sobre las transformaciones). Luego se calcula la media de la columna resultado (media sobre las muestras).

4.4.1. Métrica de Varianza Transformacional

La **Varianza Transformacional** de una activación a se define como la varianza promedio de las filas de la matriz **MT**. Cada fila i de **MT** contiene las activaciones para la muestra x_i y todas las transformaciones (Ecuación [4.1] y Figura 4.2). Por ende, la varianza se calcula sobre todos los valores de la activación a para todas las transformaciones, y la media sobre estas varianzas para todas las muestras.

Si la activación es completamente invariante a T, entonces la varianza de cada fila sería 0, así como la media sobre todas las filas. La Ecuación [4.2] muestra la definición formal de la Varianza Transformacional de una activación a en términos de su matriz $\mathbf{MT}(a)$.

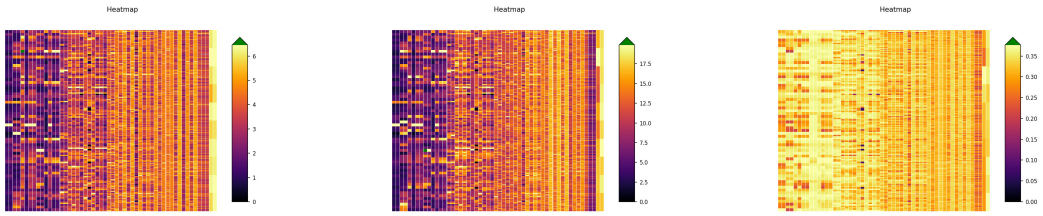
$$VT(a) = Media\left(\begin{bmatrix} Var(\mathbf{MT}(a)[1,:]) \\ \cdots \\ Var(\mathbf{MT}(a)[n,:]) \end{bmatrix}\right) \qquad [4.2]$$

Donde:

- $\mathbf{MT}(a)[i,:] = [\, \mathbf{MT}(a)[i,1] \cdots \mathbf{MT}(a)[i,m] \,]$ es un vector que contiene la fila i de $\mathbf{MT}(a)$.
- $Var([\, x_1 \cdots x_n \,]) = \frac{\sum_{i=1}^{n} x_i - \bar{x}}{n-1}$ es la definición estándar de varianza muestral para un vector de observaciones $[\, x_1 \cdots x_n \,]$.
- $\bar{x} = Media([\, x_1 \cdots x_n \,]) = \frac{\sum_{i=1}^{n} x_i}{n}$ es la definición estándar de la media muestral.

Visualización de la Varianza Transformacional

La Figura 4.5 (a) muestra el resultado de calcular la Varianza Transformacional como un mapa de calor. Cada columna del mapa de calor corresponde a una capa diferente. Dentro de cada capa/columna, cada elemento corresponde al valor de la métrica Varianza Transformacional para diferentes activaciones de esa capa.



(a) (b) (c)

Figura 4.5: (a) Varianza Transformacional, (b) Varianza Muestral y (c) Varianza Normalizada de cada activación de un modelo ResNet. El conjunto de transformaciones para este ejemplo son 16 rotaciones distribuidas uniformemente entre 0° y 360°. Los ejemplos son del conjunto de prueba de CIFAR10. Mejor visto en formato digital.

El color corresponde a diferentes niveles de invarianza . El color verde indica valores fuera de rango. Notamos que, en general, dado que las capas son arbitrarias no hay una estructura de filas en la imagen. Cada capa/columna puede tener una cantidad diferente de activaciones y por ende de filas. Aún más, la distancia vertical entre dos valores de diferentes columnas no conlleva ninguna información. No obstante, las capas correspondientes a funciones de activación, por ejemplo, sí tienen la misma estructura que la capa anterior y por eso se observa una correlación entre ambas. Para las capas cuya salida es un conjunto de feature maps (convoluciones, max-pooling, etc), mostramos solo un valor para cada feature map. Ese valor corresponde a la media de la métrica para un feature map en particular, donde la media se calcula eliminando todas las dimensiones espaciales (ver sección 4.4.4 para más detalles).

Cálculo de la Varianza Transformacional

El cálculo de Varianza Transformacional utiliza el algoritmo de Welford [CGL83] para calcular medias y varianza de forma móvil sin necesidad de almacenar todos los valores previamente. El algoritmo de Welford además tiene mejor estabilidad numérica que otros [CGL83]. De esta forma, el cómputo de Varianza Transformacional requiere una sola iteración sobre las filas de **MT**, y por ende su tiempo de ejecución es del orden $O(k \times n \times m)$ para las k activaciones. En este caso, asumimos también que el orden del tiempo de ejecución de la red es $O(k)$, donde k es el número de activaciones de la misma.

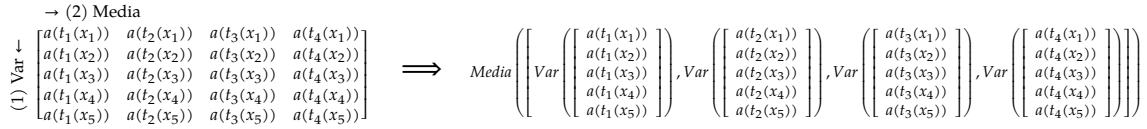


Figura 4.6: Cálculo de la métrica Varianza Muestral para $n = 5$ muestras y $m = 4$ transformaciones. Primero, se calcula la varianza de cada columna (sobre las muestras). Luego la media del vector fila resultado (media sobre las transformaciones)

Valores y unidades de la Varianza Transformacional

La métrica Varianza Transformacional tiene las mismas unidades de la activación a. Cuando $VT(a) = 0$ la activación es invariante a las transformaciones. No obstante, si $VT(a) > 0$ no hay una interpretación clara de la Varianza Transformacional, ya que la unidad de las activaciones depende tanto de las muestras utilizadas como de los parámetros del modelos. Observando la Figura 4.5 (a), estas unidades pueden variar significativamente. Para neutralizar esta fuente indeseable de variabilidad, podemos normalizar los valores de la Varianza Transformacional. Para ello, proponemos utilizar la métrica Varianza Muestral, definida a continuación, para dividir a la Varianza Transformacional y obtener la métrica Varianza Normalizada.

4.4.2. Métrica Varianza Muestral

La **Varianza Muestral** (VM) es la transpuesta conceptual de Varianza Transformacional. En lugar de computar la varianza por filas y luego la media del resultado (un vector columna), la métrica Varianza Muestral primero calcula la varianza por columnas, y luego la media sobre el vector fila resultante (Figura 4.6).

La Ecuación [4.3] presenta la definición formal de la Varianza Transformacional para una activación a en términos de su matriz $\mathbf{MT}(a)$.

$$SV = Media\left(\left[\, Var(\mathbf{MT}[:,1]) \quad \cdots \quad Var(\,\mathbf{MT}(a)[:,m]\,)\,\right]\right) \qquad [4.3]$$

Mientras que la Varianza Transformacional mide la varianza debido a las transformaciones de las muestras, la Varianza Muestral mide la varianza debido a la variabilidad natural de las muestras de la BD o dominio. La Figura 4.5 (b) muestra los resultados de calcular Varianza Muestral como un mapa de calor. Notamos que el orden de magnitud de los valores de la Varianza Muestral es similar a los de la Varianza Transformacional.

Cálculo de la Varianza Muestral

Utilizando también algoritmos de media y varianza móvil, el cómputo de la Varianza Muestral requiere una sola iteración por columnas de **MT**, por ende el tiempo de ejecución es de orden $O(k \times n \times m)$.

4.4.3. Métrica Varianza Normalizada

La métrica **Varianza Normalizada** (VN) es simplemente la relación entre la Varianza Transformacional (Ecuación [4.2]) y la Varianza Muestral (Ecuación [4.3]):

$$VN(a) = \frac{VT(a)}{VM(a)} = \frac{\frac{1}{n}\sum_{i=1}^{n} Var(\mathbf{MT}(a)[i,:])}{\frac{1}{m}\sum_{j=1}^{m} Var(\mathbf{MT}(a)[:,j])} \quad [4.4]$$

La Varianza Normalizada es entonces una relación que balancea la variabilidad debido a las transformaciones con la variabilidad debido a las muestras. Dado que ambas tienen las mismas unidades, el resultado es un valor adimensional. La Figura 4.5 (c) muestra el resultado de la métrica Varianza Normalizada para todas las activaciones.

Casos especiales de la Varianza Normalizada

En los casos donde tanto $VT(a) = 0$ como $VM(a) = 0$, definimos $VN(a) = 1$. Este caso corresponde a la definición de una activación *muerta*, que no responde a ningún patrón y por ende no tiene ningún uso en la red.

En los casos donde $VT(a) > 0$ y $VM(a) = 0$, definimos $VN(a) = +\infty$. Este caso es similar al primero, pero ahora las transformaciones *si* causan variabilidad en la activación. Esto posiblemente se deba a que generan muestras muy por fuera de la distribución original de los datos. Por ejemplo, si todas las imágenes en un conjunto de datos contienen un fondo negro y objectos centrados, como MNIST, entonces es probable que todas las activaciones correspondientes a los bordes de los feature maps, especialmente en las primeras capas, correspondan a activaciones muertas. No obstante, una transformación de escala o traslación puede cambiar tanto la imagen que estas activaciones dejan de valer siempre 0 en los bordes.

Valores de la Varianza Normalizada

Analizando los valores de la Varianza Normalizada, podemos distinguir los siguientes casos:

- Si $VN(a) = 0$, entonces $VT(a) = 0$ y la activación es claramente invariante.
- Si $VN(a) < 1$, la varianza debido a las transformaciones es menor que la varianza debido a las muestras, y por eso podemos considerar aproximadamente invariante a la activación.
- Si $VN(a) > 1$ se aplica el mismo razonamiento pero con la conclusión opuesta.
- Si $VN(a) \simeq 1$, entonces ambas varianzas están en equilibrio, y no hay distinción entre la variabilidad de las muestras y las transformaciones. En este caso, es posible que la base de datos o el dominio naturalmente contenga muestras transformadas, o simplemente que el modelo fue entrenado en forma tal que estos valores son casualmente similares.

Notamos que excepto para el caso $VN(a) = 0$, no hay una interpretación clara de la métrica en términos de valores absolutos. Sólo podemos interpretarla en términos de varianza relativa. Por ejemplo, si $VN(a) = 0.5$, entonces la varianza muestral es el doble que la varianza transformacional.

Sería posible transformar el resultado de la métrica, por ejemplo utilizando la función logística que limite el valor de la misma al intervalo $[0, 1)$. Elegimos deliberadamente no hacerlo de manera de preservar la noción de que no existe una teoría apropiada para interpretar los valores en términos absolutos.

Cálculo de la Varianza Normalizada

El cálculo de VN requiere sólo dos iteraciones sobre la matriz **MT**. La primera iteración se realiza para calcular la Varianza Transformacional, y la segunda para calcular la Varianza Muestral. Luego se realiza la división de las k activaciones. Por ende, su orden es también $O(k \times n \times m)$.

4.4.4. Especialización de las métricas para Feature Maps

Algunos tipos de capas pueden requerir una especialización de las métricas para obtener resultados más útiles y fáciles de interpretar. En esta sección describimos una especialización de las métricas de varianza para activaciones del tipo feature maps

de 2 dimensiones. Este tipo de activaciones suelen se generadas por capas convolucionales $2D$, dado que éstas se utilizan típicamente con imágenes. No obstante, la generalización a feature maps $1D$ o ND es simple.

La salida de una capa Convolucional 2D, o de MaxPooling, o de la función ELU aplicada a feature maps, genera k_f feature maps de tamaño (h, w). El número de activaciones individuales es $k_f \times w \times h$, lo cual puede ser muy grande. Más aún, las activaciones de un feature map tienen una estructura espacial, e ignorarla puede desencadenar en resultados poco interesantes de la métrica. Por ejemplo, para detección de objetos, los bordes de los feature maps generalmente otorgan poca información útil, y analizarlos individualmente no tiene mucha utilidad. Además, los feature maps suelen ser matrices ralas, con lo cual se espera que en promedio la mayoría de sus activaciones estén en 0 [Aim+18].

En tales casos, podemos medir la varianza de cada feature map mediante el agrupamiento de la varianza de las dimensiones espaciales. Dado un feature map F de tamaño $h \times w$ tal que $F(i, j)$ es la activación en la coordenada espacial i, j, podemos definir $VT(F)$ y $VM(F)$ como indica la Ecuación [4.5]:

$$\begin{aligned} VT(F) &= \sum_{i=1}^{h} \sum_{j=1}^{w} VT(F(i,j)) \\ VM(F) &= \sum_{i=1}^{h} \sum_{j=1}^{w} VM(F(i,j)) \\ VN(F) &= \frac{VT(F)}{VM(F)} \end{aligned} \tag{4.5}$$

El agrupamiento se realiza antes de la normalización para quitar la dependencia en el tamaño de la estructura del feature map. Notamos que en el caso de la métrica NV, la agrupación se realiza al nivel de las métricas TV y SV. La definición alternativa que podríamos haber utilizado es NV_{luego} (Ecuación [4.6]), que realiza la agrupación luego de la normalización. Esta definición alternativa es posible pero menos útil, dado que las relaciones individuales $\frac{VT(F(i,j))}{VM(F(i,j))}$ pueden variar demasiado y producir valores sin sentido.

$$NV_{luego}(F) = \sum_{i=1}^{h} \sum_{j=1}^{w} VN(F(i,j)) \tag{4.6}$$

Elegimos agrupar las varianzas de los feature maps via la *suma* de las métricas de cada activación en los feature maps. De esta forma $VT(F)$ y $VM(F)$ representan su varianza total. Como mencionamos anteriormente, los feature maps son ralos

[Aim+18]. Por ende, dado que los filtros suelen estar activos sólo en algunas regiones espaciales, agrupar las activaciones utilizando la función *minimo* en lugar de *suma* subestimaría significativamente la varianza del feature map. La función *maximo* podría causar el problema inverso. La *media* generaría el mismo resultado que la *suma* debido a la normalización (Ecuación [4.5]).

4.5. Métricas basadas en distancias

Las métricas de invarianza basadas en la varianza asumen una distribución unimodal de las activaciones, ya que la varianza cuantifica desviaciones de la media. No obstante, en algunos casos esa asunción puede ser incorrecta.

Las métricas de invarianza basadas en distancia son similares a aquellas basadas en la varianza, pero en lugar de calcular la varianza utilizan una función de distancia entre las activaciones de distintas transformaciones o muestras. Al computar la distancia entre todos los pares de activaciones, no es necesario realizar asunciones de una unimodalidad.

De manera análoga a las métricas basadas en varianza, definimos la métricas de Distancia Transformacional (DT), Distancia Muestral (DM) y Distancia Normalizada (DN) utilizando la distancia en las Ecuaciones [4.7a] a [4.7c]:

$$DT(a) = \text{Media}\left([\ \text{DistanciaMedia}(\mathbf{MT}[1,:]) \cdots \text{DistanciaMedia}(\mathbf{MT}[n,:])\ ]\right) \quad [4.7a]$$

$$DM(a) = \text{Media}\left([\ \text{DistanciaMedia}(\mathbf{MT}[:,1]) \cdots \text{DistanciaMedia}(\mathbf{MT}[:,m])\ ]\right) \quad [4.7b]$$

$$DN(a) = \frac{DT(a)}{DM(a)} \quad [4.7c]$$

En la Ecuación [4.7] DistanciaMedia calcula la distancia media entre todos los valores de un vector de n elementos y $d : R^2 \to R$ representa cualquier función de distancia:

$$\text{DistanciaMedia}([\ x_1 \cdots x_n\ ]) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} d(x_i, x_j)}{n^2} \quad [4.8]$$

Las métricas de distancia calculan la distancia promedio entre activaciones, ya sea por filas(Distancia Transformacional) o por columnas (Distancia Muestral). Si una activación a es completamente invariante a una transformación, entonces la distancia promedio entre los valores de a para todas las transformaciones de una muestra (DistanciaMedia($\mathbf{MT}[i,:]$)) será 0. Por ende, la Distancia Transformacio-

NAL será 0, al igual que la DISTANCIA NORMALIZADA. Si la activación no es invariante, entonces la distancia media entre las activaciones de las muestras transformadas cuantificarán el grado de la invarianza.

4.5.1. Aproximación del cómputo de la distancia media

El cómputo completo de la distancia media para todas las combinaciones de transformaciones y muestras puede ser prohibitivo. Es decir, según utilicemos la DISTANCIA TRANSFORMACIONAL o la DISTANCIA MUESTRAL, debemos calcular esencialmente la matriz de distancia entre los elementos de cada fila o columna de **MT**. Dicha matriz de distancia tiene $O(n^2)$ elementos para las muestras y $O(m^2)$ para las transformaciones. Como se mencionó en la sección 4.2.1, el cálculo de la matriz **MT** debe hacerse de manera móvil, sin almacenar la totalidad de las activaciones. Si bien la cantidad de transformaciones en ocasiones es relativamente menor y por ende m^2 no es un valor muy grande, no es así en todos los casos, y seguramente no lo es en el caso de las muestras (n^2).

Por ende, debemos utilizar una aproximación al cálculo de las distancias promedio. Dado que la iteración sobre la matriz **MT** se hace por lotes, podemos calcular solamente las distancias entre los ejemplos del mismo lote. Por ende, estaremos realizando una aproximación de la media de la matriz de distancia completa utilizando solamente bloques de la misma.

Un enfoque con mejores garantías teóricas involucraría computar una aproximación de bajo rango de la matriz completa de distancias (euclídeas, por ejemplo), y luego calcular las distancias medias [Dok+15]. No obstante, los mejores algoritmos aleatorizados son de orden $O(n + m)$ para una sola matriz [Ind+19], lo cual torna impráctico el cálculo de la métrica para una gran cantidad de activaciones k.

Orden de ejecución de las métricas

De forma análoga al caso de la VARIANZA NORMALIZADA, el cómputo de la DISTANCIA NORMALIZADA requiere dos iteraciones sobre la matriz **MT**, uno para calcular la DISTANCIA TRANSFORMACIONAL y otro para la DISTANCIA MUESTRAL. Dado que en este caso estamos calculando distancias entre todos los elementos de un lote, el orden de la DISTANCIA TRANSFORMACIONAL, la DISTANCIA MUESTRAL y por ende la DISTANCIA NORMALIZADA es de $O(b^2 \times \frac{n \times m}{b} \times k) = O(b \times n \times m \times k)$, donde b es el tamaño de lo lote y k la cantidad de activaciones.

El cálculo de las métricas de distancia tiene un costo extra b de acuerdo al tama-

ño del lote. Cuanto más grande sea b, mejor será la aproximación pero mayor será el tiempo de cómputo. No obstante, el tamaño de lote b tiene un límite duro en la cantidad de memoria RAM requerida para almacenar las activaciones de todos los ejemplos del lote, ya sea en la CPU o GPU.

Una alternativa para mejorar la aproximación consiste en realizar varias iteraciones de la matriz **MT**, pero cambiando el orden de iteración sobre los ejemplos. De esta forma, y eligiendo con cuidado el orden en que se integra sobre las muestras y transformaciones, estamos efectivamente tomando muestras de distintos bloques de la matriz de distancias y por ende mejorando la aproximación de forma arbitraria. Además, esta estrategia permite elegir de forma independiente el tamaño de lote para optimizar el cómputo de la evaluación de la red.

4.5.2. Relación entre las métricas basadas en varianza y las basadas en la distancia euclidiana

En el caso de las métricas de distancia que utilizan a la distancia euclidiana al cuadrado, las métricas de varianza y distancia son equivalentes, de acuerdo a la Ecuación [4.9]. En este caso particular, podemos entonces evitar la aproximación inducida por el muestreo ralo de la matriz de distancia con sólo computar la métrica Varianza Normalizada original. Esto indica entonces que las métricas basadas en varianza son un caso particular de las métricas basadas en distancia, en donde la aproximación no es necesaria ya que hay el algoritmo de Welford permite calcularla en forma móvil.

$$
\begin{aligned}
\text{DistanciaMedia}([\, x_1 \cdots x_n \,]) &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n^2} \\
&= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} x_i^2 - 2x_i x_j + x_j^2}{n^2} \\
&= \frac{\sum_{i=1}^{n}(n x_i^2 - 2x_i \sum_{j=1}^{n} x_j + \sum_{j=1}^{n} x_j^2)}{n^2} \\
&= \frac{n\sum_{i=1}^{n} x_i^2 - 2\sum_{j=1}^{n} x_j \sum_{i=1}^{n} x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} x_j^2)}{n^2} \qquad [4.9] \\
&= \frac{n^2 E(x^2) - 2n^2 E(x)E(x) + n^2 E(x^2)}{n^2} \\
&= 2(E(x^2) - E(x)^2) \\
&= 2\,Var([\, x_1 \cdots x_n \,])
\end{aligned}
$$

4.5.3. Especialización para los feature maps

Como mencionamos anteriormente, medir la invarianza de cada activación de un feature map puede ser indeseable, ya que el feature map tiene una estructura espacial que debe ser tomada en cuenta. El método descripto para la métrica Varianza Normalizada para calcular la varianza de feature maps (sección 4.4.4) también es válido para las métricas de distancia (Ecuación [4.10]). Es decir, podemos calcular la métrica de distancia de cada activación individual $DN(F(i,j))$ en un feature map F y luego sumar los valores de las métricas para cada activación.

$$\begin{aligned} DT(F) &= \sum_{i=1}^{h}\sum_{j=1}^{w} DT(F(i,j)) \\ DM(F) &= \sum_{i=1}^{h}\sum_{j=1}^{w} DM(F(i,j)) \\ DN(F) &= \frac{DT(F)}{DM(F)} \end{aligned} \qquad [4.10]$$

No obstante, la ventaja de utilizar una métrica basada en distancia es que podemos emplear distancias especializadas para cada tipo de activación o capa. Por ejemplo, podemos comparar los feature maps con una métrica de distancia especial y obtener entonces distancias entre *pares de feature maps* en lugar de *pares de activaciones individuales*. En el caso de los feature maps, podemos emplear para ello cualquier métrica de distancia para imágenes.

Sea F un feature map, y $\mathbf{MT}'(F)$ una matriz de tamaño $n \times m$ como antes, pero ahora cada elemento $\mathbf{MT}'(F)[i,j] \in R^{h \times w}$ consiste en el feature map obtenido al evaluar la muestra i luego de aplicarle la transformación j, es decir, $t_j(x_i)$. Entonces, podemos definir DT, DM y DN de forma similar a la Ecuación [4.7], pero para feature maps F (Ecuación [4.11]).

$$\begin{aligned} DT(F) &= \text{Media}\left([\ \text{DistanciaMedia}(\mathbf{MT}'(F)[1,:]) \cdots \text{DistanciaMedia}(\mathbf{MT}'(F)[n,:])\ ]\right) \\ DM(F) &= \text{Media}\left([\ \text{DistanciaMedia}(\mathbf{MT}'(F)[:,1]) \cdots \text{DistanciaMedia}(\mathbf{MT}'(F)[:,m])\ ]\right) \\ DN(F) &= \frac{DT(a)}{DM(a)} \end{aligned} \qquad [4.11]$$

Donde ahora d es una función de distancia entre feature maps en lugar de números reales, o sea $d : R^{h \times w} \times R^{h \times w} \to R$, y la distancia media se define como:

$$\text{DistanciaMedia}([\ F_1\ \cdots\ F_n\ ]) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} d(F_i, F_j)}{n^2} \qquad [4.12]$$

4.6. Métrica Auto-Equivarianza

En esta sección proponemos un método para calcular la auto-equivarianza (sección 2.6) aproximada de las activaciones. La auto-equivarianza requiere que tanto la entrada como la característica sean parte del dominio de t, y por ende que compartan una estructura. Por ejemplo, si la entrada a la red x son imágenes, entonces la transformación t estará definida para imágenes. Para poder calcular la auto-equivarianza, necesitamos que $f(x)$ también sea una imagen.

Por ende, y de forma similar a la sección 4.5.3, definiremos esta métrica en términos de un conjunto de activaciones $A = [\ a_1\ \cdots\ a_p\ ]$. Notamos que A es un conjunto con estructura; por ejemplo, podría ser un tensor $3D$ para imágenes RGB. El conjunto de activaciones A debe tener la misma estructura que x, es decir, t debe poder actuar sobre A.

Para definir la métrica de auto-equivariancia, recordemos su definición (Ecuación [2.19]): si A es auto-equivariante a t_i, entonces $A(t_i(x)) = t_i(A(x))$.

Las métricas de invarianza están basadas en comparar los valores de $a(t_j(x_i))$ para distintas transformaciones t_j y muestras x_i. Adaptaremos esta idea para el caso de la auto-equivarianza.

Si asumimos que $T = t_1, \ldots, t_m$ es un conjunto de transformaciones invertibles, entonces podemos calcular los valores $t_1^{-1}(A(t_1(x))), \ldots, t_m^{-1}(A(t_m(x)))$. Si A es auto-equivariante, estos valores deberían ser todos iguales ya que $t_i^{-1}(A(t_i(x))) = t_i^{-1}(t_i(A(x))) = A(x)$. Comparando estos valores podemos entonces determinar el grado de auto-equivarianza de las activaciones.

Definimos entonces la métrica Auto-Equivarianza Transformacional de Distancia (AETD) (Ecuación [4.13]) de un conjunto de activaciones A en una forma similar a la métrica Distancia Transformacional de la sección 4.5:

$$\begin{aligned} AETD(A) &= \text{Media}\ ([\ \text{DistanciaMedia}(\mathbf{MT}'(A)[1,:])\ \cdots\ \text{DistanciaMedia}(\mathbf{MT}'(A)[n,:])\ ]) \\ \mathbf{MT}'(A)[i,j] &= t_j^{-1} A(t_j(x_i)) \end{aligned} \qquad [4.13]$$

La Ecuación [4.13] utiliza una versión modificada de la matriz **MT** denotada $\mathbf{MT}'$, donde cada elemento es un conjunto de activaciones A transformadas por la inversa

de la transformación original para poder medir la auto-equivarianza.

De forma análoga al caso de la invarianza, podemos definir las métricas de Auto-Equivarianza Muestral de Distancia y Auto-Equivarianza Normalizada de Distancia.

Por supuesto, en el caso en que la función de distancia sea la euclidiana al cuadrado, la métrica Auto-Equivarianza Transformacional de Distancia se puede calcular utilizando la varianza, de forma análoga al caso de la Varianza Normalizada (sección 4.5.2). De este modo, podemos definir las métricas Auto-Equivarianza Transformacional de Varianza, Auto-Equivarianza Muestral de Varianza y Auto-Equivarianza Normalizada de Varianza, análogas a las recién definidas.

Con el objetivo de ser breve, sólo definimos las métrica basadas en distancias, sin detallar este caso particular que es más eficiente para calcular con varianzas.

4.6.1. Normalización

Para las métrica de auto-equivarianza también es posible no normalizar con la auto-equivarianza muestral. Dependiendo de la tarea, generalmente se espera que los modelos sean invariantes a las variaciones muestrales; de hecho, este es uno de los principios básicos de la generalización [Xie+17]. No obstante, la auto-equivarianza *muestral* no es (a priori) un sesgo natural de los problemas de clasificación, y por ende los valores obtenidos serían generalmente mucho más grandes que los de la auto-equivarianza *transformacional*. Por ende, para obtener valores normalizados, podemos directamente normalizar el conjunto de activaciones A. Para ello, dado que A es un tensor de algún tipo de forma, lo normalizamos dividiendo por la norma inducida por la medida de distancia para otorgarle norma unitaria. Luego computamos la métrica Auto-Equivarianza con estas activaciones de norma unitaria, sin otro esquema de normalización.

4.6.2. Valores de la Auto-Equivarianza

La métrica Auto-Equivarianza mide la distancia promedio entre los valores $t_1^{-1}(A(t_1(x)))$, $\dots$, $t_m^{-1}(A(t_m(x)))$. Cuando estos son todos iguales, la distancia promedio es 0 y por ende A es auto-equivariante. Al igual que para las métricas de invarianza, los valores de la métrica Auto-Equivarianza valores mayores a 0 indican desviaciones de la auto-equivarianza. Debido al esquema de normalización, los valores ya no pueden interpretarse en forma relativa entre la auto-equivarianza transformacional y la muestral.

4.6.3. Limitaciones para calcular la Auto-Equivarianza

Notamos nuevamente que este método está limitado a conjuntos de activaciones con la misma estructura que x. Por ejemplo, en el caso de las CNNs utilizadas para análisis de margen, este método sólo puede ser aplicado a los feature maps. La auto-equivarianza de la salida de las capas lineales no puede ser medida, dado que la mayoría de las transformaciones para imágenes $2D$ no está bien definida en vectores $1D$. No obstante, la mayoría de las arquitecturas modernas de redes neuronales para imágenes utiliza mayormente capas convolucionales, y sólo emplea una o dos capas lineales a lo sumo para producir los puntajes finales de clase. Por ende, esto representa una limitación menor.

Por otro lado, la limitación de que las transformaciones deban ser invertibles no es tan importante en el caso de las CNNs para imágenes, debido a que las transformaciones afines son invertibles, así como las transformaciones de color y muchas otras transformaciones relacionadas.

4.6.4. Formulación alternativa: Auto-Equivarianza de Distancia Simple

Otra alternativa para calcular el grado de auto equivarianza de un conjunto de activaciones A a una sola transformación t es medir el valor $||A(t(x)) - t(A(x))||$ sobre todas las muestras, donde $||\cdot||$ es la norma euclidiana u otra. Si f es auto-equivariante, entonces $A(t(x)) = t(A(x))$ y por ende $||A(t(x)) - t(A(x))|| = 0$. En el caso en que $T = [\, t_1 \cdots t_m \,]$, para calcular la auto-equivarianza para el conjunto T, podemos simplemente calcular la auto-equivarianza de cada transformación y luego calcular el promedio sobre el conjunto de transformaciones. Llamamos a esta métrica Auto-Equivarianza de Distancia Simple.

Una ventaja de esta formulación es que no requiere que las transformaciones t_j sean invertibles. Una limitación es que no puede caracterizar las interacciones entre las representaciones de distintas transformaciones. Además, no es análogo al esquema de la invarianza, lo cual dificulta la comparación entre las dos métricas. Por ende, en la sección experimental hemos elegido utilizar la primera formulación de la métrica.

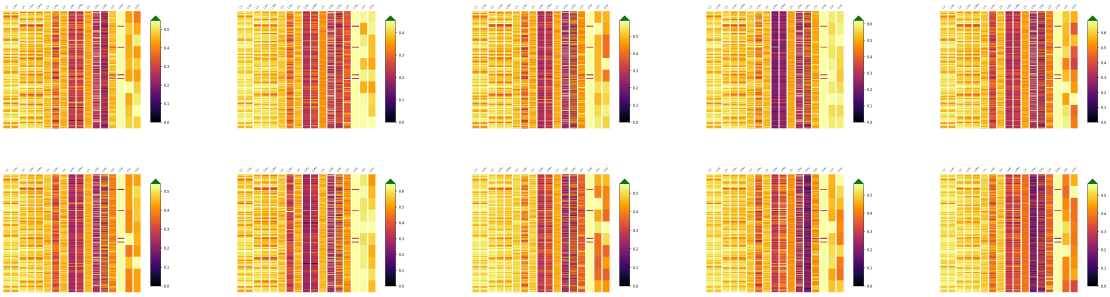


Figura 4.7: Mapas de calor de la métrica Varianza Normalizada para cada una de las 10 clases de CIFAR10. Si bien la distribución general de invarianza es la misma para todas las clases, cada una tiene características particulares.

4.7. Métricas Estratificadas

Para problemas de clasificación con un conjunto C de clases, la codificación de la entrada de las activaciones puede variar drásticamente dependiendo de la clase de las muestras. De la misma forma, la estructura de la invarianza o equivarianza podría ser específica a cada clase. Por ende, puede ser de utilidad medir la invarianza para cada clase por separado. Definimos entonces $M(a, X_c, T)$ como la métrica por clase de una activación a, donde M es una métrica arbitraria, y X_c este conjunto de muestras que pertenecen a la clase c. La Figura 4.7 muestra los mapas de calor de cada clase.

La comparación entre $M(a, X_c, T)$ y $M(a, X, T)$ puede ayudar a determinar correspondencia entre la equivarianza de una activación y los ejemplos de una clase. Además, para evaluar si existe un componente de clase específico de la codificación de la equivarianza en la red, podemos definir una métrica *estratificada* (Ecuación [4.14]):

$$M_{estratificada}(a, X, T) = Media(\begin{bmatrix} M(a, X_1, T) & \dots & M(a, X_C, T) \end{bmatrix}) \qquad [4.14]$$

La métrica estratificada es simplemente la media sobre el conjunto de las métricas específica para cada clase $M(a, X_c, T)$. La diferencia entre $M(a, X, T)$ y $M_{estratificada}(a, X, T)$ nos permite cuantificar el impacto de la clase en la métrica.

4.8. Conclusiones

En esta sección, presentamos varias métricas que permiten cuantificar la invarianza y auto-equivarianza. Las métricas tienen una granularidad alta, permitiendo calcularlas para cada valor intermedio o *activación* de la red.

Las métricas de invarianza basadas en varianza Varianza Transformacional y

Varianza Muestral se combinan para formar la Varianza Normalizada. Estas métricas utilizan la varianza para caracterizar la invarianza: si Varianza Normalizada es 0 para una activación, entonces la misma es invariante. Las métricas de invarianza basadas en distancia Distancia Transformacional, Distancia Muestral y Distancia Normalizada son análogas a las basadas en varianza, pero las generalizan. Para ello, utilizan una función de distancia entre activaciones de distintas transformaciones de las muestras para cuantificar la invarianza.

La métrica de auto-equivarianza Auto-Equivarianza utiliza también un esquema de distancia similar al de la Distancia Normalizada. No obstante, en lugar de comparar las activaciones de las distintas transformaciones directamente, aplica la inversa de la transformación original para deshacer la transformación de las activaciones. Luego toma las activaciones des-transformadas y computa la distancia entre estas activaciones.

También introdujimos variantes de las métricas para tipos de activación específicos como los feature maps, y para analizar el componente de clase de la equivarianza para problemas de clasificación.

El cómputo de las métricas utiliza la matriz de Muestras-Transformaciones **MT**. Dicha matriz contiene las activaciones para todas las combinaciones de muestras y transformaciones. Además, presentamos un esquema de generación de la matriz **MT** que permite el cómputo eficiente de las métricas.

Las métricas tienen una complejidad computacional de $O(k \times n \times m)$, lo cual es lineal en las activaciones, métricas y transformaciones, respectivamente. En el caso de las métricas de distancia, para invarianza o auto-equivarianza, dicha complejidad aumenta a $O(b^2 \times k \times \frac{n \times m}{b}) = O(b \times k \times n \times m)$, donde b es el tamaño de lote. El término cuadrático b^2 es inducido por el calculo de la matriz de distancias entre pares. En tales casos, el cálculo de distancia es aproximado, y el tamaño de lote permite ajustar el grado de aproximación, balanceando eficiencia computacional y calidad de la aproximación.

Estas métricas eficientes permiten el análisis de modelos modernos de redes neuronales en términos de la equivarianza. Se distinguen de otros esfuerzos similares [Goo+09; LV15] en que son fáciles de interpretar y eficientes para calcular, especialmente en el contexto de redes grandes con millones de activaciones.

El apéndice B describe la implementación de la librería *Transformational Measures* donde se implementan estas métricas.

A continuación, en el Capítulo 5, realizamos varios experimentos y visualizaciones para comprender la naturaleza de las métricas presentadas, y validar su utilidad. Luego las utilizamos para evaluar modelos de CNN y características de los mismos.

Capítulo 5

Evaluación de Métricas de Equivarianza

En este capítulo, realizamos varios experimentos para (1) validar las métricas desarrolladas en el Capítulo 4, (2) estudiar su comportamiento y (3) analizar modelos existentes de redes neuronales en términos de sus invarianzas y equivarianzas.[1].

Para validar las métricas, realizamos experimentos para determinar si las mismas son capaces de medir la propiedades deseadas. Luego, estudiamos el comportamiento general de las métricas en términos de diferentes factores: inicialización aleatoria de los pesos de la red, tamaño de la BD, subconjunto de la BD (entrenamiento o prueba), evaluación estratificada, y especializaciones para feature maps. Finalmente, medimos equivarianzas en varios modelos populares de CNN, y para el modelo , cuantificamos el impacto de las capas de Batch Normalization, MaxPooling, distintas funciones de activación y el tamaño del kernel de las convoluciones.

Si bien las métricas pueden utilizarse para analizar cualquier tipo de modelo de redes neuronales, nos enfocamos en problemas de clasificación de imágenes para caracterizar las métricas y a los modelos.

5.1. Metodología

La metodología general de los experimentos consiste en entrenar un modelo con una BD y un conjunto de transformaciones determinadas para aumentación de datos. La aumentación de datos para el entrenamiento consiste en aplicar a cada ejemplo una transformación elegida de forma aleatoria del conjunto de transformaciones.

[1]El código para replicar estos experimentos está disponible en https://github.com/facundoq/transformational_measures_experiments

Luego, en la mayoría de los casos evaluamos una métrica utilizando el modelo entrenado y el *mismo* conjunto de transformaciones que se utilizó para la aumentación de datos. En otras ocasiones, que aclararemos oportunamente, este esquema varía.

A continuación, describimos las bases de datos, transformaciones y modelos utilizados en los experimentos.

5.2. Métricas

Para limitar la exposición hemos decidido sólo utilizar algunas métricas en los experimentos. Dada su mayor eficiencia y, en general utilizaremos métricas basadas en varianza en lugar de las basadas en distancia. No obstante, realizamos una comparación entre ambas para determinar sus diferencias y, sobre todo, la calidad de la aproximación de las métricas de distancia. Por ende, la mayoría de los experimentos utilizan solamente las métricas Varianza Normalizada y Auto-Equivarianza Normalizada de Varianza.

5.2.1. Bases de datos

Todos nuestros experimentos utilizan las BDs MNIST y CIFAR10. Ambas BDs son reconocidas y cualquier análisis que se realice con ellas será fácil de interpretar y relacionar con los modelos actuales. Además, si bien el cálculo de las métricas es eficiente y permite utilizar BDs grandes, al analizar las métricas realizamos una variedad de experimentos y por ende el costo de cómputo agregado es considerable. Por ende, el hecho de que ambas BDs sean relativamente pequeñas (60000 muestras) permite realizar dichos experimentos.

Si bien MNIST es una BD simple, provee resultados fácilmente interpretables. Como todos los modelos obtienen una tasa de aciertos cercana al 100 % en esta BD, nos permite analizar un modelo en el régimen de clasificación perfecta. CIFAR10, por otro lado, contiene imágenes naturales más complejas y los modelos no alcanzan un desempeño perfecto, lo cual permite complementar el análisis.

5.2.2. Transformaciones

Para simplificar la interpretación de los experimentos, definimos cuatro conjuntos de transformaciones comunes. Éstos conjuntos representan transformaciones afines, y por ende proveen un rango de diversidad de modo de establecer las propie-

dades de las métricas de manera independiente a las transformaciones específicas utilizadas.

1. **Rotación** (16 transformaciones). Rotaciones desde 0° a 360°, discretizadas en 16 ángulos. Las rotaciones se realizan siempre utilizando el centro de la imagen como origen.

2. **Escalado** (11 transformaciones). Escalamos las imágenes utilizando los coeficientes de escala: 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.05, 1.10, 1.15, 1.20, 1.25. Estos coeficientes son tanto para el alto como para el ancho, por lo cual se mantiene fija la relación de aspecto de las imágenes originales. La distribución de estos coeficientes no es uniforme, ya que tenemos un rango mayor para achicar la imagen (0.5) que para agrandarla (1.25). Determinamos estos límites visualmente, verificando que un escalado más intenso causa que los objetos ya no sean reconocibles en ambas BDs. Sólo escalamos el contenido del imagen, manteniendo su tamaño constante (28×28px y 32×32px para MNIST y CIFAR10, respectivamente). Al achicar las imágenes, rellenamos los bordes con píxeles negros.

3. **Traslaciones** (24 transformaciones): Generamos todas las traslaciones de $d = 2^i$ píxeles con el siguiente esquema: $(-d,-d)$, $(-d,d)$, $(d,-d)$, (d,d), $(0,d)$, $(d,0)$, $(0,-d)$, $(-d,0)$. Los valores de i utilizados son $0,1,2$, para un total de $8*3 = 24$ transformaciones. En el caso de MNIST o CIFAR10, estas traslaciones representan desplazamientos de hasta 15 % de la imagen en cada dirección.

4. **Combinadas** (144 transformaciones): Generar todas las posibles combinaciones de las transformaciones anteriores resultaría en un conjunto de 4224 transformaciones. Tal cantidad de transformaciones es prohibitiva computacionalmente para la cantidad de experimentos que debemos realizar. Por ende, generamos una combinación rala de las transformaciones anteriores, utilizando 6 rotaciones, distribuidas uniformemente en los 360°, 3 escalados $(0.5, 1, 1.25)$ y las 8 traslaciones posibles de $d = 2^3 = 8$ pixeles con el esquema anterior, para un total de $6 \times 3 \times 8 = 144$ transformaciones.

Nos referiremos a estos conjuntos simplemente como las rotaciones, escalados, traslaciones y combinadas. La Figura 5.1 muestra ejemplos de cada uno estos conjuntos para las BDs MNIST y CIFAR10.

Dado que el conjunto de transformaciones **combinadas** es relativamente grande y por ende computacionalmente pesado, sólo lo utilizamos en algunos experimentos.

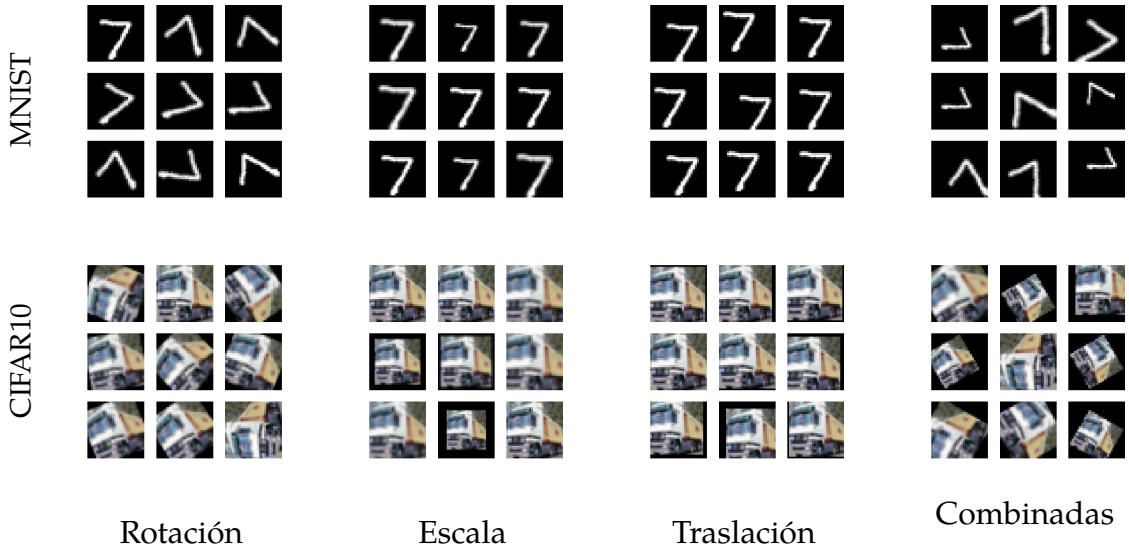


Figura 5.1: Muestras de MNIST y CIFAR10 para cada uno de los conjuntos de transformaciones.

5.2.3. Modelos

Para la mayoría de los experimentos, utilizamos el modelo (Figura 5.2), detallado también en la sección 2.4. Este modelo consiste solamente de capas tradicionales de Convolución, MaxPooling y Lineales. Sólo utiliza funciones de activación *ELU*, excepto por la capa final $Softmax$. Todas las convoluciones utilizan un espaciado de $1px$ y un tamaño de kernel de 3×3. Las capas MaxPooling son 2D, con $paso = 2$ y tamaño de kernel de $k = 2 \times 2$. fue el modelo más simple con sólo esos tres tipos de capa que tiene una tasa de acierto del 80 % en CIFAR10, un desempeño similar a otros modelos que evaluaremos (sección 5.5.5).

Dado que nuestro objetivo no es obtener tasas de acierto del estado del arte, para priorizar la consistencia y simplicidad utilizamos el optimizador AdamW [LH19] con una tasa de aprendizaje de $1-e4$ para entrenar todos los modelos. La cantidad de épocas utilizadas para el entrenamiento de cada modelo fue determinado de forma específica para cada BD de forma de asegurar que el modelo converge. Para escalar la cantidad de épocas en base al tamaño del conjunto de transformaciones, se multiplicó el número de épocas base para cada modelo/base de datos por $log(m)$, donde m es el tamaño del conjunto de transformaciones.

Dado que CIFAR10 es un conjunto de datos más desafiante que MNIST, los modelos para esta última BD han sido modificados de forma tal de utilizar la mitad de filtros o características en todas las capas. Dado que la invarianza no puede separarse completamente de la tasa de aciertos, verificamos que en todos los casos la tasa de acierto de los modelos sea superior al 95 % en MNIST y al 75 % en CIFAR10

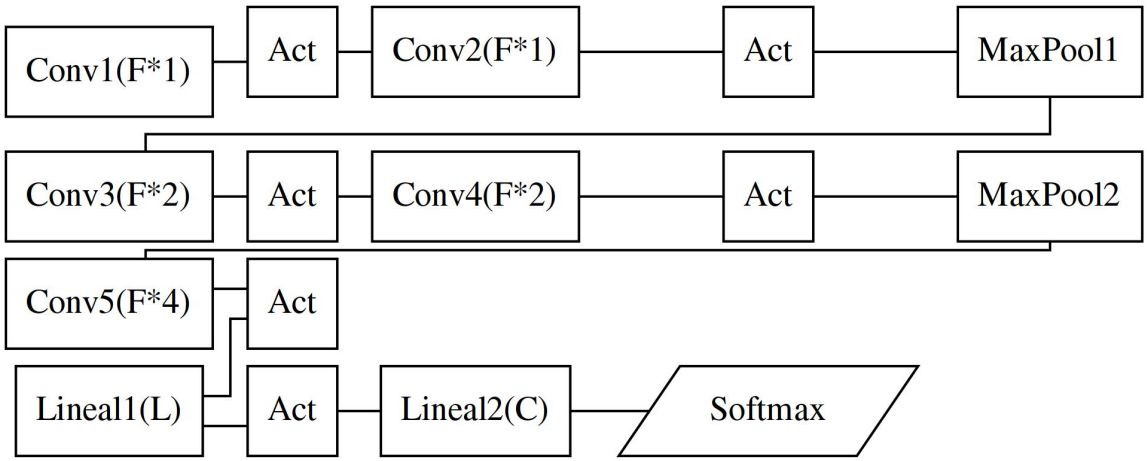


Figura 5.2: Arquitectura del modelo . El modelo es una red convolucional típica con capas Convolucionales, de MaxPooling y Lineales.

(Figura 5.3).

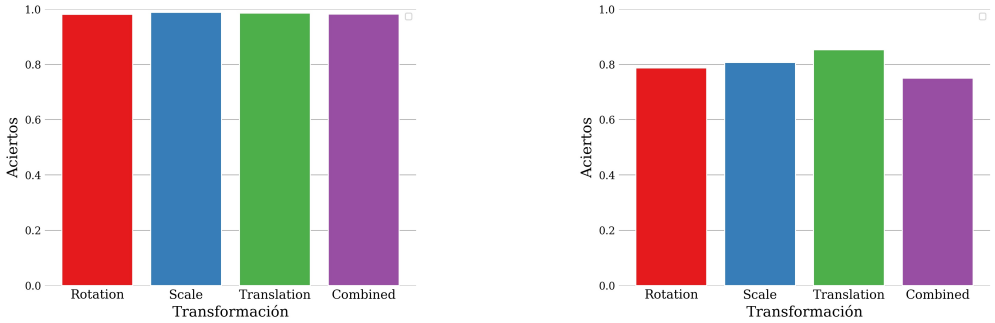


Figura 5.3: Tasas de acierto del modelo en los 4 conjuntos de transformaciones utilizando MNIST y CIFAR10.

A continuación presentamos los experimentos realizados, comenzando con los de validación de las métricas, siguiendo luego con el análisis de las mismas, y finalizamos con la evaluación de distintos modelos de CNN y sus características.

5.3. Validación de las métricas

Para comprender mejor la información provista por las métricas presentadas, las comparamos y analizamos de acuerdo a su tipo. Para simplificar la comparación, presentamos los resultados de las métricas agrupados por capas. Para ello, calculamos el valor medio de la métrica para todas las activaciones de la misma capa, y graficamos el valor promedio de cada capa.

Finalmente, evaluamos las métricas en modelos entrenados con y sin aumentación de datos. Los modelos entrenados sin aumentación de datos tienen baja tasa de acierto cuando se evalúan con muestras transformadas (Capítulo 3). Por otro lado, los modelos entrenados con aumentación de datos tienen un desempeño similar al ser evaluados con muestras transformadas. Por ende, esperamos que estos últimos modelos posean más invarianza, al menos en la capa final de clasificación. De esta forma, podemos determinar si las métricas son representativas de la invarianza y auto-equivarianza.

5.3.1. Métricas Varianza Transformacional y Varianza Muestral

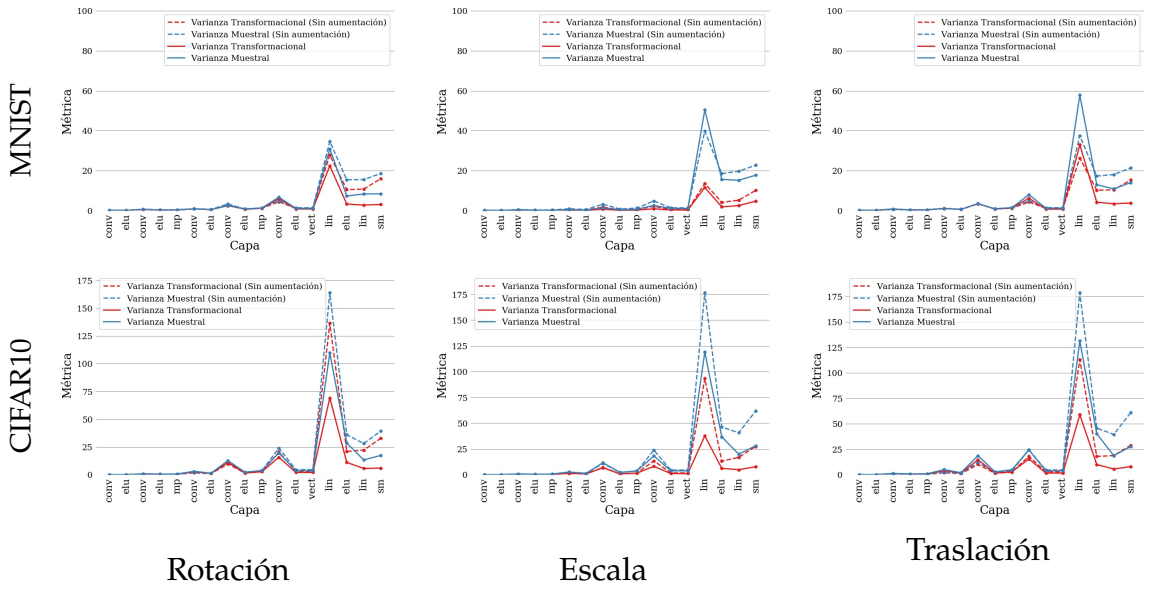


Figura 5.4: Comparación de las métricas Varianza Transformacional y Varianza Muestral para varios conjuntos de transformaciones y BDs, y el modelo SimpleConv. Las líneas punteadas indican modelos entrenados sin aumentación de datos.

La Figura 5.4 muestra la distribución de la invarianza en las distintas capas para las métricas Varianza Transformacional y Varianza Muestral (numerador y denominador, respectivamente, de la métrica Varianza Normalizada).

Primero, notamos que la magnitud de ambas métricas se encuentra en el mismo rango si la BD y el conjunto de transformaciones es el mismo. No obstante, varía significativamente para distintas transformaciones y bases de datos. Notamos también que las magnitudes varían significativamente entre capas. La invarianza de las capas convolucionales es significativamente menor que la de las capas lineales. Esto se debe a que las unidades de las activaciones de las capas convolucionales son en general menores que las de las lineales. La varianza para los modelos en MNIST es también mucho menor que la de los mismos modelos en CIFAR10. En el caso particular de

MNIST, la Varianza Normalizada para las transformaciones de rotación es significativamente menor que la de otras transformaciones. Esto confirma los argumentos presentados en el sección 2.8 respecto a la importancia de normalizar la Varianza Transformacional para obtener valores que sean interpretables entre capas, BDs y transformaciones.

Segundo, los modelos entrenados con aumentación de datos tienen menos activaciones variantes, como se esperaba. Esto indica que las métricas están de hecho reflejando la invarianza del modelo. Es interesante notar, no obstante, que en algunas capas los modelos entrenados sin aumentación de datos tienen valores menores de Varianza Transformacional o Varianza Muestral, excepto por las capas finales. Si bien esto es posible en principio debido a distintos esquemas de codificación entre los modelos, notamos que cuando esto ocurre afecta tanto a Varianza Transformacional como a Varianza Muestral, lo cual es otra razón para buscar la normalización de la métrica. Las capas finales son siempre más invariantes (valores más chicos de Varianza Transformacional y Varianza Muestral) en modelos entrenados con aumentación de datos, lo cual es esperable debido a que la optimización de la función de error con datos aumentados está especificando estrictamente que la última capa, al menos, sea invariante.

El hecho de que la Varianza Transformacional de cada capa sea casi siempre menor que la Varianza Muestral indica también que la invarianza es una propiedad global, es decir, codificada en toda la red y no en alguna capa en especial.

5.3.2. Métricas Distancia Transformacional y Distancia Muestral

La Figura 5.5 muestra la distribución de las métricas Distancia Transformacional y Distancia Muestral (numerador y denominador, respectivamente, de la métrica Distancia Transformacional) para MNIST y CIFAR10. Dichas métricas fueron calculadas con la distancia euclídea al cuadrado, y utilizando distancias entre todas las activaciones sin ninguna especialización para los feature maps. Notamos aquí nuevamente la necesidad de normalizar las métricas transformacionales para obtener resultados interpretables, dadas las distintas magnitudes.

Por otro lado, los valores de la Distancia Transformacional y Distancia Muestral son similares en estructura a los de la Varianza Transformacional y Varianza Muestral (Figura 5.4), lo cual indica que la aproximación es útil.

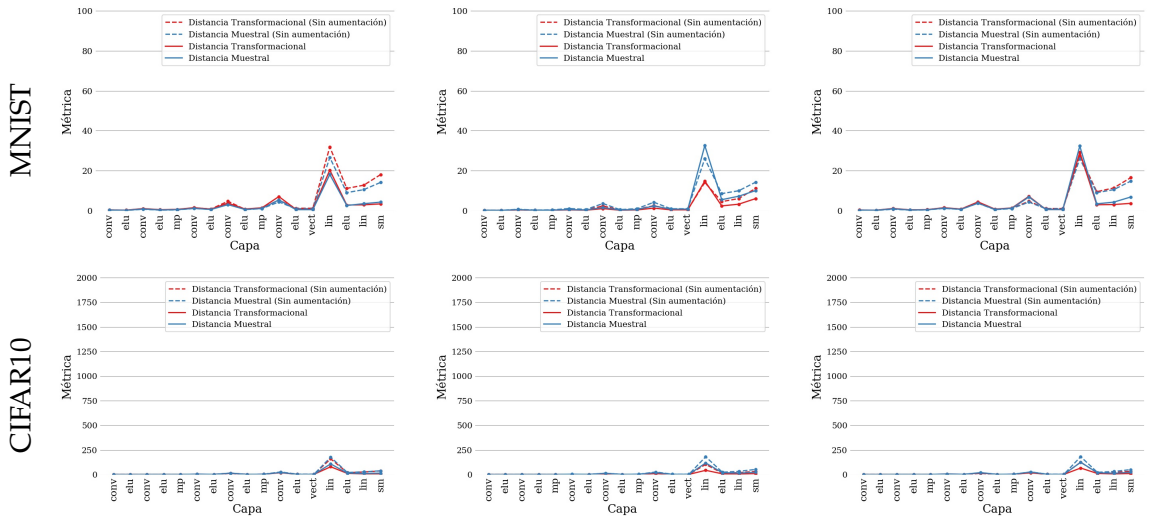


Figura 5.5: Comparación de las métricas Distancia Transformacional y Distancia Muestral para varios conjuntos de transformaciones y BDs, y el modelo SimpleConv. Las líneas punteadas indican modelos entrenados sin aumentación de datos.

5.3.3. Métricas Normalizadas

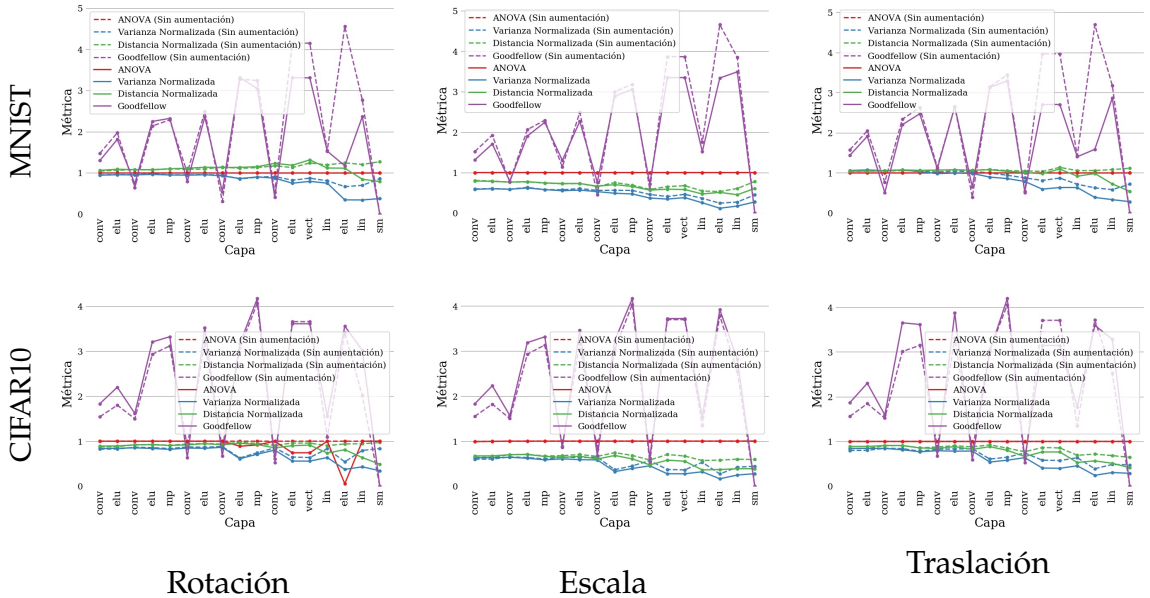


Figura 5.6: Comparación de las métricas normalizadas Varianza Normalizada, Distancia Normalizada, Goodfellow y ANOVA para el modelo SimpleConv. Las líneas punteadas indican modelos entrenados sin aumentación de datos.

La Figura 5.6 muestra el resultado de las métricas normalizadas propuestas Varianza Normalizada y Distancia Normalizada, y el de la métrica ANOVA y la Goodfellow (sección 2.8).

La métrica ANOVA es muy sensible dado que rechaza la hipótesis nula en casi todos los casos y por ende no considera ninguna activación como invariante. Esto indica que esta métrica no resulta de utilidad para medir la invarianza.

Incluimos en esta comparación la métrica Goodfellow (sección 2.8). En este caso, para poder calcular la métrica con el modelo SimpleConv, adaptamos el algoritmo para determinar el umbral t, de modo que en lugar de calcular el percentil 1 %, calculamos el valor z para el cual la normal tiene $P(f <= z) = 0.01$, y utilizamos dicho valor z como umbral t. De esta forma evitamos la necesidad de almacenar todos los valores para calcular el percentil..

Recordamos que para esta métrica, valores más bajos también indican mayor invarianza. La métrica Goodfellow parece detectar la invarianza de los modelos entrenados con aumentación de datos, mayormente en las capas lineales. Si bien su magnitud se mantiene estable, sus unidades varían aproximadamente entre 0 y 5. Además, presenta picos bajos en las capas convolucionales, que no se explican debido a que como demostramos en el apéndice C, las funciones de activación ELU nunca aumentan la varianza.

Las métricas Varianza Muestral y Distancia Normalizada ambas detectan la mayor invarianza de los modelos entrenados con aumentación de datos. Ambas métricas están levemente correlacionadas, como era esperable (sección 2.8). La métrica Varianza Muestral tiene valores ligeramente menores que la Distancia Normalizada. Esto puede deberse a la aproximación que utiliza la Distancia Normalizada. Más allá de esta diferencia en magnitud, la estructura de la invarianza en ambos casos es similar.

5.3.4. Métrica Auto-Equivarianza Normalizada de Varianza

La Figura 5.7 muestra los resultados por capa de la métrica Auto-Equivarianza Normalizada de Varianza (sección 2.8). Recordemos que valores bajos de esta medida indican una mayor auto-equivarianza. Notamos también que la misma solo se calcula para capas cuyas salidas tienen la misma estructura que la entrada. En este caso, dichas capas son las que generan feature maps (capas Convolucionales, de Max Pooling y ELU cuya entrada también es un feature map).

La Auto-Equivarianza Normalizada de Varianza tiene una tendencia negativa en todos los casos, tanto al usar aumentación de datos, como al no hacerlo. También en ambos casos la salida de las capas convolucionales tiende a ser menos equivariante que las salidas de las capas MaxPooling y ELU. Este fenómeno es esperable dado que estas activaciones tienden a incrementar la varianza (apéndice C).

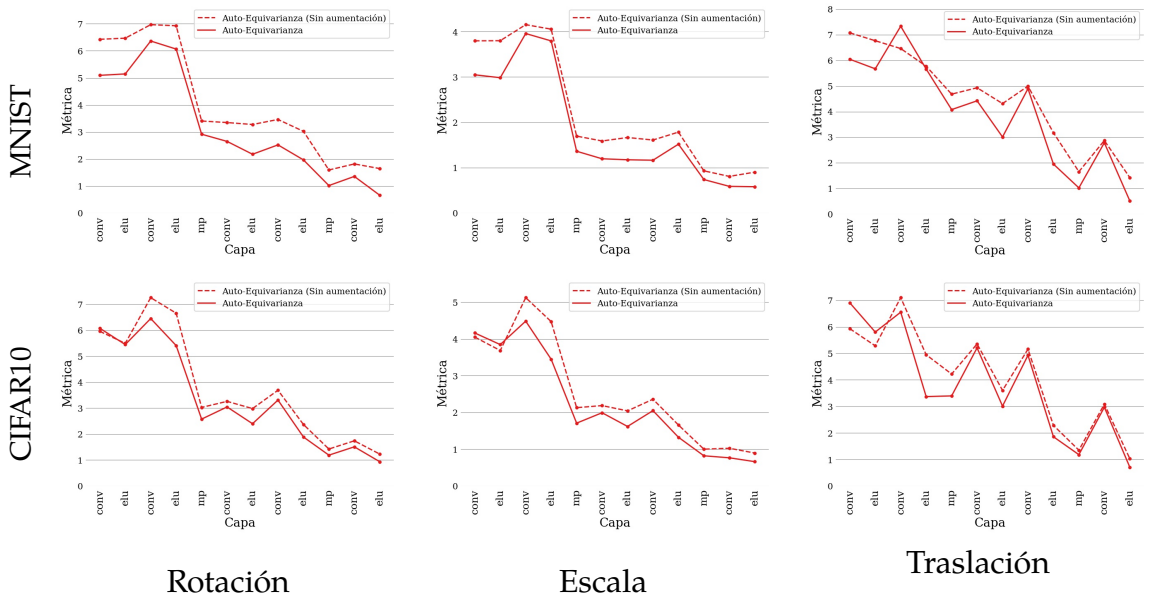


Figura 5.7: Auto-Equivarianza Normalizada de Varianza para el modelo Simple-Conv. Las líneas punteadas indican un modelo entrenado sin aumentación de datos (mejor ver en color)

Por último, los modelos entrenados con aumentación de datos muestran ligeramente más auto-equivarianza que sus contrapartes en casi todas las capas. Las representaciones invariantes causan menor variación en sus salidas, y por ende la distancia entre activaciones se reduce. No obstante, la normalización de las activaciones independiza la distancia de la norma de las activaciones. Por ende, los resultados sugieren que la aumentación de datos para obtener invarianza *también* aumenta la equivarianza.

Por ende, en base a la formulación de la Auto-Equivarianza Normalizada de Varianza (Capítulo 4) podemos decir que la métrica Auto-Equivarianza Normalizada de Varianza mide la auto-equivarianza de las redes.

5.4. Análisis de las Métricas

En esta sección, analizamos las métricas en términos de varios parámetros, como el impacto del tamaño del conjunto de la base de datos, el subconjunto de la base de datos utilizado (entrenamiento/prueba), estabilidad de las métricas a distintas inicializaciones de la red, valores de las métricas para redes con pesos aleatorios, evolución de las métricas durante el entrenamiento, estrategias de normalización y agrupamiento, y complejidad y diversidad de las informaciones.

5.4.1. Tamaño de la base de datos y subconjunto

Analizamos el efecto de variar el tamaño del conjunto de datos utilizado para calcular las métricas, con el objetivo de obtener un estimado empírico del orden de magnitud de las muestras necesarias para que los valores sean estables. También comparamos las diferencias entre evaluar las métricas con el conjunto de prueba o entrenamiento.

Tamaño del conjunto de datos

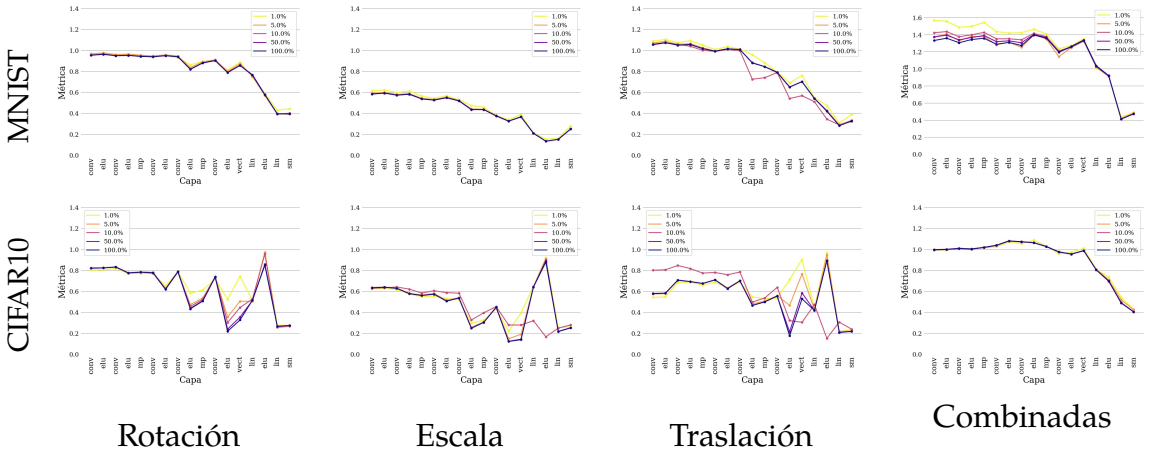


Figura 5.8: VARIANZA NORMALIZADA para distintos tamaños de muestra. Cada línea en cada gráfico corresponde a un distinto porcentaje del conjunto de prueba utilizado para calcular la métrica, el cual tiene de 10000 ejemplos.

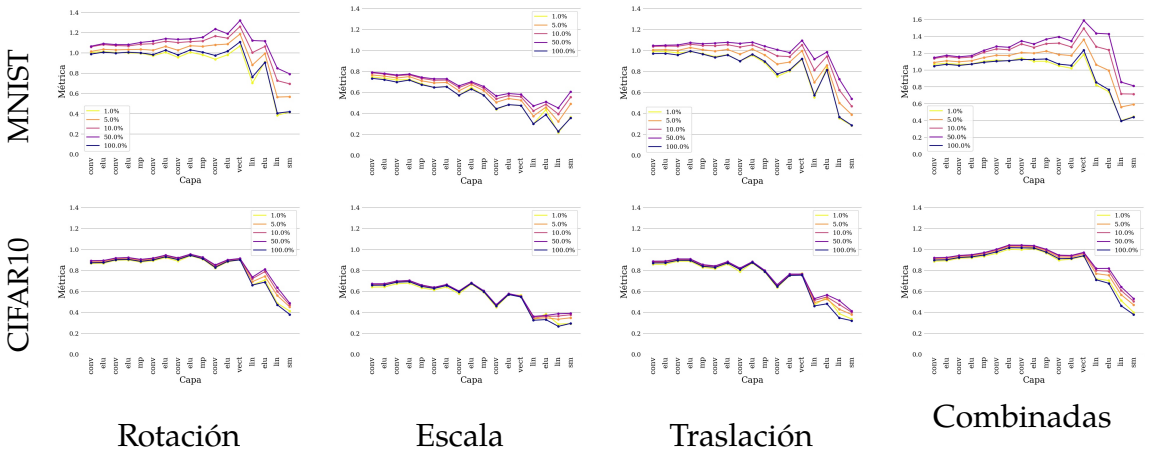


Figura 5.9: DISTANCIA NORMALIZADA para distintos tamaños de muestra. Cada línea en cada gráfico corresponde a un distinto porcentaje del conjunto de prueba utilizado para calcular la métrica, el cual tiene de 10000 ejemplos.

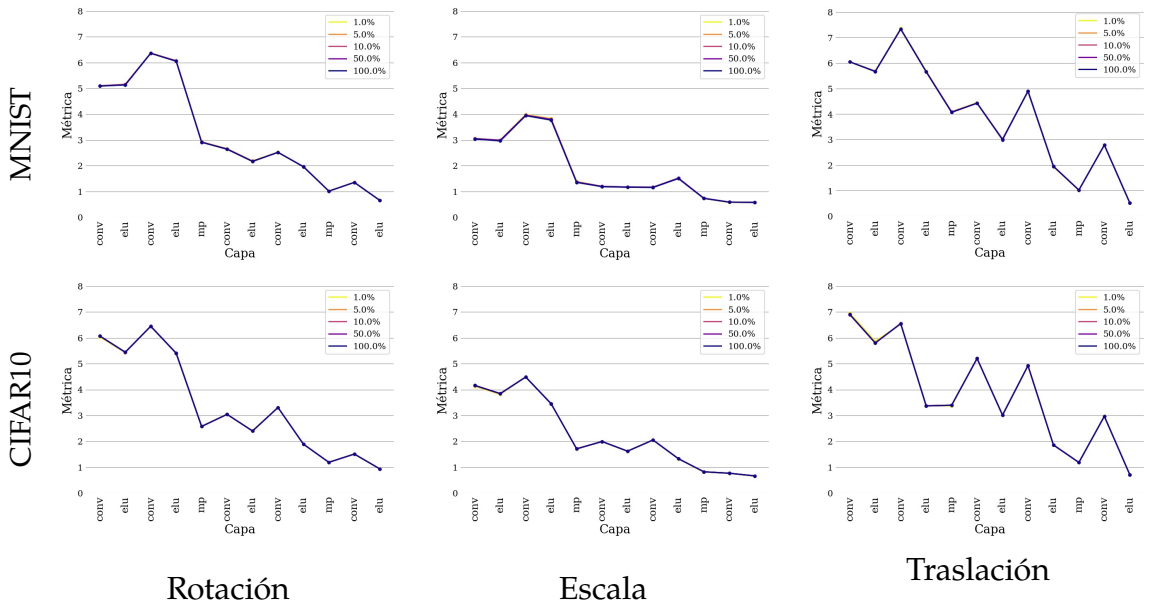


Figura 5.10: Auto-Equivarianza Normalizada de Varianza para distintos tamaños de muestra. Cada línea en cada gráfico corresponde a un distinto porcentaje del conjunto de prueba utilizado para calcular la métrica, el cual tiene de 10000 ejemplos.

Las Figuras 5.8 a 5.10 contienen los resultados para las métricas Varianza Normalizada, Distancia Normalizada y Auto-Equivarianza Normalizada de Varianza. Cada gráfico presenta los valores de las métricas promediados por capas, para distintos tamaños de muestra. Los tamaños de muestra se expresan como porcentajes del conjunto de prueba, que contiene 10000 muestras para ambas BDs.

La Varianza Normalizada es muy estable. Como se observa en la figura, calcular la métrica con la mitad del conjunto de prueba (5000 muestras) o la totalidad (10000 muestras) arroja el mismo resultado. Tamaños de muestra más pequeños causan variaciones indeseadas. En el caso de la métrica Distancia Normalizada, observamos resultados más sensibles al tamaño de muestra, observando que la métrica incrementa a medida que se incrementa el tamaño de muestra. Esto es esperable ya que la métrica de distancia realiza una aproximación de la matriz de distancias. No obstante, como en el caso de Varianza Normalizada, los resultados son muy similares al utilizar 5000 o 10000 muestras, lo que sugiere que la métrica converge con esa cantidad de ejemplos.

Notamos que en todos los casos, la variabilidad de las métricas es mayor para CIFAR10 y que esta BD requiere más muestras para producir una métrica estable.

En el caso de la Auto-Equivarianza Normalizada de Varianza, la Figura 5.10 nos indica que dicha métrica es estable para todos los tamaños del conjunto de prueba. Es posible que esto se deba al esquema de normalización.

Por ende, en el resto de los experimentos de este capítulo utilizaremos siempre 5000 muestras para calcular las métricas. Utilizamos el mismo valor para *todas* las métricas para evitar efectos de tamaño que puedan afectar la comparación de las mismas.

Comparación entre el conjunto de entrenamiento y prueba

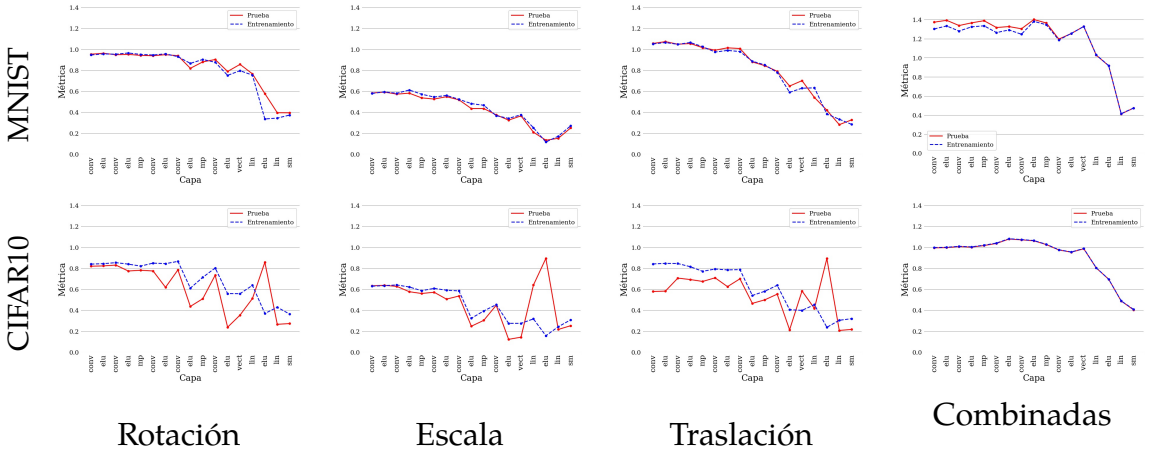


Figura 5.11: Varianza Normalizada calculada con los dos subconjuntos de cada BD, el de entrenamiento y el de prueba. El tamaño de muestra es el mismo en cada caso, 10000 ejemplos.

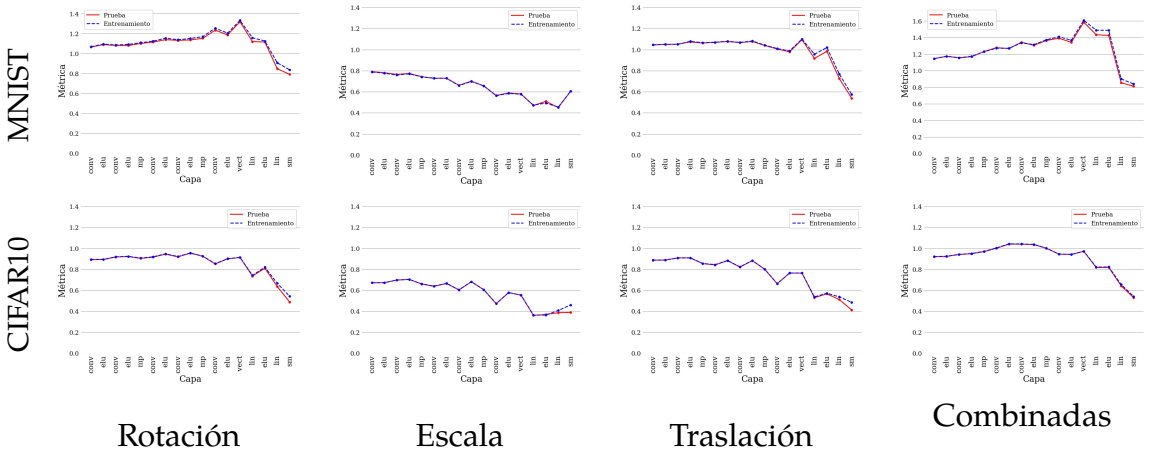


Figura 5.12: Distancia Normalizada calculada con los dos subconjuntos de cada BD, el de entrenamiento y el de prueba. El tamaño de muestra es el mismo en cada caso, 10000 ejemplos.

Las Figuras 5.11 a 5.13 comparan el resultado de las métricas en base al subconjunto de la base de datos utilizada para medirla: el de prueba o el entrenamiento. Para Distancia Normalizada y Auto-Equivarianza Normalizada de Varianza, no

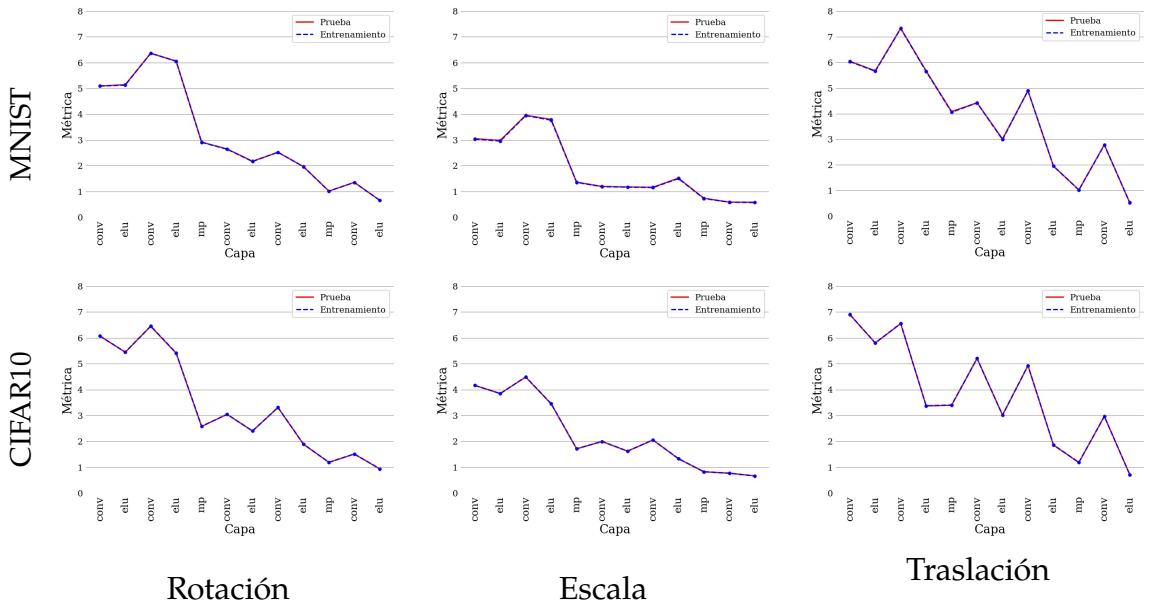


Figura 5.13: Auto-Equivarianza Normalizada de Varianza calculada con los dos subconjuntos de cada BD, el de entrenamiento y el de prueba. El tamaño de muestra es el mismo en cada caso, 10000 ejemplos.

observamos ninguna diferencia entre los dos subconjuntos. En el caso de Varianza Normalizada para MNIST, tampoco. En el caso de Varianza Normalizada para CIFAR10, observamos algunas variaciones. Notablemente, el único patrón relevante que discernimos indica que la penúltima capa lineal tiene mucho mayor variabilidad en el conjunto de prueba que en el conjunto de entrenamiento. Esto podría indicar que la representación aprendida por la red en base al conjunto de entrenamiento en esta capa difiere significativamente para el conjunto de prueba, generando valores fuera de rango de la distribución esperada.

Por último, dado que no observamos en general una gran variación entre dos subconjuntos, elegimos en los experimentos siguientes utilizar siempre el de prueba. Esta elección se basa en las mismas razones por las que utilizamos este conjunto para medir el error o tasa de acierto de la red, ya que en verdad dichas cantidades no son más que otras métricas de los modelos.

5.4.2. Inicialización aleatoria en las métricas

Para estudiar el efecto de la inicialización aleatoria del modelo en las métricas, entrenamos 10 instancias de cada modelo utilizando la misma arquitectura, base de datos y conjunto de transformaciones, hasta que las mismas converjan. Cada instancia del mismo modelo fue inicializada con distintos pesos aleatorios. Luego, todas las instancias fueron evaluadas con la misma métrica.

Las Figuras 5.14 a 5.16 muestran el resultado para todas las métricas. Los resultados sugieren que las métricas varían muy ligeramente con respecto a la inicialización, aunque hemos detectado desviaciones ocasionales (Figura 5.14 (c)). Estos datos sugieren que ésta es una propiedad emergente del modelo aún con la inicialización aleatoria de sus pesos.

Por ende, este patrón debe depender mayormente de la arquitectura del modelo, las muestras y las transformaciones. En todo caso, la estabilidad de las métricas con respecto a la inicialización del modelo es una propiedad deseable ya que no siempre es posible realizar varios entrenamientos y mediciones. De esta forma, podemos evitar utilizar varias corridas de la misma métrica en el resto de los experimentos.

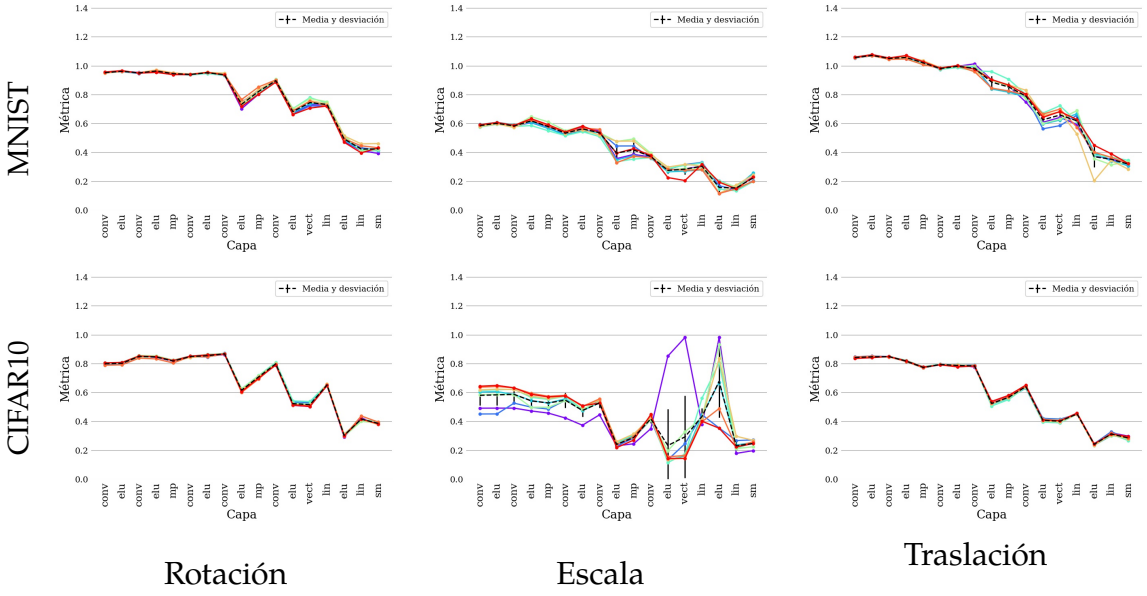


Figura 5.14: Estabilidad a las inicializaciónes de la métrica Varianza Normalizada para el modelo SimpleConv. Cada línea corresponde a la invarianza obtenida para distintas instancias del mismo modelo. La línea con guiones indica el valor promedio de la métrica. Las barras verticales indican la desviación estándar.

5.4.3. Evolución de las Métricas durante el Entrenamiento

Para comprender la dinámica del *aprendizaje* de la invarianza, calculamos la Varianza Normalizada a medida que un modelo se entrena. Evaluamos el modelo cuando se encontraba a ciertos porcentajes de compleción de su entrenamiento, en base al número total de épocas. Los porcentajes utilizados son 0 %, 1 %, 3 %, 5 %, 10 %, 30 %, 50 %, 100 %.

La Figura 5.17 muestra la distribución de las invarianzas en las capas de cada modelo durante el entrenamiento. En todos los casos podemos observar que luego

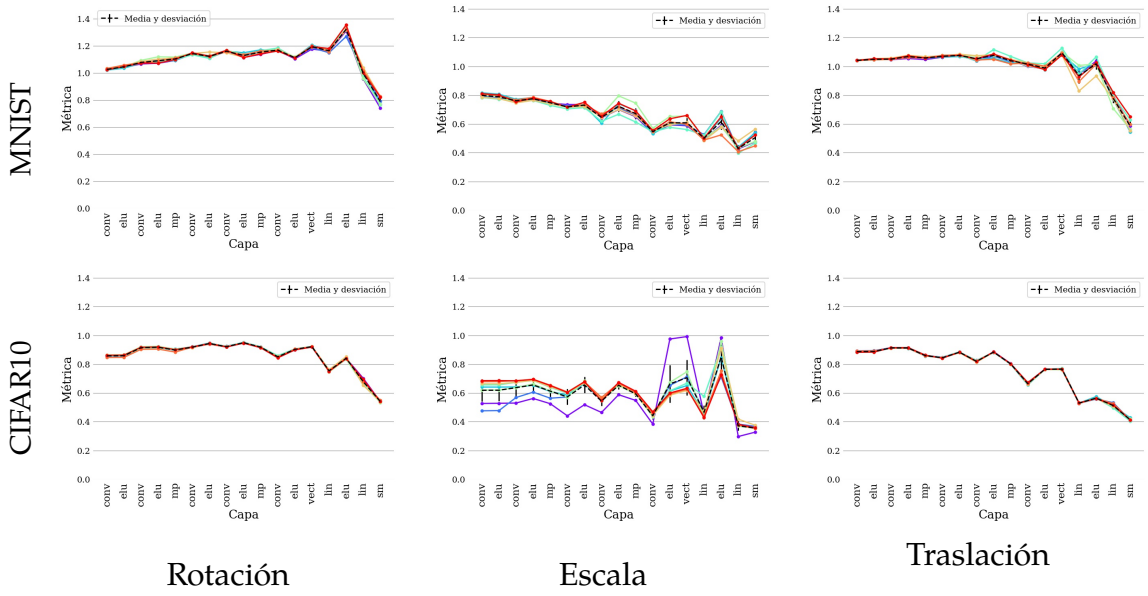


Figura 5.15: Estabilidad a las inicializaciónes de la métrica Distancia Normalizada para el modelo SimpleConv. Cada línea corresponde a la invarianza obtenida para distintas instancias del mismo modelo. La línea con guiones indica el valor promedio de la métrica. Las barras verticales indican la desviación estándar.

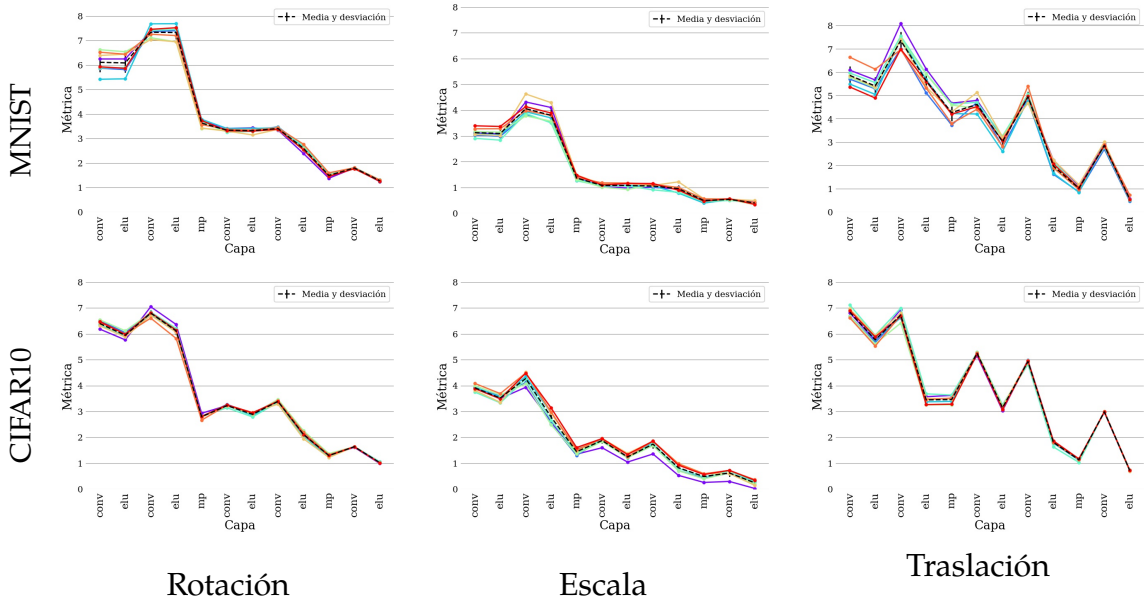


Figura 5.16: Estabilidad a las inicializaciónes de la métrica Auto-Equivarianza Normalizada de Varianza para el modelo SimpleConv. Cada línea corresponde a la invarianza obtenida para distintas instancias del mismo modelo. La línea con guiones indica el valor promedio de la métrica. Las barras verticales indican la desviación estándar.

de las primeras épocas, la invarianza de la red se mantiene mayormente fija. Además, esta estructura es diferente de la estructura de las redes en la época, cero es decir, de redes con pesos aleatorios sin entrenar (sección 5.4.4).

La Varianza Normalizada de las primeras capas convolucionales cambia sólo ligeramente durante el entrenamiento. La Varianza Normalizada de las capas convolucionales finales y las lineales si cambia durante el entrenamiento, mayormente decrementando. En el caso de MNIST esta transición es mucho menor que para CIFAR10, posiblemente debido a las diferentes complejidades de las bases de datos.

No obstante, también observamos que la Varianza Normalizada de las funciones de activación ELU se incrementa durante el aprendizaje. Este efecto puede deberse al ajuste de las otras capas que si tienen parámetros, de modo de utilizar el régimen cercano al valor cero de la función de activación que permite activar o desactivar su salida.

Para CIFAR10, se observa que la Varianza Normalizada de las ELU en ocasiones sube y luego baja. Esto indica también que el proceso de adquisición de la invarianza no es monótono, al menos en estas capas.

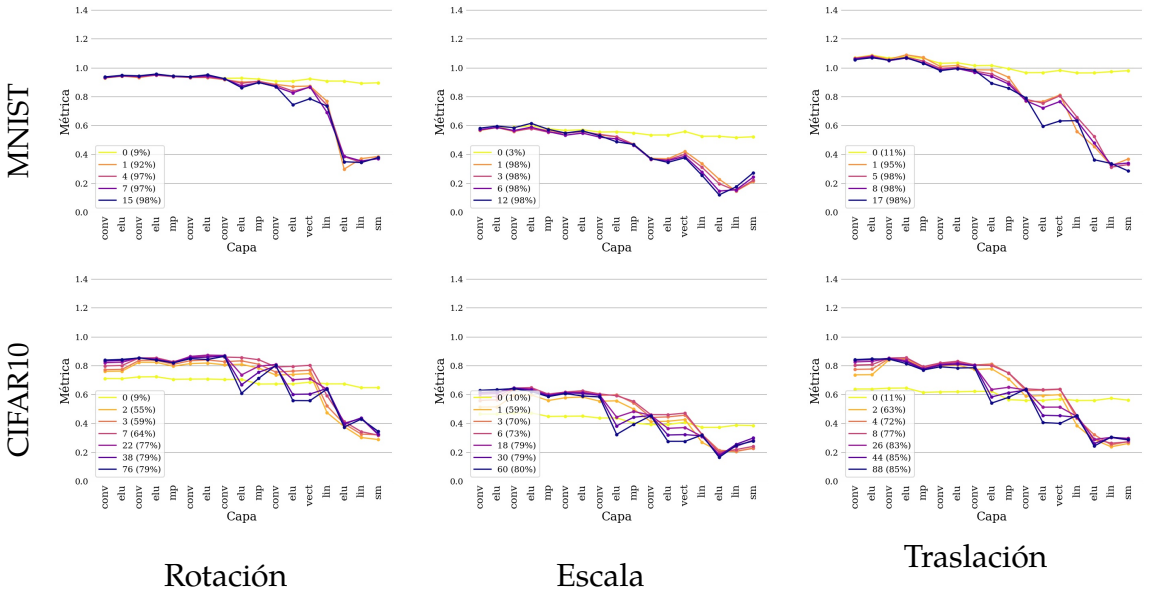


Figura 5.17: Resultados de la métrica Varianza Normalizada durante el entrenamiento. Cada línea corresponde a la métrica evaluada en el mismo modelo pero en una época distinta del entrenamiento. El porcentaje indica la tasa de aciertos del modelo en esa época.

Para la métrica Auto-Equivarianza Normalizada de Varianza (Figura 5.18) observamos una situación similar, aunque con cambios aun más significativos.

5.4.4. Redes con Pesos Aleatorios

Para comprender aún mejor el origen de la estructura del invarianza o auto equivarianza de las redes, estudiamos el valor de las métricas Varianza Normalizada

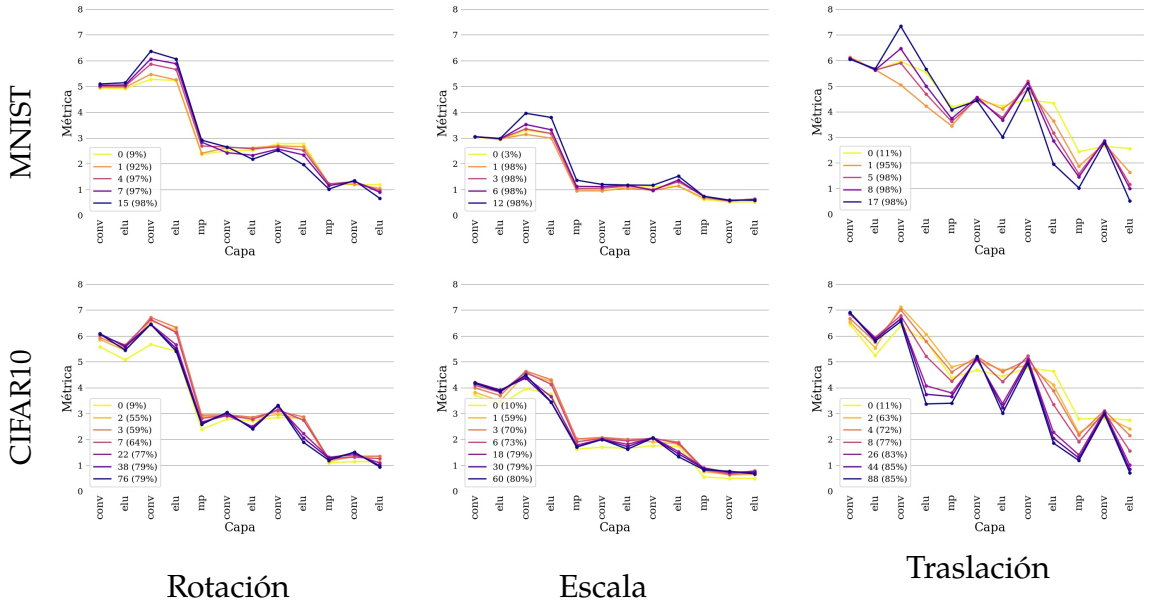


Figura 5.18: Resultados de la métrica Auto-Equivarianza Normalizada de Varianza durante el entrenamiento. Cada línea corresponde a la métrica evaluada en el mismo modelo pero en una época distinta del entrenamiento. El porcentaje indica la tasa de aciertos del modelo en esa época.

y Auto-Equivarianza Normalizada de Varianza para modelos sin entrenar, es decir, modelos con pesos completamente aleatorios. Tanto las capas lineales como las convolucionales utilizan como algoritmo de inicialización el esquema de Kaiming uniforme [He+15]. Dado que la inicialización es aleatoria, calculamos la métrica para ocho modelos aleatorios en cada caso.

La Figura 5.19 muestra la Varianza Normalizada para varios modelos con pesos aleatorios. Se puede observar que la Varianza Normalizada depende altamente del conjunto de datos y el conjunto de transformaciones, pero no del conjunto de pesos aleatorios de la red. Esto sugiere que dada una inicialización controlada, la invarianza es mayormente una propiedad de la arquitectura de la red en lugar de los valores específicos de los pesos. No obstante, como vimos en la sección 5.4.3, la estructura de la invarianza cambia con el entrenamiento, por ende esta lógica sólo se aplica a los modelos sin entrenar.

Por otro lado, es interesante que la invarianza de las redes sin entrenar no tiene tendencia, es decir no varía en base a las capas. Como notamos en la sección 5.3, los modelos entrenados con aumentación de datos muestran una tendencia negativa de la varianza, con más invarianza en las etapas finales y menos en las primeras.

Dado que al entrenar la red la invarianza se estabiliza (sección 5.4.2), podemos concluir que la adquisición de invarianza a través del entrenamiento con aumentación de datos no es un proceso con una sola solución. Es decir, de la misma manera

en que el entrenamiento de una red no consiste en buscar una inicialización óptima para luego encontrar un único mínimo global [Kaw16], el proceso de adquisición de invarianza a partir de una inicialización arbitraria llega a obtener una invarianza en alguno de los mínimos locales en los que se encuentra la red al finalizar su entrenamiento.

En el caso de la Auto-Equivarianza Normalizada de Varianza (Figura 5.20)), podemos observar que la auto-equivarianza no varía significativamente, y además es muy similar al de los modelos ya entrenados (sección 5.4.2). Esto indica también que esta propiedad no depende de los pesos de la red, sino de su arquitectura.

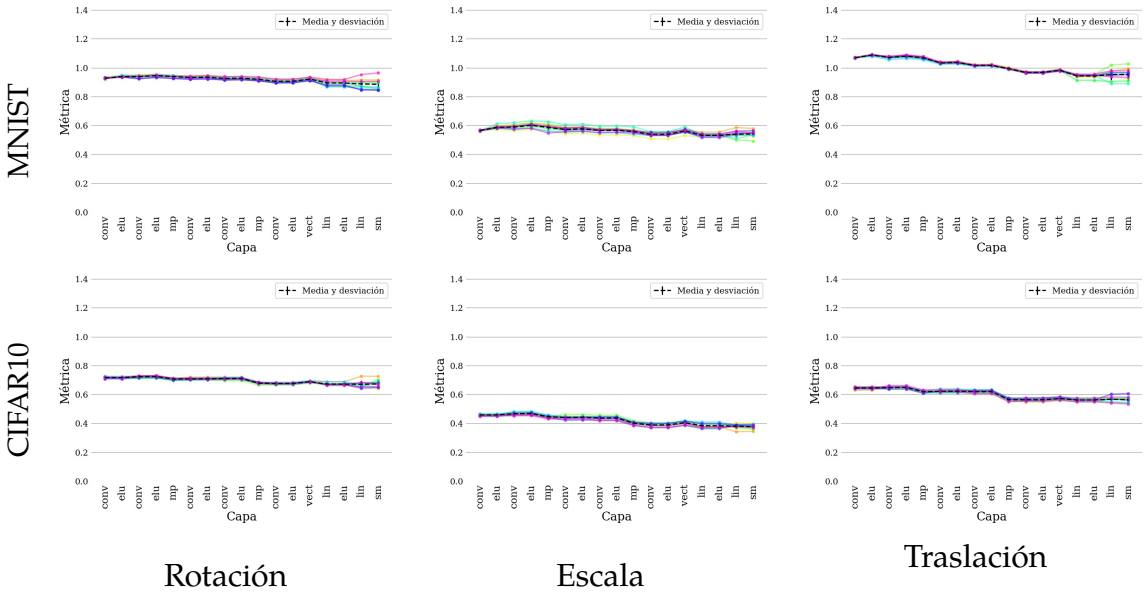


Figura 5.19: Varianza Normalizada para SimpleConv con pesos aleatorios. Cada línea corresponde a la métrica calculada con distintos pesos aleatorios.

5.4.5. Complejidad de las Transformaciones

Medimos el efecto de variar la complejidad de las transformaciones al calcular la métrica.

Para ello entrenamos los modelos utilizando aumentación de datos con un conjunto de transformaciones. Luego calculamos las métricas con ese mismo conjunto de transformaciones pero también con subconjuntos de menor intensidad (pero del mismo tipo). Nos referiremos a estos conjuntos como las *transformaciones de entrenamiento* y las *transformaciones de prueba*. Por ejemplo, entrenamos un modelo con traslaciones de hasta $8px$ como aumentación de datos. Luego, medimos la invarianza del modelo a traslaciones de $2px$, de $4px$, y también de $8px$.

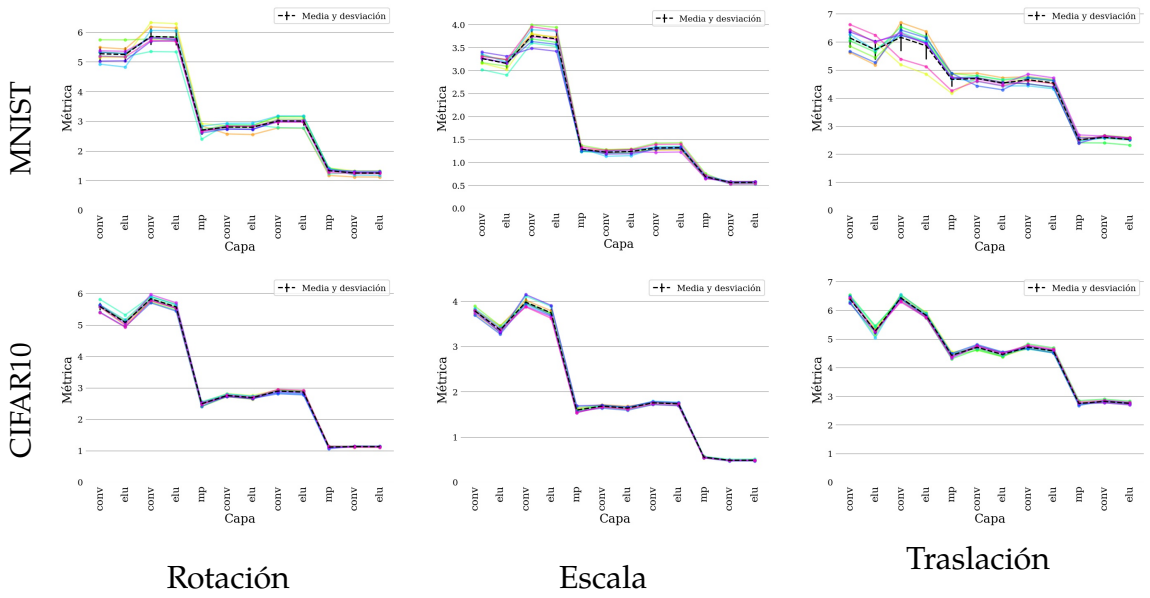


Figura 5.20: Auto-Equivarianza Normalizada de Varianza para SimpleConv con pesos aleatorios. Cada línea corresponde a la métrica calculada con distintos pesos aleatorios.

Utilizamos este esquema para los tres tipos de transformaciones afines previamente mencionadas: rotación, escala y traslación. Entonces, el conjunto de transformaciones de entrenamiento siempre contiene a todas las transformaciones de cada tipo. En el conjunto de transformaciones de prueba, para cada tipo de transformación variamos la intensidad de las transformaciones. No obstante, mantenemos fija *cantidad* de transformaciones para controlar efectos de tamaño en la medición de la muestra.

De esta forma, podemos analizar si la equivarianza de un modelo varía *estructuralmente* para transformaciones más complejas, o si simplemente cambia la escala de la equivarianza.

Las Figura 5.21 muestra el resultado de las métrica Varianza Normalizada por capas. Podemos observar que la estructura de la equivarianza para cada tipo de transformación es distinta. No obstante, en la mayoría de los casos aunque el orden de magnitud de la métrica cambia (se incrementa) con la complejidad, su estructura no lo hace. Esto indica que las transformaciones menos complejas simplemente generan menos varianza, lo cual es esperable, de forma consistente en todas las capas del modelo. Esto sugiere que la invarianza se codifica de forma distinta para cada tipo de transformación, pero no para sus distintas complejidades. La Figura 5.22 muestra una situación similar para la Auto-Equivarianza Normalizada de Varianza.

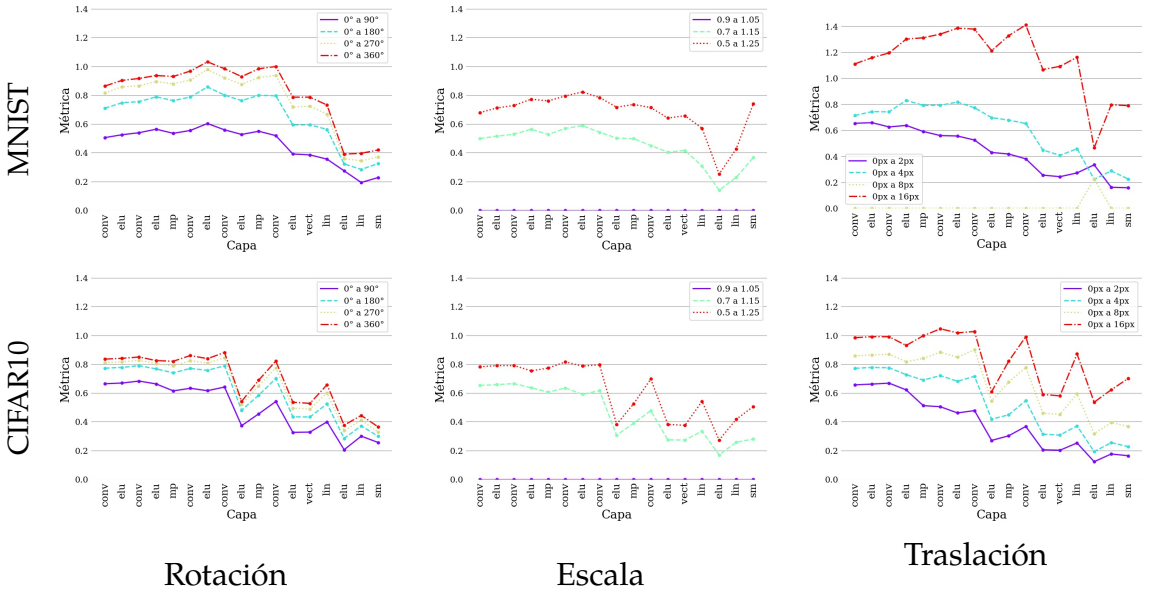


Figura 5.21: Variaciones de Varianza Normalizada en función de la complejidad de la transformación de prueba para SimpleConv. Cada línea corresponde a una transformación de prueba diferente. Los gráficos de cada columna corresponden a diferentes transformaciones de entrenamiento.

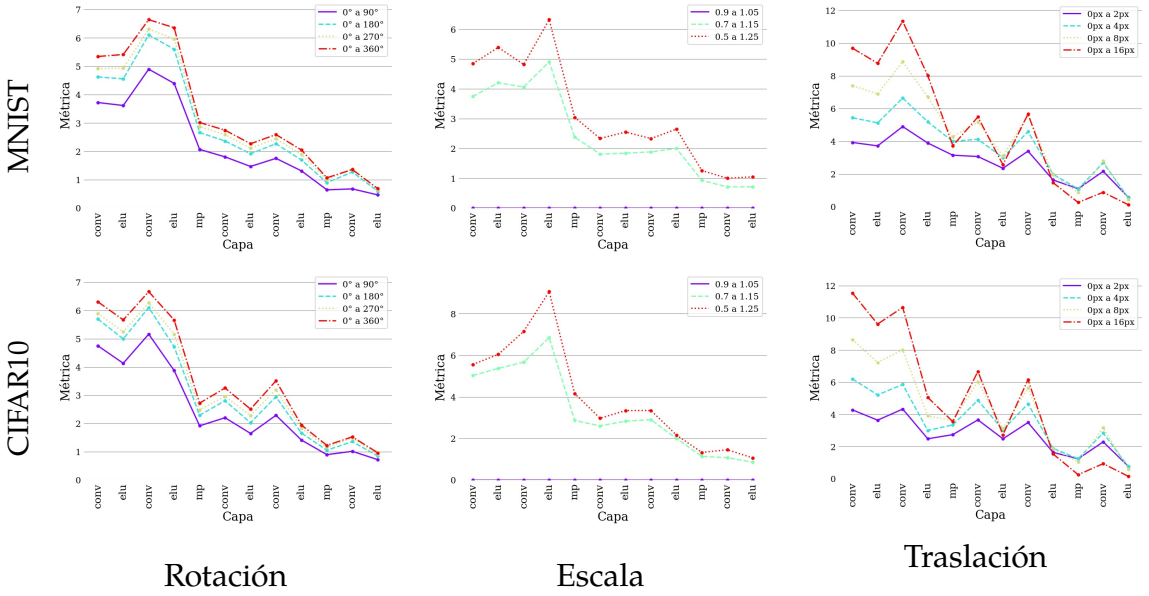


Figura 5.22: Variaciones de Auto-Equivarianza Normalizada de Varianza en función de la complejidad de la transformación de prueba para SimpleConv. Cada línea corresponde a una transformación de prueba diferente. Los gráficos de cada columna corresponden a diferentes transformaciones de entrenamiento.

5.4.6. Diversidad de las Transformaciones

Medimos también el efecto de la diversidad de las transformaciones en las métricas. Para ello, nuevamente entrenamos un modelo utilizando un conjunto de trans-

formaciones de aumentación de datos (*transformaciones de entrenamiento*), y luego medimos la invarianza a ese conjunto de transformaciones pero también a otros de distintos tipos (*transformaciones de prueba*). De esta forma, podemos analizar si la equivarianza de un modelo varía estructuralmente para distintos tipos de transformaciones. Esto indicaría si es una propiedad dinámica, dependiente de la transformación de prueba, o si es una propiedad estática dependiente de la estructura de la red.

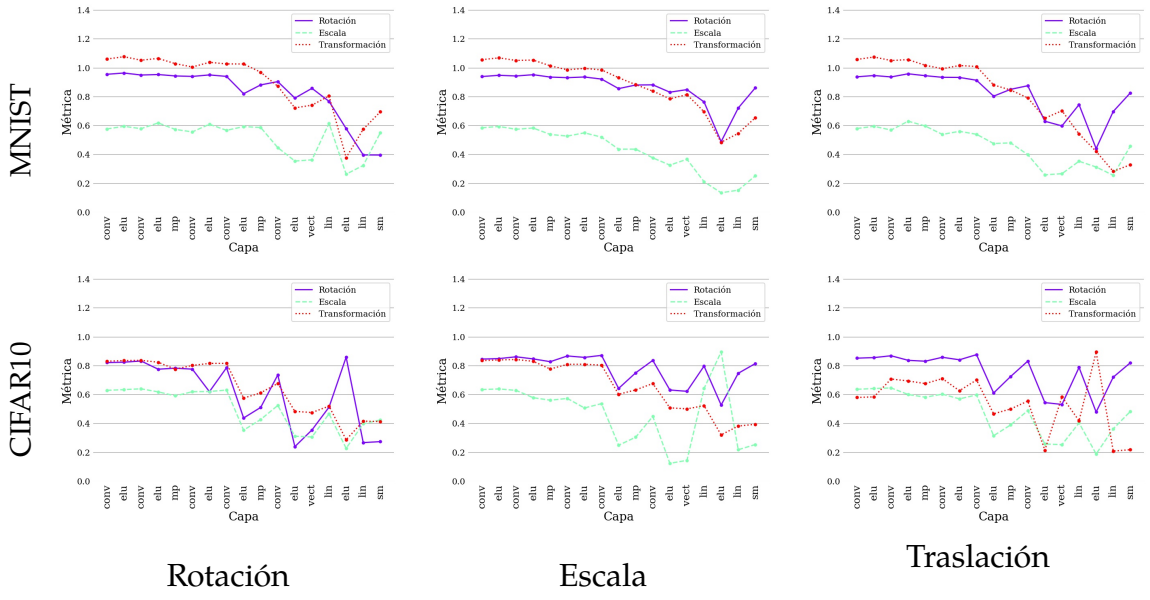


Figura 5.23: Varianza Normalizada para el modelo SimpleConv en función de la diversidad de las transformaciones de prueba. Cada línea corresponde a una transformación de prueba diferente. Los gráficos de cada columna corresponden a diferentes transformaciones de entrenamiento.

Las Figuras 5.23 y 5.24 muestran la Varianza Normalizada y Auto-Equivarianza Normalizada de Varianza de SimpleConv. Los resultados no presentan ningún tipo de patrón para las combinaciones de transformaciones de entrenamiento y prueba. Por ende, indicarían que estas propiedades son dinámicas, es decir, dependientes de la transformación de prueba y no sólo de la de entrenamiento. No obstante, para el caso de Auto-Equivarianza Normalizada de Varianza observamos que su valor es similar para cada transformación de prueba, sin importar la transformación de entrenamiento, y variando sólo ligeramente con la base de datos. Esto indica que la auto-equivarianza es una propiedad más propia de la arquitectura y la transformación que la invarianza.

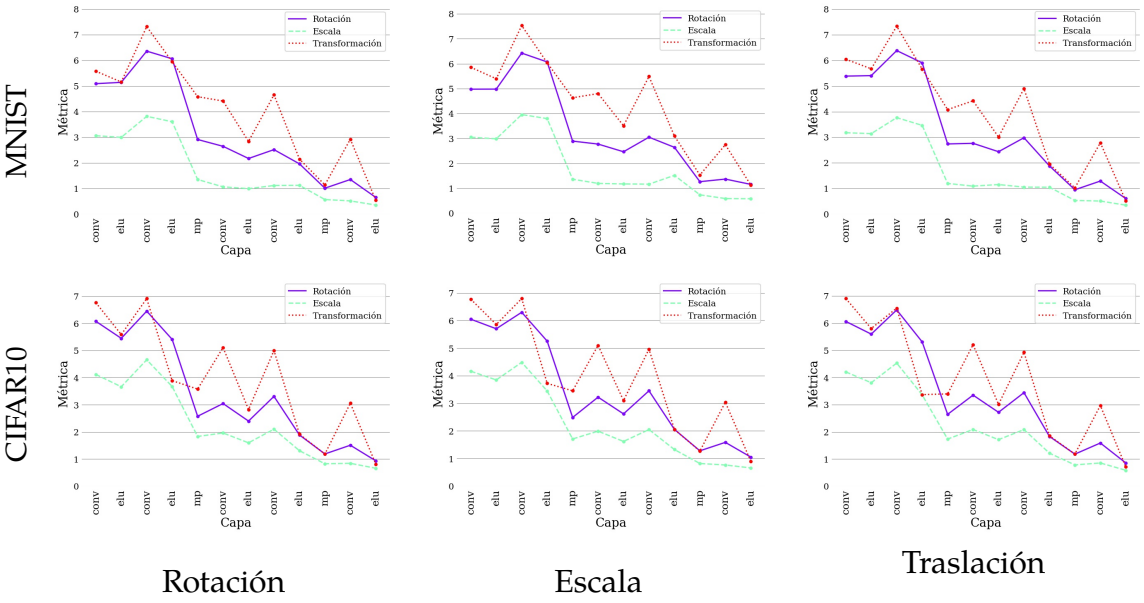


Figura 5.24: Auto-Equivarianza Normalizada de Varianza para el modelo SimpleConv en función de la diversidad de las transformaciones de prueba. Cada línea corresponde a una transformación de prueba diferente. Los gráficos de cada columna corresponden a diferentes transformaciones de entrenamiento.

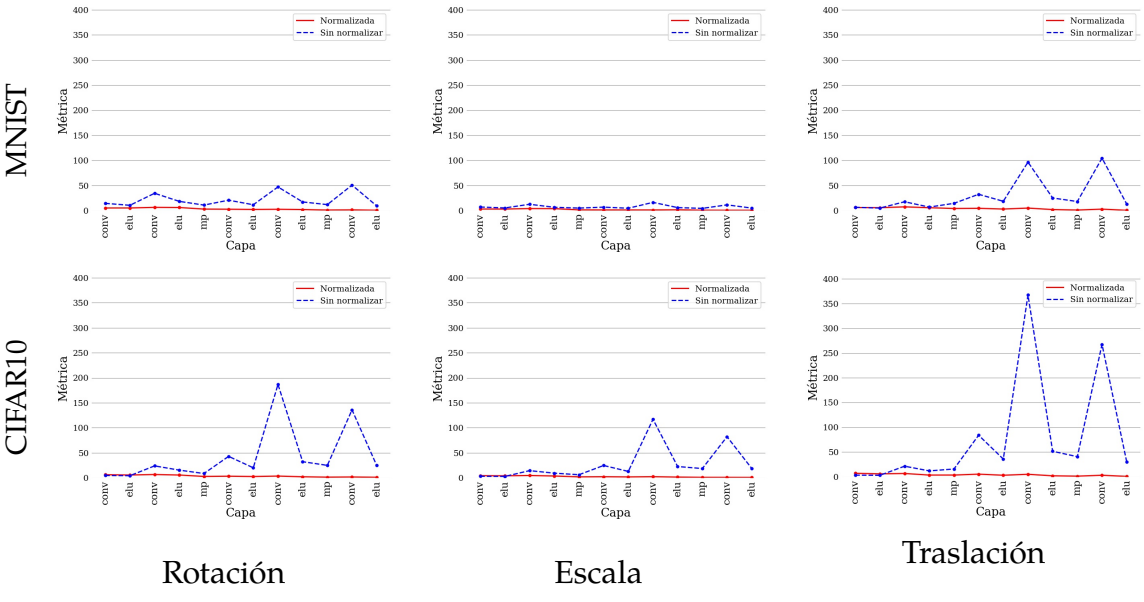


Figura 5.25: Auto-Equivarianza Normalizada de Varianza con y sin normalización para el modelo SimpleConv.

5.4.7. Normalización para la Auto-Equivarianza

Para el estudiar el efecto de la normalización de las activaciones para la métrica Auto-Equivarianza Normalizada de Varianza, evaluamos dicha métrica con y sin la normalización. La Figura 5.25 muestra los resultados para cada capa. Se puede

observar claramente que sin la normalización las magnitudes varían significativamente con respecto a la transformación, la base de datos y las capas. Esto justifica la necesidad de normalizar también en la métrica Auto-Equivarianza Normalizada de Varianza.

5.4.8. Transferencia de Equivarianza entre Bases de Datos

Para evaluar si la estructura de la invarianza es una propiedad que depende de la base de datos, entrenamos modelos con una base de datos, y luego computamos las métricas tanto con la base de datos de entrenamiento así como con otra. De esta forma, podemos cuantificar la importancia de la base de datos de entrenamiento en las métricas.

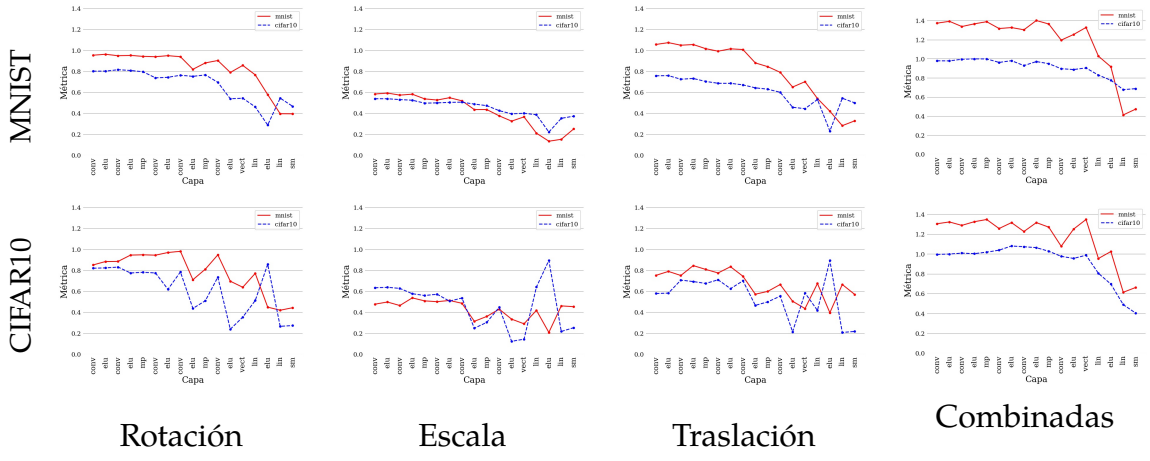


Figura 5.26: Varianza Normalizada medida con ambas BDs para un modelo Simple-Conv entrenado con una sola de las BDs.

La Figura 5.26 muestra el resultado de la Varianza Normalizada para las dos bases de datos. Si bien podemos observar una correlación, en general los valores de la Varianza Normalizada son distintos. Sorprendentemente, los modelos entrenados con MNIST obtienen un valor menor de Varianza Normalizada al evaluarlos con CIFAR10. Además, estos valores son muy similares a los correspondientes obtenidos al entrenar con CIFAR10. Esto indica que es más importante para la métrica la base de datos que se utiliza para evaluarla que la que se utilizó para entrenar el modelo.

Por otro lado, observamos que la penúltima capa lineal es especialmente sensible al cambio de base de datos. Esto confirma los resultados presentados en el Capítulo 3 que indican que es necesario re-entrenar dicha capa para realizar transferencia de aprendizaje de la invarianza.

Estas dos evidencias sugieren que en términos de invarianza, no habría necesidad de realizar una transferencia de aprendizaje salvo por las últimas capas de la red.

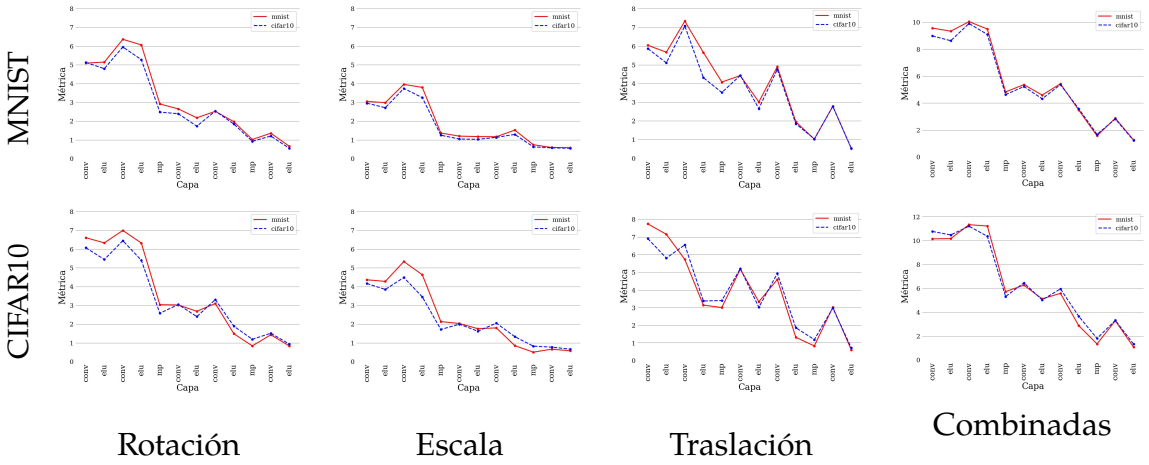


Figura 5.27: Auto-Equivarianza Normalizada de Varianza medida con ambas BDs para un modelo SimpleConv entrenado con una sola de las BDs.

En el caso de la Auto-Equivarianza Normalizada de Varianza, la Figura 5.27 indica sorprendentemente que los resultados son muy similares al evaluar con cualquiera de las dos bases de datos. Este fenómeno sugiere que una vez adquirida la auto-equivarianza su valor no depende de la base de datos con la cual se la mide. Esto refuerza la evidencia de las otras secciones que indican que la auto-equivarianza, medida por la métrica Auto-Equivarianza Normalizada de Varianza, pareciera ser una propiedad más fundamental del modelo y no tanto de su entrenamiento.

5.4.9. Métricas estratificadas

Realizamos experimentos con las versiones estratificadas de las métricas (Capítulo 4). En este caso, las métricas se evaluaron utilizando el conjunto de entrenamiento de forma de que la versión estratificada pueda tener la misma cantidad de muestras en cada una de sus clases como en la versión original de las métricas.

Las Figuras 5.28 y 5.29 presentan el resultado de las versiones estratificadas de las métricas Varianza Normalizada y Auto-Equivarianza Normalizada de Varianza respectivamente. En el caso de la métrica Auto-Equivarianza Normalizada de Varianza, no observamos ningún tipo de diferencia entre ambas versiones; esto refuerza aún más la noción de que la auto-equivarianza es una propiedad del modelo y no del conjunto de datos con el que se evalúa.

Para CIFAR10, la métrica Varianza Normalizada no varía de su versión estratificada, excepto por las capas de clasificación. En cambio, para MNIST la versión estratificada tiene significativamente más Varianza Normalizada que la normal. Es posible que esto se deba a que la Varianza Muestral de cada clase es menor, ya

que los ejemplares de las mismas se parecen mucho, un efecto que es más grande en MNIST debido al fondo negro y la simplicidad de las imágenes. Por otro lado, la Varianza Transformacional debe ser similar a la original, por ende la Varianza Normalizada es mayor en el caso estratificado.

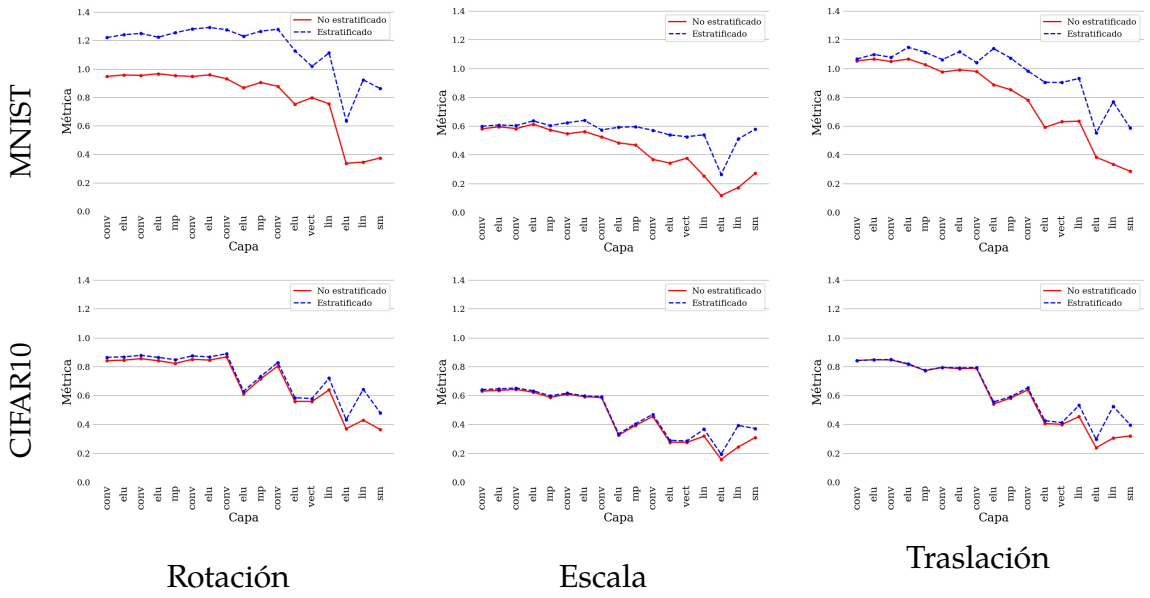


Figura 5.28: Comparación de la métrica Varianza Normalizada con su versión estratificada para el modelo SimpleConv

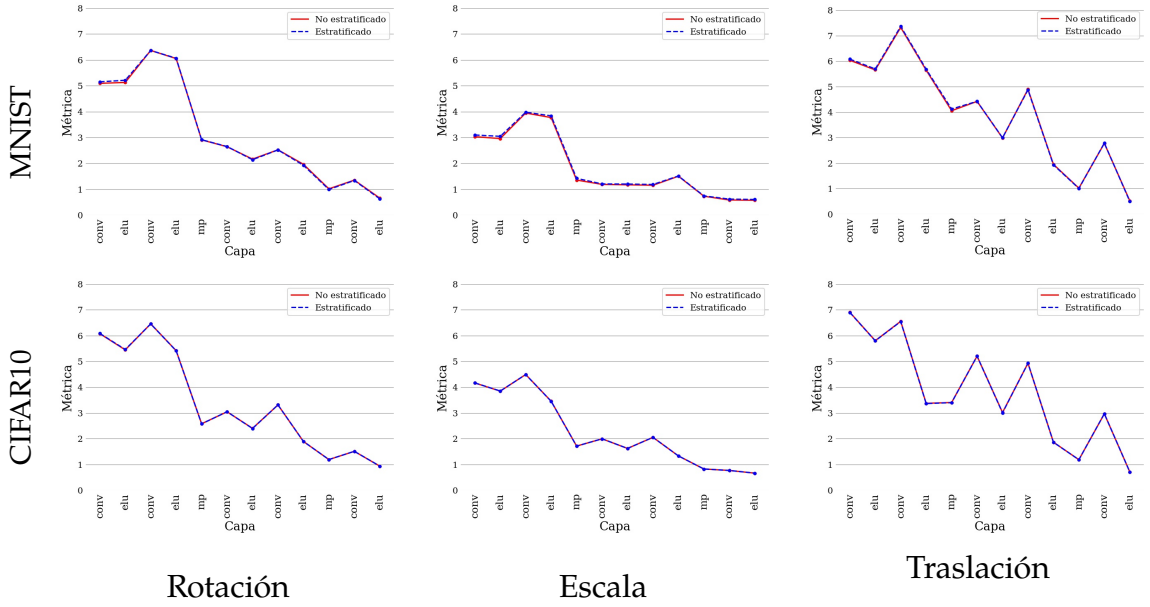


Figura 5.29: Comparación de la métrica Auto-Equivarianza Normalizada de Varianza con su versión estratificada para el modelo SimpleConv

5.5. Análisis de Modelos de CNN

En esta sección, analizamos diversas características de las arquitecturas modernas de CNN, de modo de caracterizarlas en términos de su equivarianza.

En particular, variamos los siguientes parámetros:

- Función de Activación
- Tamaño del Kernel Convolucional
- Uso de Max-Pooling o Convoluciones con *paso* $= 2$
- Uso de Normalización por Lotes (BN)

Además, comparamos la equivarianza de con los modelos ResNet, AllConvolutional, VGG(sección 2.4) y TI-Pooling (sección 2.7).

5.5.1. Funciones de Activación

Para probar el efecto de las funciones de activación en las métricas, experimentamos variando las mismas para el modelo SimpleConv. Reemplazamos la función *ELU* original con *ReLU*, *PReLU*, *TanH* and *Sigmoid*. Los modelos fueron entrenados desde cero para cada una de estas funciones.

La Varianza Normalizada (Figura 5.30) no muestra diferencias significativas entre las distintas funciones de activación. No obstante, podemos observar que las funciones ELU, ReLU y en menor medida PReLU generan picos con valores menores de Varianza Normalizada luego de algunas capas convolucionales, y picos de valores mayores de Varianza Normalizada para la penúltima capa lineal de la red. Este comportamiento es esperable, dado que dichas funciones de activación naturalmente reducen la varianza de las activaciones (apéndice C).

En el caso de la Auto-Equivarianza Normalizada de Varianza (Figura 5.31) si bien los valores difieren entre función de activación no se discierne ningún patrón particular. Esto sugiere que dicha variación es simplemente debida a fluctuaciones normales de la codificación entre las funciones de activación, dado que, por ejemplo, el tipo de transformación afecta significativamente más a la Auto-Equivarianza Normalizada de Varianza que la elección de función de activación.

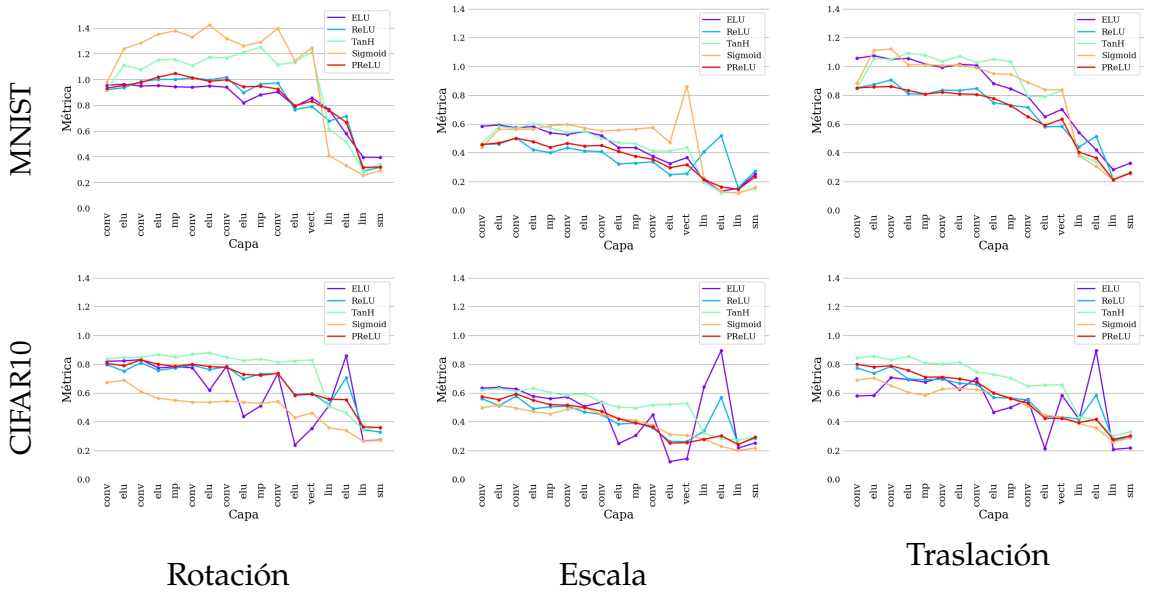


Figura 5.30: VARIANZA NORMALIZADA para el modelo SIMPLECONV con diferentes funciones de activación que reemplazan a la ELU original.

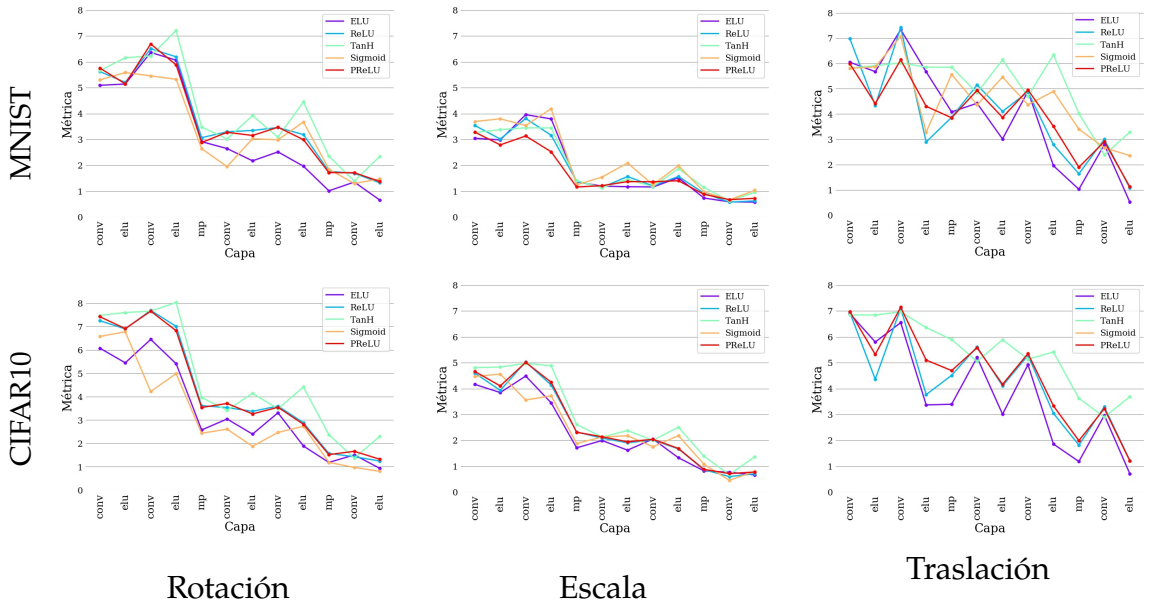


Figura 5.31: AUTO-EQUIVARIANZA NORMALIZADA DE VARIANZA para el modelo SIMPLECONV con diferentes funciones de activación que reemplazan a la ELU original.

5.5.2. Tamaño del Kernel Convolucional

Variamos el tamaño del kernel de las capas convolucionales para evaluar su impacto en la invarianza o auto-equivarianza del modelo. Para ello, entrenamos desde cero modelos kernels cuadrados y tamaños de kernel $k = 3$ (valor utilizado en el resto de los experimentos), $k = 5$ y $k = 7$, y evaluamos la métrica con cada uno por separado.

En el caso de la Varianza Normalizada (Figura 5.32), no encontramos diferencias significativas entre los distintos tamaños, excepto por los picos causados por $k = 3$ en CIFAR10, los cuales desaparecen con $k = 5$ o $k = 7$. La tasa de acierto de los tres modelos es similar en todos los casos, con lo cual interpretamos estos picos como un indicativo de algún factor que requiere más estudio para su comprensión. En el caso de la Auto-Equivarianza Normalizada de Varianza (Figura 5.33), en todos los casos $k = 3$ obtuvo valores más bajos de la métrica. Esto sugiere que los tamaños de kernel más grandes realizan transformaciones más intensas que reducen la auto-equivarianza del modelo.

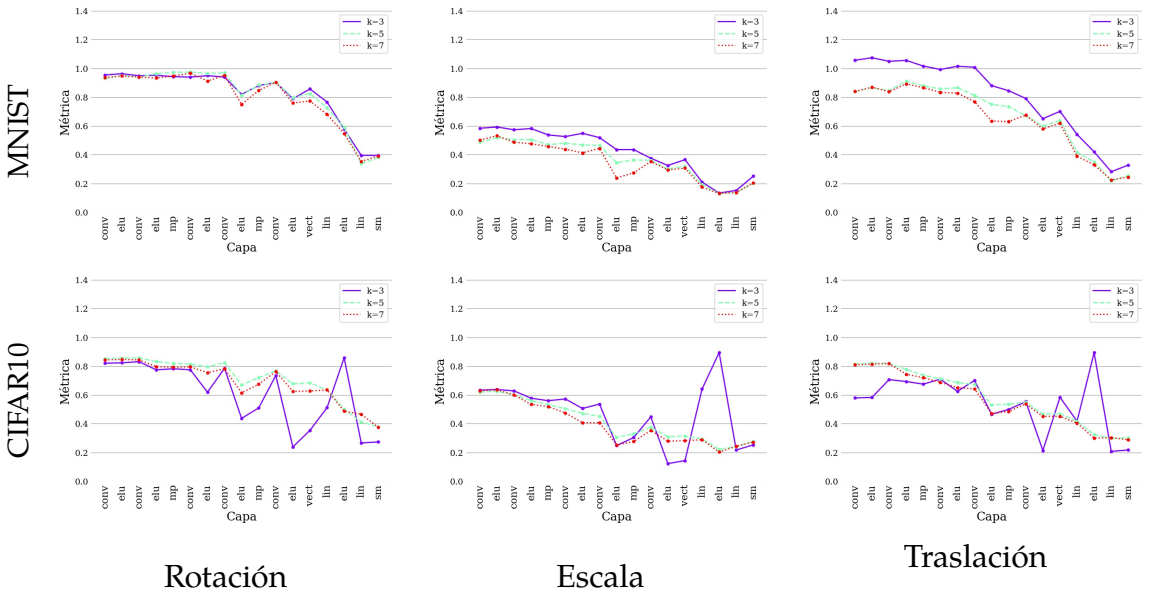


Figura 5.32: Varianza Normalizada para distintos tamaños de kernel en el modelo SimpleConv.

5.5.3. Max-Pooling

Las capas de MaxPooling permiten reducir la dimensión espacial de un feature map, pero dicho efecto también puede lograrse con una Convolución con *paso* $= 2$. Para comparar las diferencias entre estos dos enfoques, entrenamos modelos SimpleConv tradicionales que utilizan las capas MaxPooling, así como modelos modificados que reemplazan dicha capa por una convolución con *paso* $= 2$ y tamaño de kernel 2×2.

La Figura 5.34 indica que la Varianza Normalizada es similar en ambos casos. No obstante, el uso de convoluciones con *paso* $= 2$ evita los picos de Varianza Normalizada en la penúltima capa lineal de la red. Al igual que en la sección 5.5.2, se requerirían otros experimentos para determinar el origen de dichos picos. La métrica

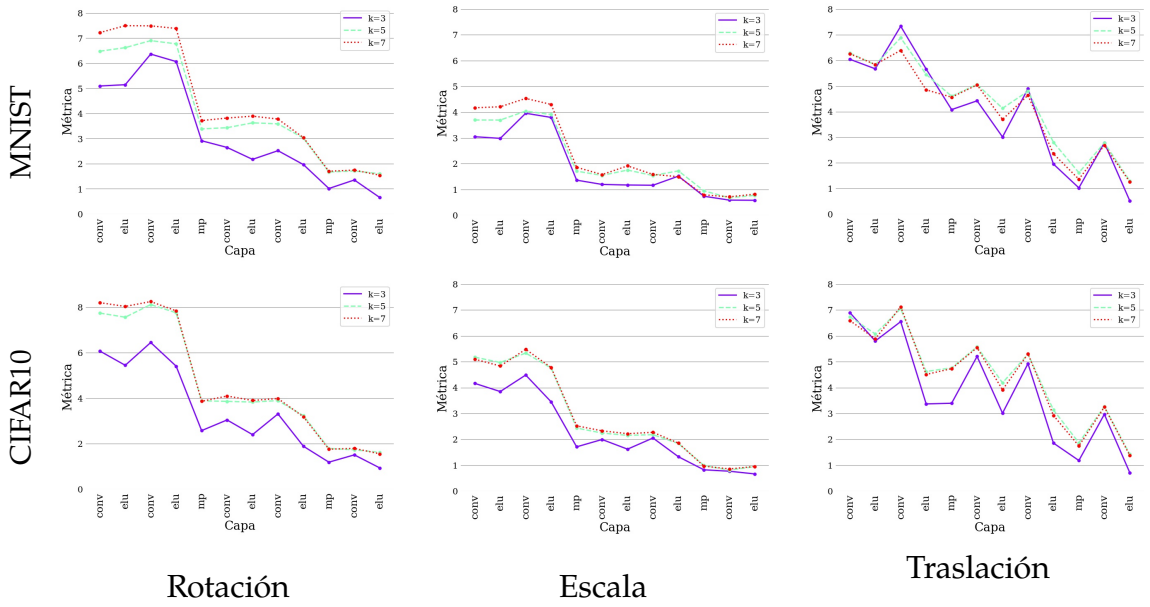


Figura 5.33: Auto-Equivarianza Normalizada de Varianza para distintos tamaños de kernel en el modelo SimpleConv.

Auto-Equivarianza Normalizada de Varianza (Figura 5.35), por otro lado, obtiene valores más bajos (más auto-equivarianza) con MaxPooling. Este efecto puede deberse a que las capas de convolución con *paso* $= 2$ realizan una transformación aprendida de su entrada, con lo cual la modifican más que las capas MaxPooling correspondientes que simplemente realizan un escalado de su entrada con el *máximo* como función de muestreo.

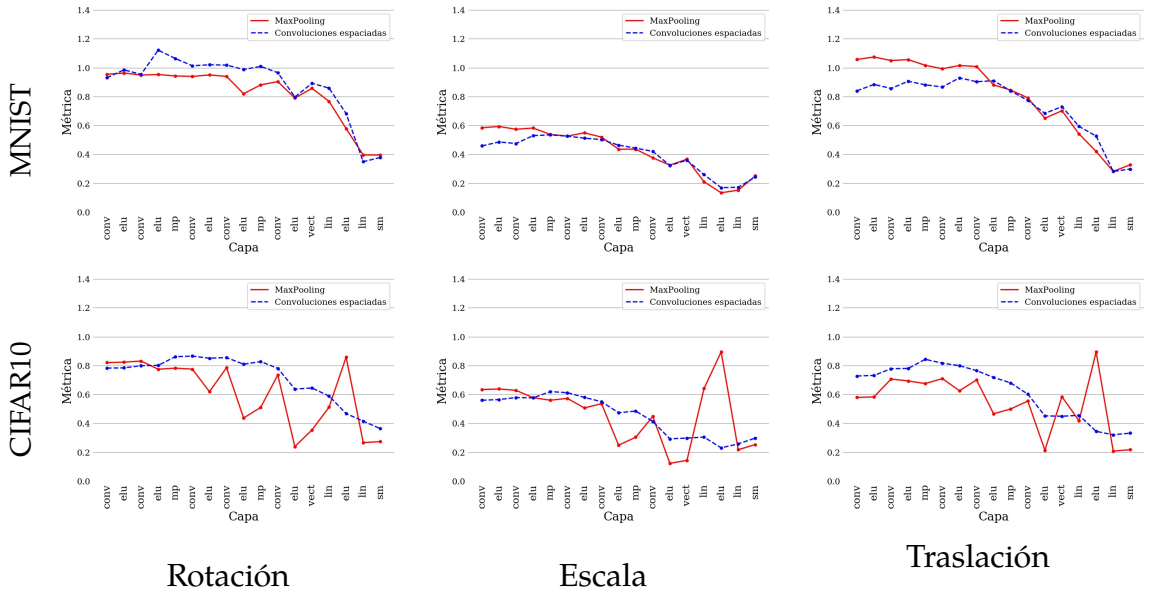


Figura 5.34: Varianza Normalizada para SimpleConv con capas MaxPooling o Convoluciones con *paso* $= 2$

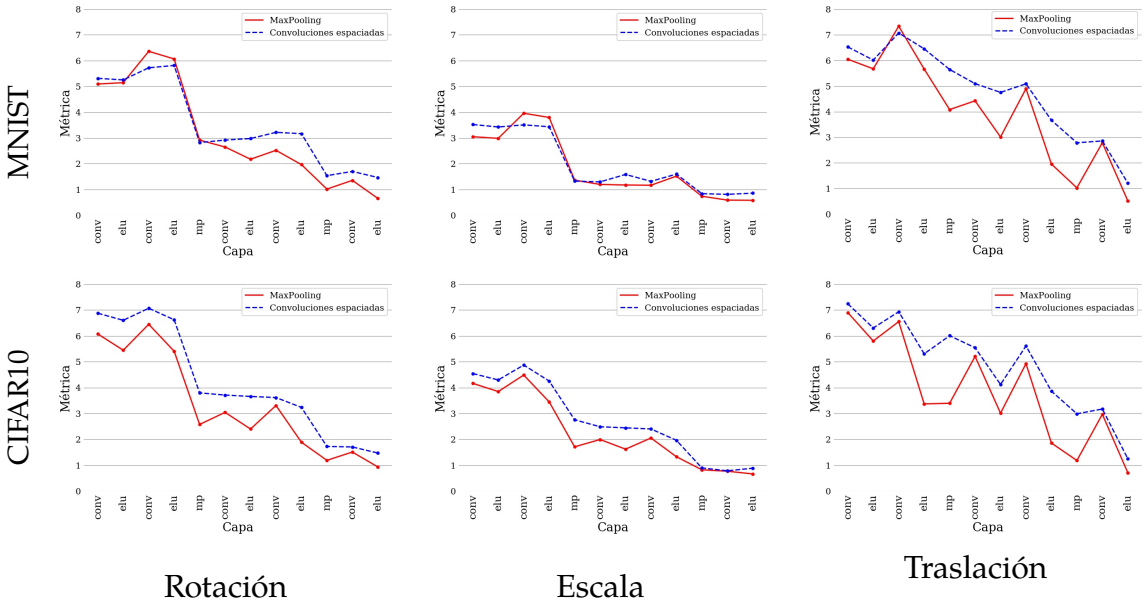


Figura 5.35: Auto-Equivarianza Normalizada de Varianza para SimpleConv con capas MaxPooling o Convoluciones con *paso* = 2

5.5.4. Normalización por Lotes

La Normalización por Lotes (BN, por *Batch Normalization*) [IS15] es una técnica para controlar la media y varianza de las activaciones durante el entrenamiento, en un intento para reducir la cantidad de tiempo necesaria para entrenar la red. Dado que el método busca controlar la varianza de las activaciones, podría influir en la invarianza de la red. Por ende, calculamos el valor de métricas para el modelo SimpleConv con y sin BN. Las versiones con BN de SimpleConv agregan una capa BN después de cada función de activación.

Las Figuras 5.36 y 5.37 muestran los resultados de la Varianza Normalizada y Auto-Equivarianza Normalizada de Varianza para los modelos con capas BN. Podemos observar que las capas BN en sí no afectan al valor de las métricas para dichos modelos.

En las Figuras 5.38 y 5.39 podemos observar la comparación entre modelos con y sin BN. No observamos diferencias significativas entre ambos modelos. No obstante, en el caso de Varianza Normalizada y CIFAR10, encontramos que el pico de Varianza Normalizada de la penúltima capa se suaviza al utilizar BN. Dada la importancia de esta capa para la invarianza, estos resultados sugieren que sería de utilidad investigar el impacto de BN para la invarianza en términos de la tasa de aciertos. No obstante, los resultados inmediatos sugieren al menos que dicha capa no afecta negativamente a la invarianza o auto-equivarianza del modelo.

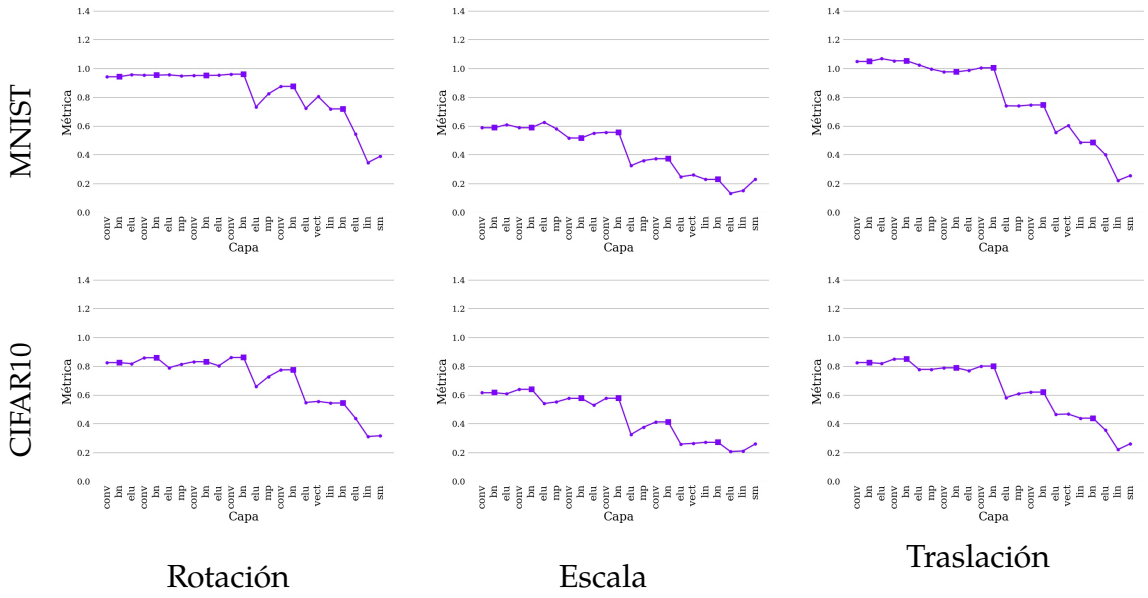


Figura 5.36: Varianza Normalizada de SimpleConv con capas BN. Las capas BN se indican con una cuadrado en lugar de un círculo.

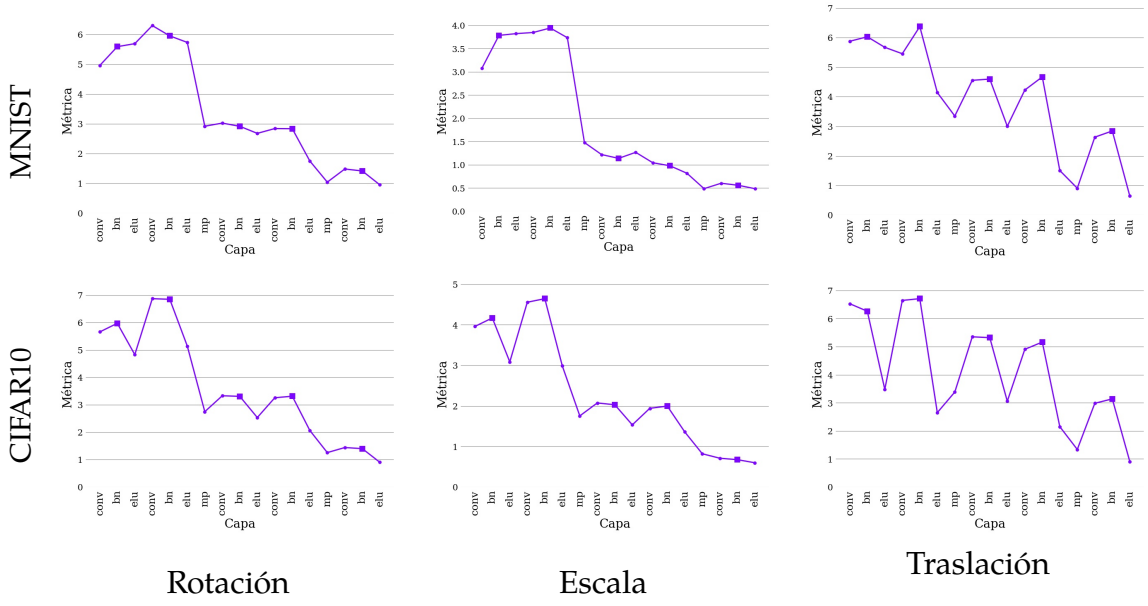


Figura 5.37: Auto-Equivarianza Normalizada de Varianza de SimpleConv con capas BN. Las capas BN se indican con una cuadrado en lugar de un círculo.

5.5.5. Comparación de Arquitecturas de CNN

Dado que la invarianza puede aprenderse mediante aumentación de datos, una hipótesis razonable es que algunas arquitecturas de red pueden impactar en la forma en que la misma adquiere la invarianza. Para probar esta hipótesis, seleccionamos tres modelos adicionales de redes neuronales, con diversidad y facilidad de experimentación como criterios (sección 2.4).

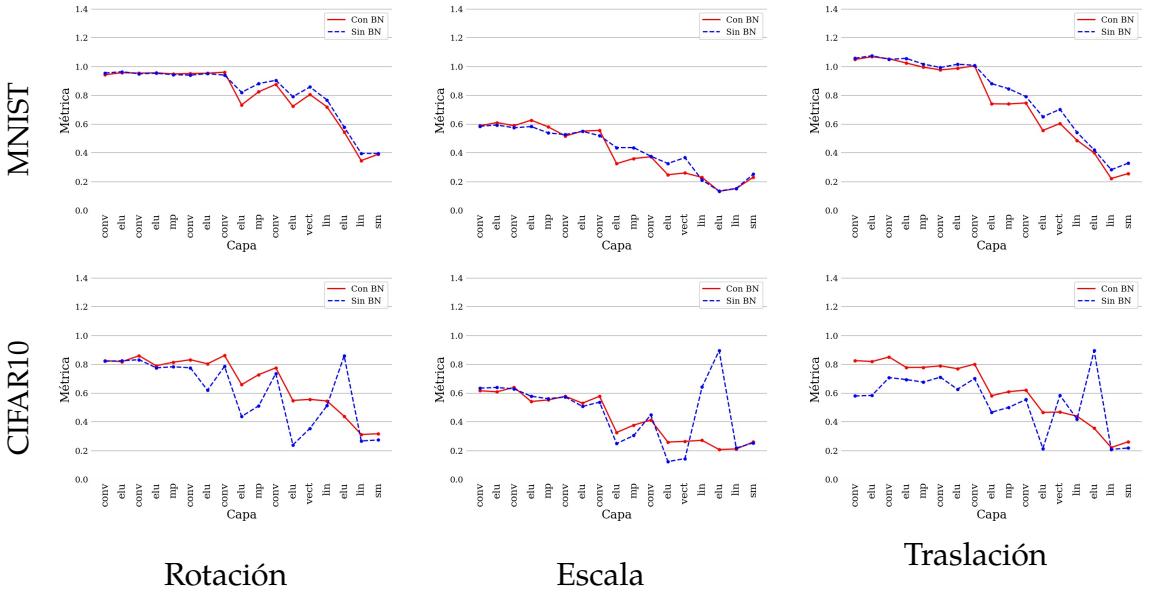


Figura 5.38: Comparación del efecto de BN sobre la Varianza Normalizada para el modelo SimpleConv. Los valores de la métrica para las capas BN se omiten del gráfico para facilitar la comparación de los valores del resto de las capas.

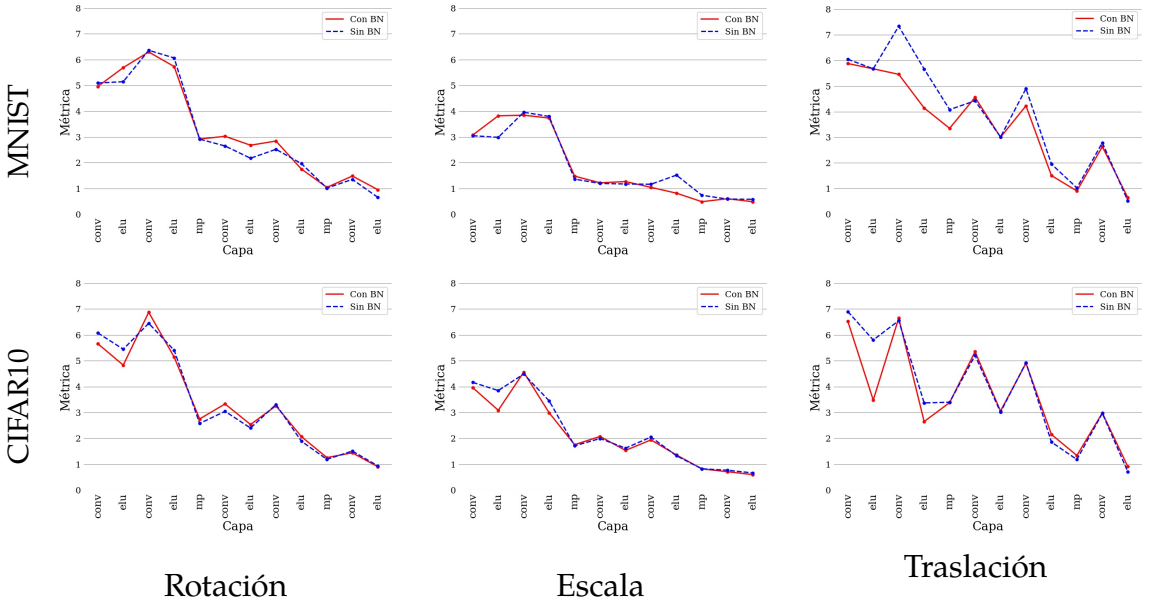


Figura 5.39: Comparación del efecto de BN sobre la Auto-Equivarianza Normalizada de Varianza para el modelo SimpleConv. Los valores de la métrica para las capas BN se omiten del gráfico para facilitar la comparación de los valores del resto de las capas.

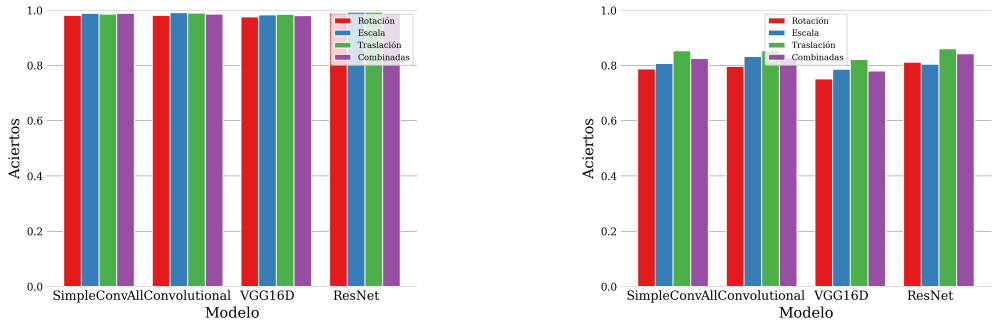
1. **VGG16D** [SZ14], una de las primeras redes con amplio éxito en varias tareas de imágenes. La familia de modelos VGG contiene redes con una gran cantidad de parámetros (VGG16D utiliza 138M de parámetros), y su desempeño por parámetro es mucho menor que otros modelos actuales. No obstante, es

una de las redes más usadas, especialmente como modelo pre-entrenado para transferencia de aprendizaje.

2. **AllConvolutional** [Spr+15], una CNN que no utiliza capas lineales. AllConvolutional descarta la cabeza tradicional de clasificación con capas lineales, y en su lugar utiliza una capa de Global Average Pooling seguida por una Softmax para transformar la salida de la capa convolucional final a probabilidades de clase. Este modelo ofrece la posibilidad de evaluar el impacto de las capas lineales en la equivarianza.

3. **ResNet18** [He+16], es el modelo más simple de la familia ResNet, y fue incluida para evaluar el efecto de las conexiones residuales en la equivarianza. ResNet es el modelo más moderno de los tres. Si bien existen modelos más nuevos y con mejor desempeño que ResNet, lo utilizamos porque es un modelo relativamente simple y con bloques residuales bien conocidos.

Seguimos el mismo protocolo que con el modelo SimpleConv, utilizando la mitad de las características o filtros para MNIST que para CIFAR10, y entregando los modelos hasta su convergencia con el optimizador AdamW. La Figura 5.40 presenta las tasas de acierto de los modelos para cada combinación de transformación/base de datos.



MNIST CIFAR10

Figura 5.40: Tasas de acierto de los cuatro modelos y conjuntos de transformaciones en MNIST y CIFAR10.

La Figura 5.41 muestra el resultado de la Varianza Normalizada para los modelos. Es interesante observar que la *magnitud* de la métrica no difiere significativamente entre los modelos. Además, todos tienen la misma tendencia negativa. A un nivel macro, la figura indica que el modelo más distinto es VGG16D, que tiene valores menores de Varianza Normalizada en general y estos son menos variables. Esto sugiere que la gran cantidad de capas Convolucionales pareciera regularizar el mo-

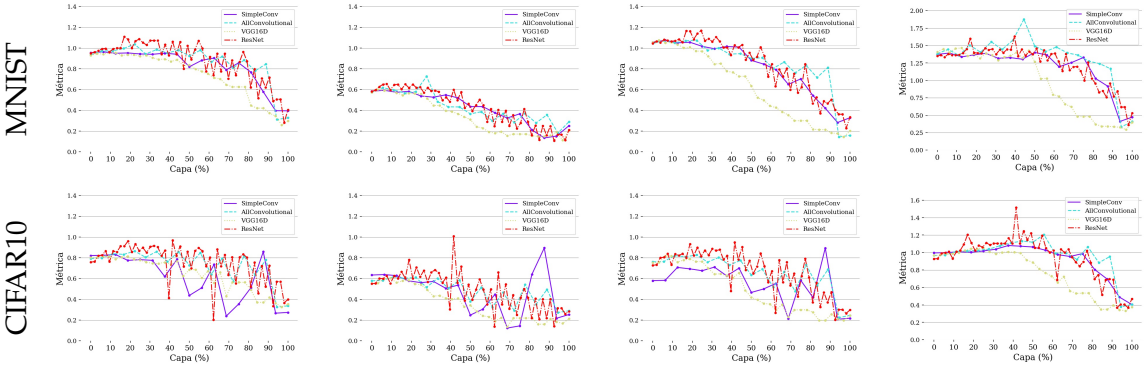


Figura 5.41: VARIANZA NORMALIZADA para varias arquitecturas CNN conocidas. El eje x indica la capa de la red. Dado que cada modelo tiene tipos y cantidades de capas distintas, el eje x no muestra los nombres de las capas. En su lugar, se indica el porcentaje correspondiente al número de capa con respecto al total.

delo en términos de su invarianza. No obstante, su patrón de invarianza es similar al del resto de los modelos.

El modelo SIMPLECONV en CIFAR10 difiere del resto en los picos altos de la penúltima capa lineal, como vimos anteriormente; no obstante, dichos picos son una particularidad de la formulación y desaparecen con cambios ligeros de la arquitectura (secciones 5.5.1 a 5.5.4). También se observan picos *bajos* en algunos bloques residuales de RESNET, aunque luego desaparecen. Estos podrían estar indicando algún tipo particular de codificación en la red que sugiere la necesidad de un estudio adicional.

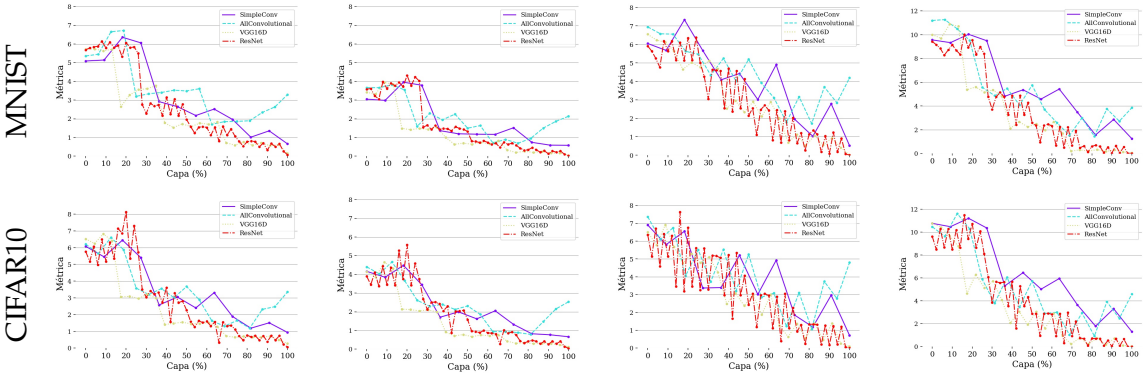


Figura 5.42: AUTO-EQUIVARIANZA NORMALIZADA DE VARIANZA para varias arquitecturas CNN conocidas. El eje x indica la capa de la red. Dado que cada modelo tiene tipos y cantidades de capas distintas, el eje x no muestra los nombres de las capas. En su lugar, se indica el porcentaje correspondiente al número de capa con respecto al total.

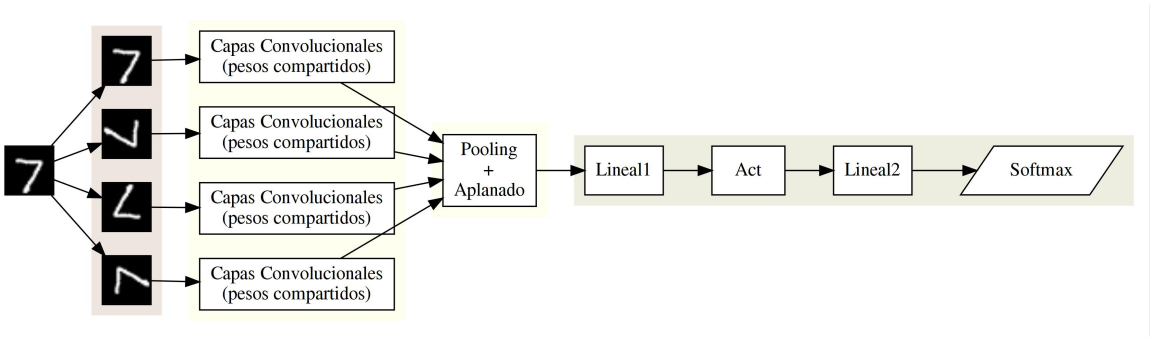


Figura 5.43: Arquitectura de la versión TI Pooling del modelo SimpleConv.

Por otro lado, la Figura 5.42 muestra el resultado de la Auto-Equivarianza Normalizada de Varianza. Al igual que en otros experimentos, notamos que el patrón de la auto-equivarianza medido por la Auto-Equivarianza Normalizada de Varianza para cada modelo es muy estable respecto de la transformación y sobre todo de la BD. No obstante, cada modelo tiene una estructura de auto-equivarianza ligeramente distinta al resto, aunque todos tengan una tendencia negativa.

Notamos que en el caso de AllConvolutional la Auto-Equivarianza Normalizada de Varianza se pudo calcular hasta en su ante-penúltima capa como resultado de estar compuesta sólo por capas Convolucionales. Los valores bajos en la última capa de este modelo sugieren que se produce una disminución radical de la auto-equivarianza ya que estos valores se utilizan para generar los puntajes de clase que deben ser invariantes.

5.5.6. Equivarianza del modelo *Transformation-Invariant Pooling*

La técnica de Agrupamiento Invariante a las Transformaciones (TI Pooling, por Transformation-Invariant Pooling) [Lap+16] es una técnica utilizada para proveer invarianza a las transformaciones de forma similar a la aumentación de datos (sección 2.7). A diferencia de otros métodos [Zha+17; Lua+17; CW16b] TI Pooling puede aplicarse a cualquier tipo de red y conjunto de transformaciones, como en el caso de la aumentación de datos. Esto lo convierte en un modelo ideal para analizar, dado que podemos comparar las diferencias en la representación de las invarianzas de la red.

La aumentación de datos funciona entrenando con un conjunto de ejemplos aumentados por las transformaciones, y luego simplemente evaluando la red para predecir la clase de un nuevo ejemplo. En lugar de esto, TI Pooling Transforma las muestras tanto al entrenar como al calcular la salida de la red, ya que dicho cálculo involucra transformar las muestras. Para adquirir invarianza a un conjunto de trans-

formaciones, basta con que las transformaciones que utiliza la red sean las mismas a la cual se busca la invarianza.

Para aplicar TI Pooling a un modelo (Figura 5.43), debe elegirse una cierta capa como cuello de botella. Entonces, al evaluar un ejemplo, el modelo computa la salida de esa capa de forma independiente para cada transformación. Es decir, aplica cada una de las transformaciones a la entrada, y luego se calcula la salida de la capa usando como entrada cada una de las muestras transformadas. Esto implica una evaluación para cada transformación posible. No obstante, el cálculo de la salida de la capa de cuello de botella utiliza los mismos pesos compartidos para todas las transformaciones, es decir, es una red siamesa. Entonces, la cantidad de parámetros no depende de la cantidad de transformaciones. La cantidad de feature maps, por otro lado, es directamente proporcional a las mismas.

Luego, las características generadas (feature maps, en este caso) se agrupan con una operación de pooling de modo de producir un solo vector característica, de un tamaño independiente al número de transformaciones. Finalmente, la característica agrupada se utiliza por el resto de las capas para producir el resultado final.

En este caso aplicamos el esquema TI Pooling al modelo SimpleConv. Debido a que el modelo SimpleConv con TI Pooling genera más características y luego las agrupa, para que la comparación sea más justa en esta versión del modelo hemos utilizado la mitad de los filtros convolucionales y dimensión de las capas lineales que en el modelo original, para ambas BDs.

La Figura 5.44 muestra la comparación de la Varianza Normalizada entre un modelo SimpleConv, y un modelo SimpleConv al cual se le aplicó la arquitectura TI Pooling. En general, podemos observar un patrón muy similar de la Varianza Normalizada en ambos casos. Esto sugiere que en términos de invarianza no hay ventajas aparentes entre utilizar la arquitectura TI Pooling para el modelo SimpleConv.

En términos de la Auto-Equivarianza Normalizada de Varianza (Figura 5.45), el modelo TI Pooling parece tener ligeramente más auto-equivarianza. Es posible que entonces el esquema de TI Pooling genere presión durante el entrenamiento para obtener características más equivariantes, de modo que el esquema de pooling posterior funcione más adecuadamente.

5.6. Conclusiones

En este capítulo, utilizamos las métricas definidas en el Capítulo 4 para evaluar la invarianza y auto-equivarianza de modelos modernos de CNN. Además, validamos

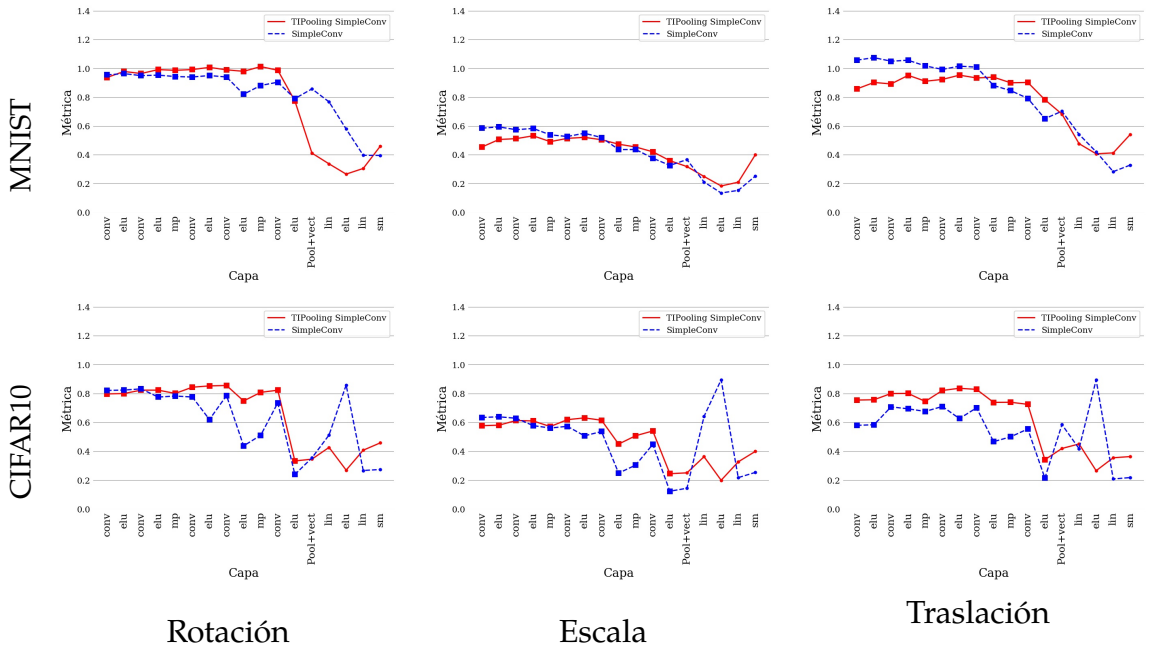


Figura 5.44: Comparación entre aumentación de datos y TI Pooling para el modelo SimpleConv con la métrica Varianza Normalizada. Para las capas siamesas del modelo TI Pooling, el valor de la métrica se ha promediado sobre las distintas transformaciones de modo que pueda compararse con las del modelo original.

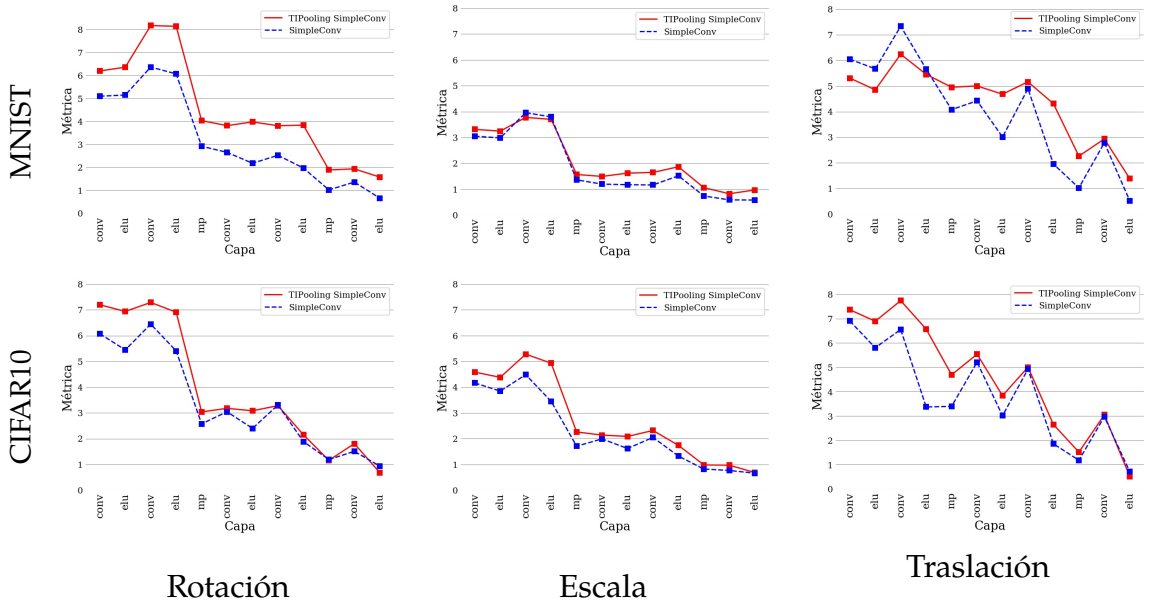


Figura 5.45: Comparación entre aumentación de datos y TI Pooling para el modelo SimpleConv con la métrica Auto-Equivarianza Normalizada de Varianza. Para las capas siamesas del modelo TI Pooling, el valor de la métrica se ha promediado sobre las distintas transformaciones de modo que pueda compararse con las del modelo original.

y analizamos varios aspectos de las métricas.

Para estos experimentos, utilizamos las bien reconocidas bases de datos CIFAR10 y MNIST. Además, utilizamos 4 conjuntos de transformaciones afines importantes para varias aplicaciones de procesamiento de imágenes: rotaciones, escalados, traslaciones y una combinación de estas tres. En la mayoría de los experimentos, utilizamos el modelo , que representa el factor común de las características de los modelos de CNN modernos. La utilización de , junto con la variedad en muestras y transformaciones, permite que los resultados presentados tengan amplia validez.

Primero, validamos las métricas al medirlas en modelos entrenados con y sin aumentación de datos. De esa forma, notamos que las métricas Varianza Normalizada y Distancia Normalizada de hecho miden invarianza, y que æmide autoequivarianza. También comparamos el comportamiento de las métricas con o sin esquemas de normalización, y encontramos que dichos esquemas son necesarios y permiten interpretaciones simples de las métricas.

Además, comparamos las métricas con la métrica de Goodfellow [Goo+09]. En el caso de la métrica ANOVA, encontramos no es apta para detectar invarianza en redes, dado que siempre rechaza la hipótesis nula, declarando como variantes a todas las activaciones. No obstante, las métricas Varianza Normalizada y Distancia Normalizada son más sensibles que la de Goodfellow y reflejan mejor la invarianza de las redes. No comparamos la métrica de Auto-Equivarianza Normalizada de Varianza con otras del mismo tipo, debido a que no existen en la literatura.

En base a los análisis, encontramos que las métricas requieren entre 5000 a 10000 muestras para evaluarse y pueden calcularse con el conjunto de prueba o el de entrenamiento de forma indistinta. Son estables respecto a la inicialización aleatoria de la red, ya que convergen a valores similares sin importar la inicialización.

Confirmando resultados previos de otros autores [Goo+09; LV15], encontramos que la invarianza de las redes entrenadas con aumentación de datos decrementa a medida que se avanza en las capas de una red feedforward. Estudiando el comportamiento con redes aleatorias y durante el entrenamiento, los datos sugieren que la invarianza es una propiedad mayormente aprendida durante el entrenamiento, aunque su estructura final no está determinada por este sino por las transformaciones y muestras. Su estructura y magnitud varían con el tipo y complejidad de transformación, respectivamente, pero no viceversa. Aunque los resultados no son del todo sólidos, la evidencia indica que la invarianza depende tanto de la base de datos utilizada para evaluarla como de la empleada para su entrenamiento.

Además, las funciones de activación *ReLU*, *PReLU* y sobre todo *ELU* favorecen ligeramente la invarianza en las redes. El tamaño de kernel parece no afectar sobremanera a la invarianza, y tampoco hay diferencias significativas entre utilizar Max-

Pooling o capas Convolucionales con $paso = 2$. La normalización por lotes (BN) tampoco parece tener efecto sobre la invarianza, aunque es importante notar que no la perjudica. Finalmente, comparamos varias arquitecturas convolucionales modernas, y todas parecen tener estructuras de invarianza similares, aún aquellas específicamente diseñadas para lidiar con la invarianza.

La auto-equivarianza, al igual que la invarianza, se incrementa al avanzar por las capas de la red. Depende mayormente de la arquitectura de la red y no tanto de los pesos que genera el entrenamiento. Las pruebas evaluando la auto-equivarianza con distintas bases de datos indica que la auto-equivarianza no depende de la base de datos utilizada para evaluar la métrica o la que se utilizó para entrenar el modelo. Por otro lado, tampoco depende del conjunto de transformaciones con el que se entrenó el modelo, pero si del conjunto de transformaciones que se utilice para evaluar la métrica. Aún más, en el caso de las transformaciones, su estructura depende del tipo de transformación, y su magnitud de la complejidad de la misma. Por ende, para la auto-equivarianza la arquitectura y el tipo y complejidad de las transformaciones utilizadas para evaluar son las claves que determinan su estructura y magnitud.

La función de activación y uso de normalización por lotes no parecen tener efecto sobre la auto-equivarianza. Los experimentos con el tamaño del kernel indican que a mayor tamaño, menor es la auto-equivarianza. Esto es esperable ya que la auto-equivarianza se mide en feature maps que generan las capas convolucionales, y éstas son muy sensibles a este parámetro. El uso de MaxPooling en lugar de una capa Convolucional con $paso = 2$ también pareciera aumentar la auto-equivarianza, debido a que no realiza ninguna transformación de su entrada. Por último, la mayoría de los modelos modernos de CNN parece tener una auto-equivarianza similar, salvo por el modelo con TI Pooling, el cual pareciera ser forzado a un nivel mayor de auto-equivarianza con su esquema siamés de transformaciones y pooling.

En resumen, los experimentos realizados nos permiten realizar una caracterización de la invarianza y la equivarianza de los modelos analizados. Dicha caracterización posibilita tener otra perspectiva de los modelos en términos de estas propiedades. Esperamos que éstas conclusiones puedan ser utilizadas para crear mejores modelos equivariantes.

Capítulo 6

Redes Convolucionales para la Clasificación de Formas de Manos

La clasificación de formas de mano es el paso más importante de un sistema de Reconocimiento de Lengua de Señas. Las diversas arquitecturas de CNNs han posibilitado un avance del estado del arte del desempeño de los modelos de clasificación de imágenes. No obstante, no se ha realizado un análisis de la aplicabilidad de dichos modelos al problema del reconocimiento de formas de mano.

En este capítulo presentamos dicho análisis, comparando los modelos SimpleConv, AllConvolutional, VGG, ResNet e Inception (sección 2.4) en las bases de datos LSA16 y RWTH (sección 2.5) [1].

El área de clasificación de formas de mano se ha caracterizado por la búsqueda de esquemas de preprocesamiento y características para mejorar el desempeño de los sistemas [Coo+12]. No obstante, una de las fortalezas de las CNNs es que aprenden sus propias características. Por ende, realizamos una comparación de distintos esquemas de preprocesamiento para determinar si afectan al desempeño de las redes convolucionales, de modo de determinar el pre-procesamiento óptimo.

[1] Estas bases de datos y otras de formas de mano pueden adquirirse fácilmente a través de la librería de código libre https://github.com/midusi/handshape_datasets que desarrollamos para esta tesis.

6.1. Comparación de tasa de aciertos de distintas arquitecturas convolucionales

6.1.1. Metodología

Realizamos experimentos de clasificación con los modelos y bases de datos antes mencionados, midiendo la tasa de acierto [2]. Dado que no hay un conjunto de prueba oficial para estas BDs y su tamaño es relativamente pequeño, promediamos 10 corridas de validación cruzada, utilizando muestreo estratificado aleatorio para estimar la tasa de acierto del conjunto de prueba [Mit97; GBC16]. Además de SimpleConv, utilizamos una red feedforward simple compuesta solo por capas lineales y activaciones ELU como modelo de base, para comparar con las otras arquitecturas. Llamaremos a dicha red SimpleLineal.

En el caso de Inception, utilizamos la red entrenada desde cero, pero también utilizamos una red pre-entrenada en la BD ImageNet [Den+09]. Para la versión pre entrenada, que llamamos Inception+NN, aplicamos transferencia de aprendizaje para re-entrenar el modelo utilizando dos capas lineales como cabeza de clasificación, re-entrenando solo las últimas dos capas y reutilizando todas las capas convolucionales.

Entrenamos todas las redes utilizando una tasa de aprendizaje de 0.001 y el algoritmo de optimización ADAM [KB14], que utiliza un esquema adaptativo para ajustar la tasa de aprendizaje de forma de eliminar efectos de la misma del experimento. Las arquitecturas SimpleLineal y SimpleConv fueron entrenadas por 20 épocas, mientras que las arquitecturas AllConvolutional, ResNet, VGGe Inception fueron entrenadas por 50, 50, 60 y 120 épocas respectivamente, dado que contienen una mayor cantidad de parámetros. Determinamos estos hiperparámetros durante una corrida inicial para asegurar que las redes converjan. No realizamos optimización de hiperparámetros en los subsiguientes experimentos para asegurar una comparación justa y que pueda relacionarse con el desempeño de estos modelos en otros dominios.

En los experimentos de transferencia con la red Inception, las dos capas lineales tienen tamaños 512 y 16 para LSA16 o 30 para RWTH.

Dado que las BDs que son más pequeñas que aquellas en las que se probaron originalmente los modelos, redujimos su capacidad para evitar el sobre-ajuste y permitir que las redes entrenen exitosamente. Para la arquitectura AllConvolutional, decrementamos el tamaño de los filtros convolucionales a $(96, 96, 96, 192, 192, 192)$.

[2]. El código de los experimentos está disponible en: https://github.com/midusi/convolutional_handshape

Para la arquitectura VGGen su versión VGG16D redujimos la cantidad de filtros a $(32, 32, 64, 64, 64, 64, 64, 128, 128, 128, 128, 128, 128)$ y el tamaño de las capas lineales a $(512, 512)$. Utilizamos la versión de 18 capas de ResNet, sin modificación [He+16]. Para Inception, la cantidad de filtros de las capas convolucionales no fue modificado en los experimentos, debido a que utilizamos un modelo pre-entrenado. Por último, la arquitectura SimpleConv [3] utiliza 4 capas convolucionales con $(32, 64, 128, 256)$ filtros de tamaño 3×3, respectivamente, y una capa lineal de 512 elementos. En todos los casos la última capa es una lineal con tantas salidas como clases y la función de activación *Softmax*, excepto por AllConvolutional que utiliza Global Average Pooling.

En todos los modelos encontramos que utilizar una capa de Normalización por Lotes después de cada capa de MaxPooling y reemplazar las funciones ReLU por ELUs (sección 2.2) reduce el tiempo de entrenamiento manteniendo la tasa de aciertos.

Respecto a las BDs, utilizamos la versión pre-segmentada y RGB del conjunto de datos LSA16 [Qui+16a], y las 30 clases con más ejemplos de RWTH [KNB16].

6.1.2. Resultados

En la tabla 6.1 encontramos los resultados para todos los modelos y BDs. Podemos observar que todos los modelos tienen menor tasa de aciertos en RWTH, lo cual es esperable debido a que tiene más clases, desbalance de clases, manos sin segmentar y peor calidad de imagen.

El modelo SimpleLineal tiene la peor tasa de aciertos, lo cual también es esperable dada la ventaja comparativa de las capas convolucionales. Las arquitecturas convolucionales todas tienen un desempeño similar en esta BD.

La tasa de aciertos del modelo DeepHand [KNB16] es mejor que para otros modelos 85.50 %. Debemos notar, no obstante, que DeepHand utiliza una red Inception pre-entrenada en ImageNet, pero adiciona un esquema de entrenamiento débilmente supervisado con 1 millón de imágenes extra cuyas etiquetas son ruidosas. Esto le otorga una ventaja frente a otros modelos. Sacando de lado est modelo, la arquitectura que mejor resultados obtiene en RWTH es VGG16D (82.88 %).

Para LSA16, todas las arquitecturas convolucionales tienen mejor desempeño que ProbSom, un método diseñado específicamente para formas de mano [Qui+16a]. No obstante, el modelo VGG16D nuevamente es el que mejor tasa de aciertos obtiene.

[3] El modelo SimpleConv de este capítulo es distinto que el utilizado en los Capítulos 3 y 5. No obstante, mantuvimos el mismo nombre dado que el espíritu del modelo es el mismo en ambos casos.

Sorprendentemente, la arquitectura SimpleConv logró tasas de reconocimiento para ambos conjuntos de datos relativamente buenas. Esto indica que si bien los modelos más avanzados pueden aprovechar la gran cantidad de ejemplos en otras BDs para obtener buenas tasas de acierto, para estas BDs pequeñas un modelo más chico es más adecuado.

Por último, en este caso vemos que los enfoques de transferencia de aprendizaje no obtienen buen desempeño. Tanto para la red pre-entrada como un generador de características y SVM como clasificador, o al agregar dos capas lineales de clasificación el desempeño es menor que otros métodos, incluyendo SimpleConv. Una posible explicación para este fenómeno es que al utilizar conjuntos de datos pequeños, el re-entrenamiento de las últimas capas no es suficiente para re-adaptar la red ante el nuevo problema.

Tabla 6.1: Tasa de acierto de varias redes neuronales en dos BDs: LSA16 [Qui+16a] y RWTH [KNB16]. Los resultados de los métodos anotados con * fueron tomados de otros artículos.

Modelo	*LSA16*	*RWTH*
DeepHand* [KNB16]	-	**85.50**
Probsom* [Qui+16a]	92.30	-
SimpleLineal	86.58	60.27
SimpleConv [LeC+98]	95.78	81.19
AllConvolutional [Spr+15]	94.56	80.29
VGG16D [SZ14]	**95.92**	**82.88**
ResNet [He+16]	93.49	80.89
Inception [Sze+15a]	91.98	75.33
Inception+SVM [Sze+15a]	93.67	78.12
Inception+NN [Sze+15a]	80.62	75.97

6.2. Comparación de estrategias de preprocesamiento

6.2.1. Metodología

Para determinar si el tipo de pre-procesamiento de las imágenes puede influir en la clasificación, realizamos experimentos utilizando el modelo SimpleConv en LSA16. Elegimos este modelo debido a que es el más simple y posee buen desempeño en esta BD. Utilizamos la misma metodología para los experimentos que antes.

La BD fue elegida debido a que se disponen de varias versiones de la misma (Figura 6.1), que comparamos en términos de la tasa de aciertos:

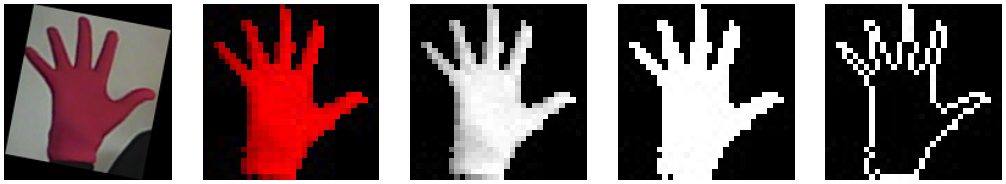


Figura 6.1: Muestras de la base de datos LSA16 en sus distintas versiones. De izquierda a derecha: Imagen original, segmentada, en escala de grises, blanco y negro, máscara de contorno.

1. Imágenes originales recortadas con la mano en el centro
2. Mano segmentada con fondo negro
   a) A color
   b) En escala de grises
   c) En blanco y negro
   d) Sólo máscara de contorno

6.2.2. Resultados

La tabla 6.2 muestra las tasas de acierto obtenidas con los distintos esquemas de pre-procesamiento y el modelo SimpleConv. Del gran incremento en la tasa de acierto al segmentar las imágenes con respecto a las imágenes originales (+12.64) podemos concluir que la segmentación ayuda significativamente a la tarea de clasificación de la mano. El resto de los pre-procesamientos parece influir negativamente en la tasa de aciertos. La pérdida de desempeño entre representar las imágenes segmentadas con RGB, escala de grises, blanco y negro o máscara de contorno nos sugiere que además de la forma general de la mano, el modelo está utilizando la textura de la mano para reconocer su forma, lo cual confirma hallazgos de otros investigadores en la base de datos ImageNet [Gei+19] respecto a la importancia de la textura para los modelos de CNN. En este caso, puede que la información de la textura ayude a distinguir mejor los dedos y los pliegues de las manos.

6.3. Evaluación de aumentación de datos para invarianza

Como mencionamos en la sección 2.9, la misma forma de mano puede aparecer en distintas orientaciones, posiciones o tamaños, en base a la distancia a la cámara, el

Tabla 6.2: Tasa de acierto del modelo SimpleConv en el conjunto de datos LSA16 con distintos esquemas de preprocesamiento.

Pre-procesamiento	Tasa de acierto
Original (RGB)	83.54
Mano segmentada	
RGB	96.18
Escala de grises	87.08
Blanco y Negro	91.38
Máscara de contorno	80.64

ángulo, etc. Por ende, en esta sección combinamos el enfoque de este capítulo con el del Capítulo 3, evaluando la tasa de aciertos de varios modelos de CNN entrenados con distintas transformaciones de los datos, para problemas de clasificación de mano.

6.3.1. Metodología

Para evaluar la capacidad de los modelos para clasificar formas de mano transformadas, los comparamos en los conjuntos de datos RWTH y LSA16 (sección 2.5).

Evaluamos los modelos SimpleConv, AllConvolutional, VGG16D y ResNet18. En estos experimentos, utilizamos una configuración de entrenamiento similar al de los Capítulos 3 y 5, donde entrenamos los modelos utilizando aumentación de datos con varios conjuntos de transformaciones.

Al igual que en el Capítulo 3, primero entrenamos cada modelo sin aumentación de datos. Luego, por cada conjunto de transformaciones, entrenamos los modelos con aumentación de datos a esas transformaciones. Finalmente, los evaluamos a todos utilizando datos sin transformar y datos transformados.

Además, en todos los casos utilizamos el optimizador AdamW [LH19], una tasa de aprendizaje de 0.001, y elegimos una cantidad de iteraciones base para cada modelo de forma que siempre sea suficiente para que converja. Para los modelos entrenados con aumentación de datos dicha cantidad de iteraciones base se multiplica por $log(m)$, donde m es el tamaño del conjunto de transformaciones. Dado que nuestro objetivo es obtener una idea del comportamiento general de los modelos ante las transformaciones y no nos interesa obtener un desempeño del estado del arte, buscamos un conjunto de hiperparámetros que funcionen bien para todo el conjunto de modelos entrenados. En este caso, buscamos que la tasa de aciertos mínima sea de 0.5 para RWTH y 0.6 para LSA16.

Utilizamos cuatro conjuntos de transformaciones como en el Capítulo 5: rotaciones, escalados, traslaciones y una combinación de las anteriores. Dado que en estos

conjuntos de datos el costo computacional es menor, utilizamos un conjunto de transformaciones más amplio que en el Capítulo 5, de modo de generar una comparación mas exhaustiva. Dado que las manos no se encuentran centradas en RWTH, no pudimos utilizar traslaciones más intensas.

1. **Rotación** (16 transformaciones). Rotaciones desde 0° a 360°, discretizadas en 16 ángulos. Las rotaciones se realizan siempre utilizando el centro de la imagen como origen.

2. **Escalado** (13 transformaciones). Escalamos las imágenes con los coeficientes de escala: 0.40, 0.50, ...1.0, 1.05, ..., 1.30.

3. **Traslaciones** (48 transformaciones): Generamos todas las traslaciones de $d = 2^i$ píxeles con el siguiente esquema: $(-d,-d)$, $(-d,d)$, $(d,-d)$, (d,d), $(0,d)$, $(d,0)$, $(0,-d)$, $(-d,0)$. Los valores de i utilizados son $0,1,2,3,4,5$, para un total de $8 \times 6 = 48$ transformaciones.

4. **Combinadas** (9984 transformaciones): Generamos todas las posibles combinaciones de las transformaciones anteriores, para un total de $16 \times 13 \times 48 = 9984$ transformaciones.

La Figura 6.2 muestra ejemplos de ambas bases de datos a lo cuales se les aplican estas transformaciones.

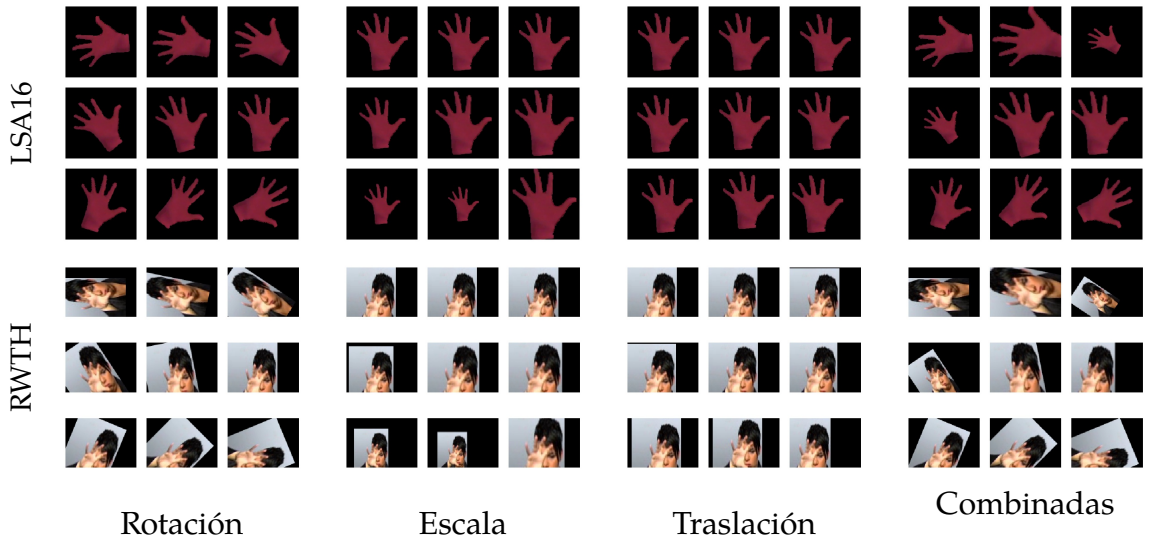


Figura 6.2: Muestras transformadas de las bases de datos LSA16 y RWTH.

Para evaluar la tasa de aciertos, repetimos el esquema del Capítulo 3, donde evaluamos la tasa utilizando el conjunto de prueba, y donde cada ejemplo del conjunto

de prueba es transformado por una de las transformaciones elegida de forma aleatoria. De este modo, si el conjunto de prueba tiene n ejemplos, entonces se evalúa al modelo con n ejemplos, donde cada ejemplo es evaluado con una transformación aleatoria (posiblemente) distinta.

6.3.2. Resultados

La Figura 6.3 muestra los resultados de este experimento. Dado que no hay una optimización particular para cada combinación de base de datos y transformación, las tasas de acierto obtenidas son menores a las de la sección 6.1.

En este dominio podemos observar también como los modelos entrenados sin aumentación de datos tienen tasas de acierto mucho menores al ser evaluados en conjuntos de datos transformados. Este efecto es mayor para las rotaciones, y menor para otras transformaciones, lo cual parece proporcional a la intensidad de las transformaciones (Figura 6.2). En todos los casos podemos observar que la aumentación de datos recupera el desempeño de los modelos normales tanto en el conjunto de prueba normal como en el transformado. Notablemente, este resultado también se aplica al caso de las transformaciones combinadas, que representan un conjunto de transformaciones muy complejas.

En estas bases de datos los modelos entrenados con aumentación de datos obtienen mayor desempeño que aquellos con un entrenamiento normal. Este efecto no fue observado en la sección 3.2 al comparar estos modelos en MNIST y CIFAR10. Esto puede deberse a que la cantidad de ejemplos de de LSA16 y RWTH es significativamente menor respecto de MNIST y CIFAR10, y por ende la aumentación de datos compensa la falta de ejemplos de entrenamiento. Por otro lado, puede ser también debido a que las bases de datos de formas de mano contienen naturalmente algunos ejemplares transformados, pero solo para algunas muestras, con lo cual la aumentación de datos trae conocimiento apriori que permite clasificar ejemplos difíciles.

6.4. Conclusiones

La clasificación de formas de mano en imágenes es el subproblema más importante del Reconocimiento de Señas en Videos.

En este capítulo, evaluamos distintos modelos de CNN modernos en este problema, con el objetivo de determinar el más apto. Los resultados indican que si bien todos los modelos basados en CNN tienen un buen desempeño, el modelo VGG16D obtiene los mejores resultados, aún al compararlo con modelos pre-entrenados o

desarrollados específicamente para este problema. Su desempeño ligeramente inferior al de DeepHand [KNB16] en RWTH, un modelo basado en Inception, aunque dicho modelo utiliza un esquema de entrenamiento más avanzado con datos extra.

También evaluamos los distintos esquemas de preprocesamiento con un modelo convolucional simple y la base de datos LSA16. Dicha base de datos cuenta con varios tipos de pre-procesamiento. En particular, cuenta con una versión de la BD donde el fondo está segmentado. Los resultados sugieren que la segmentación de la mano en la imagen es una etapa de preprocesamiento crucial que aumenta significativamente el desempeño de las CNNs.

Por último, evaluamos la aumentación de datos como técnica para otorgarle invarianza a las redes y así poder clasificar imágenes de formas de mano transformadas. Encontramos que la aumentación de datos permite entrenar modelos invariantes que pueden clasificarlas con una tasa de acierto similar a la de su contraparte no invariante. Aún más, en algunos casos los modelos invariantes obtienen una tasa de acierto superior a la de los modelos originales, posiblemente debido a la presencia de ejemplares transformados en el conjunto de prueba original, es decir, que el dominio naturalmente presenta objetos transformados.

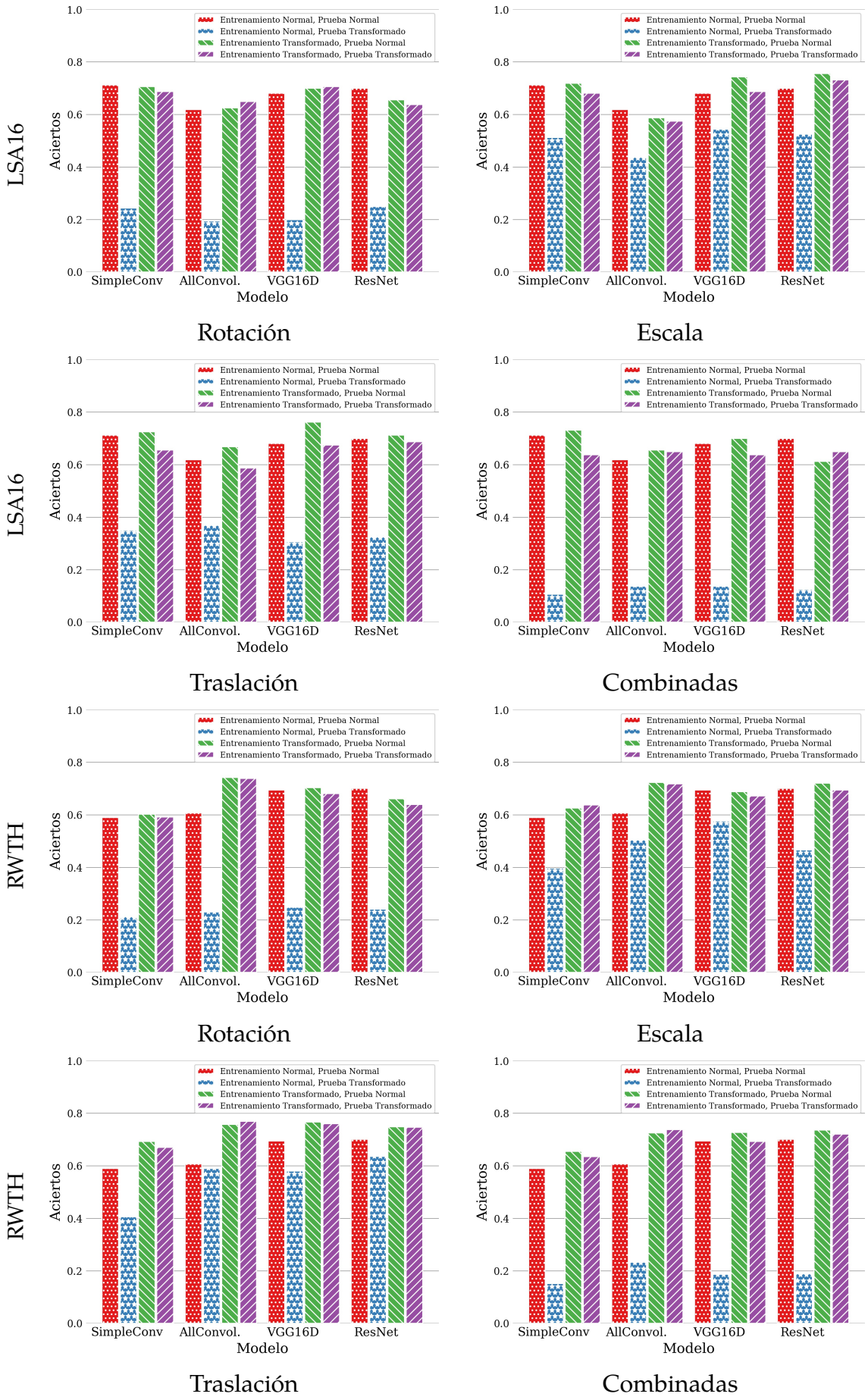


Figura 6.3: Tasas de acierto para varios tipos de transformaciones de aumentación de datos en las bases de datos LSA16 y RWTH.

Capítulo 7

Conclusiones y trabajos futuros

Los modelos basados en Redes Neuronales actualmente tienen el estado del arte en términos de desempeño en la mayoría de los campos del Aprendizaje Automático. En particular, son muy eficaces para lidiar con problemas de clasificación de imágenes al utilizar capas Convolucionales para crear Redes Neuronales Convolucionales.

El Reconocimiento Automático de Lengua de Señas en vídeos es un problema de visión de computadoras con varias aplicaciones que pueden mejorar la interacción entre la comunidades sordas y hablantes. Un paso crucial para generar sistemas robustos de Reconocimiento de Lengua de Señas es la clasificación de formas de mano en imágenes. Este problema requiere que los clasificadores puedan reconocer las formas de mano aún cuando están sujetas a transformaciones afines como la rotación, escalado y traslación. Es decir, requiere que los modelos sean *invariantes* a estas transformaciones.

La propiedad de invarianza es un caso especial de la *equivarianza*, que nos permite determinar como un modelo se comporta ante la transformación de su entrada. La *auto-equivarianza*, otro caso especial de la equivarianza, permite determinar si ante una transformación de la entrada las representaciones internas de la red varían de la misma forma.

Los modelos basados en Redes Neuronales Convolucionales tradicionales no poseen equivarianzas naturalmente. Por ende, existen dos maneras de obtener equivarianzas en estos modelos: dotar al modelo con características especiales para la equivarianza, o aumentar los datos mediante transformaciones. No obstante, en ambos casos se conoce muy poco sobre como los modelos realmente adquieren la equivarianza, especialmente en el caso de la aumentación de datos.

El objetivo de esta tesis es evaluar los métodos para otorgar equivarianza a los modelos de Redes Neuronales Convolucionales, y comprender los mecanismos me-

diante los cuales estos modelos adquieren la equivarianza. Un objetivo secundario es transferir el análisis y las técnicas desarrolladas para mejorar los modelos de clasificación de formas de mano. A continuación detallamos los logros obtenidos en base a estos objetivos.

7.1. Logros

Comparación de aumentación de datos y modelos especializados para la invarianza a las rotaciones La invarianza rotacional es una propiedad deseable para varias aplicaciones en el campo de la clasificación de imágenes, y puede adquirirse con aumentación de datos o modelos especializados. En el Capítulo 3 comparamos estas dos estrategias para generar redes invariantes. En base a dicha comparación, entendemos que si bien la aumentación de datos puede requerir más tiempo de entrenamiento, puede obtener un desempeño similar al de modelos especializados manteniendo la simplicidad de evaluación. Además, determinamos las capas más importantes a re-entrenar un modelo pre-entrenado para adquirir invarianza de forma eficiente. Aún más, confirmamos la noción de que las primeras capas convolucionales de una red aprenden un conjunto de filtros redundantes que pueden ser reutilizados para otra tareas, aún en el contexto de la invarianza.

Métricas de invarianza y auto-equivarianza En el Capítulo 4 presentamos una serie de métricas flexibles, eficientes e interpretables para cuantificar la invarianza y auto-equivarianza de cualquier modelo basado en Redes Neuronales. Nuestras métricas son completamente adaptables en términos de la arquitectura de la red, tipos de capas, conjuntos de transformaciones y formatos de entrada. Las métricas tienen una granularidad alta, permitiendo medir éstas propiedades en cada activación de la red de forma individual, y luego combinar dichos resultados para evaluar estructuras de más alto nivel como capas o redes enteras. También introdujimos variantes de las métricas para tipos de activación específicos como los feature maps, y para analizar el componente de clase de la equivarianza para problemas de clasificación. Estas métricas eficientes permiten el análisis de modelos modernos de redes neuronales en términos de la equivarianza. Se distinguen de otros esfuerzos similares en que son fáciles de interpretar y eficientes para calcular, especialmente en el contexto de redes grandes con millones de activaciones. Además, desarrollamos una librería de código abierto para la librería PyTorch que provee implementaciones de referencia de las métricas desarrolladas y simplifican el desarrollo de nuevas.

Validación y análisis de las métricas. Caracterización de modelos de CNN en base a la invarianza y auto-equivarianza En el Capítulo 5 realizamos varios experimentos y visualizaciones para comprender la naturaleza de las métricas propuestas en el Capítulo 4, y validar su utilidad. Luego utilizamos éstas métricas para evaluar modelos de CNN y características de los mismos. Para estos experimentos, empleamos las bien reconocidas bases de datos CIFAR10 y MNIST, y los modelos ResNet, VGGy AllConvolutional. Además, utilizamos 4 conjuntos de transformaciones afines importantes para varias aplicaciones de procesamiento de imágenes: rotaciones, escalados, traslaciones y una combinación de estas tres. En la mayoría de los experimentos utilizamos el modelo SimpleConv, que representa el factor común de las características de los modelos de CNN modernos. La utilización de SimpleConv, junto con la variedad en muestras y transformaciones, permite que los resultados presentados tengan amplia validez. Los experimentos realizados nos permiten realizar una caracterización de la invarianza y la equivarianza de los modelos y sus particularidades. Dicha caracterización posibilita tener otra perspectiva de los modelos en términos de estas propiedades. Esperamos que estas conclusiones puedan ser utilizadas para crear mejores modelos equivariantes. Creemos que las comparaciones con aumentación de datos y las métricas presentadas forman en conjunto una metodología poderosa para evaluar la equivarianza de los modelos y así comprenderlos más completamente.

Modelos de Redes Convolucionales para la clasificación de formas de mano En el Capítulo 6evaluamos distintos modelos de CNN modernos para clasificar formas de mano, con el objetivo de determinar el más apto. Los resultados indican que si bien todos los modelos basados en CNN tienen un buen desempeño, el modelo VGG16D obtiene los mejores resultados, aún al compararlo con modelos pre-entrenados o desarrollados específicamente para este problema. Su desempeño ligeramente inferior al de DeepHand [KNB16] en RWTH, un modelo basado en Inception, aunque dicho modelo utiliza un esquema de entrenamiento más avanzado con datos extra. También evaluamos los distintos esquemas de preprocesamiento con un modelo convolucional simple y la base de datos LSA16. Dicha base de datos cuenta con varios tipos de pre-procesamiento. En particular, cuenta con una versión de la base de datos donde el fondo está segmentado. Los resultados sugieren que la segmentación de la mano en la imagen es una etapa de preprocesamiento crucial que aumenta significativamente el desempeño de las CNNs.

7.2. Trabajos Futuros

Creemos que es posible aprender más acerca de las Redes Neuronales y Convolucionales estudiando sus equivarianzas, y así mejorar los modelos existentes para hacer posible nuevas aplicaciones.

Para avanzar en esta dirección, identificamos varias áreas de trabajo posibles.

Con respecto a la adquisición de invarianza mediante aumentación de datos, en principio vemos la necesidad de expandir los dominios de evaluación, considerando otros problemas de clasificación, así como problemas de segmentación, localización y regresión. También deberían considerarse problemas en los cuales las muestras aparecen naturalmente transformadas. Además, resultaría interesante caracterizar la relación entre la complejidad de las transformaciones a las cuales se busca ser invariante y la complejidad del modelo, medida en términos de la cantidad de parámetros que tiene y tiempo de ejecución, así como del tiempo de cómputo extra necesario para aprender la invarianza. Por último, se requiere avanzar en las técnicas de transferencia de aprendizaje para adquirir invarianza, ya que en varios dominios la transferencia de aprendizaje es la técnica más directa para obtener modelos con un buen desempeño.

Con respecto a las métricas de equivarianza propuestas, sería interesante otorgarles la capacidad de detectar automáticamente estructuras de invarianza o autoequivarianza en la red en términos de grupos de activaciones. De este modo, se posibilitaría la realización de análisis a una granularidad intermedia entre las activaciones individuales o las capas de la red. Además, sería útil expandir las métricas al caso de la equivarianza, sin necesidad de un algoritmo de optimización intermedio como el de [LV15]. También es necesario realizar una caracterización estadística de las métricas de modo que se puedan realizar pruebas de hipótesis para, por ejemplo, comparar dos modelos en términos de la invarianza. En dicha caracterización se debería incluir un análisis de los Ataques y Defensas Adversariales en términos de la invarianza de los modelos. Finalmente, tenemos planeado implementar soporte para Tensorflow y Tensorboard en la librería de métricas transformacionales que desarrollamos para esta tesis de modo que toda la comunidad de investigadores pueda hacer uso de estas técnicas. También continuaremos con el desarrollo de la misma para mejorar su desempeño y proveer más funcionalidades, como el filtrado de capas y el pre-procesamiento de activaciones.

Con respecto a la caracterización de los modelos, creemos que es necesario profundizar en este aspecto analizando la dependencia entre la cantidad de filtros o características y la equivarianza adquirida. Al igual que con los experimentos de au-

mentación de datos, sin duda sería de utilidad comparar la equivarianza de modelos entrenados desde cero con modelos que han sido entrenados mediante transferencia de aprendizaje y de esta forma complementar la teoría de este último campo. También nos interesa ampliar el abanico de transformaciones y problemas con los cuales aplicar la métrica, fuera de las transformaciones afines y la clasificación de imágenes.

En base a los argumentos presentados, creemos que se puede aprender más acerca de las CNNs estudiando sus invarianzas y equivarianzas.

Apéndice A

Pseudocódigo del cómputo de las métricas

En este apéndice presentamos el pseudocódigo de algunos algoritmos presentados en el Capítulo 4.

El cálculo online de la matriz **MT** para una red con varias activaciones se presenta en el Listado A.1.

Los algoritmos Listados A.2 a A.4 muestran un pseudocódigo del cálculo de las métricas Varianza Normalizada, Varianza Muestral y Varianza Transformacional, respectivamente, para una sola activación a.

```
import numpy as np
def generar_matriz_mt(x,t,a):
  # x: Conjunto de ejemplos [x[1],...,x[n]]
  # t: Conjunto de transformaciones [t[1],...,t[m]]
  # a: Funciones de activación de una red neuronal [a[1],...,a[k]].
  # retorna: Elementos de las matrices MT de a[1],...,a[k].
  n,m,k = len(x),len(t),len(a)
  for i in range(n):
    for j in range(m):
      mt_ij = np.zeros(k)
      for l in range(k):
         mt_ij[l]$ = a[l](t[j](x[i]))
      yield mt_ij
```

Algoritmo A.1: Función `GenerarMatrizMT` para el cálculo de la matriz **MT** de forma online. La operación *yield* genera un valor para consumir, de forma que el recorrido pueda realizarse con un uso de memoria constante $O(k)$. Cada *yield* genera el elemento i, j de la matriz **MT** para todas las activaciones. En este caso, el recorrido se realiza por filas, cambiando primero las transformaciones (columnas).

```
def varianza_normalizada(x,t,a):
    return varianza_transformacional(x,t,a)/varianza_muestral(x,t,a)
```

Algoritmo A.2: Función que calcula la métrica Varianza Normalizada para una sola activación a.

```
def varianza_transformacional(x,t,a):
    # x: Conjunto de ejemplos [x[1],...,x[n]]
    # t: Conjunto de transformaciones [t[1],...,t[m]]
    # a: Función de activación de una red neuronal.
    n,m   = len(x),len(t)
    mean  = 0
    count = 0
    for i in range(n):
        #Varianza para el ejemplo i
        count_i = 0
        mean_i  = 0
        var_i   = 0
        for j in range(m):
            v = a(t[j](x[i]))
            #Actualizar varianza para la transformación t
            #con el algoritmo de Welford
            count_i  += 1
            delta_i   = v-mean_i
            mean_i   += delta_i/count_i
            deltav_i  = v - mean_i
            var_i    += delta_i * deltav_i
        #Actualizar la media de la varianza con var_i
        count += 1
        mean  += (var_i-mean) / count
    return mean
```

Algoritmo A.3: Función que calcula la métrica Varianza Transformacional para una sola activación a.

```
def varianza_muestral(x,t,a):
    # x: Conjunto de ejemplos [x[1],...,x[n]]
    # t: Conjunto de transformaciones [t[1],...,t[m]]
    # a: Función de activación de una red neuronal.
    n,m   = len(x),len(t)
    mean  = 0
    count = 0
    for j in range(m):
        #Varianza para el ejemplo i
        count_j = 0
        mean_j  = 0
        var_j   = 0
        for i in range(n):
```

```
            v = a(t[j](x[i]))
            #Actualizar varianza para la transformación t
            #con el algoritmo de Welford
            count_j  += 1
            delta_j   = v-mean_j
            mean_j   += delta_j/count_j
            deltav_j  = v - mean_j
            var_j    += delta_j * deltav_j
        #Actualizar la media de la varianza con var_j
        count += 1
        mean  += (var_j-mean) / count
    return mean
```

Algoritmo A.4: Función que calcula la métrica Varianza Muestral para una sola activación a.

Apéndice B

Diseño e implementación de la librería de Medidas Transformacionales

Para facilitar la implementación de las métricas, hemos desarrollado una librería de código abierto llamada *Transformational Measures*[1] que facilita el recorrido de las matrices **MT**. Además provee implementaciones de todas las métricas definidas en esta tesis, además de la métrica de Goodfellow [Goo+09].

La clase central de la librería es el `ActivationsIterator` (Figura B.1), que en base a un conjunto de datos, de transformaciones, y un modelo que genera activaciones,

La definición de las métricas se realiza en Python utilizando la librería Numpy. Por el contrario, las implementaciones de la clase `ActivationsIterator` son específicas de cada framework específico como PyTorch, Tensorflow u otros. De esta

[1]Sitio web: https://github.com/facundoq/transformational_measures.

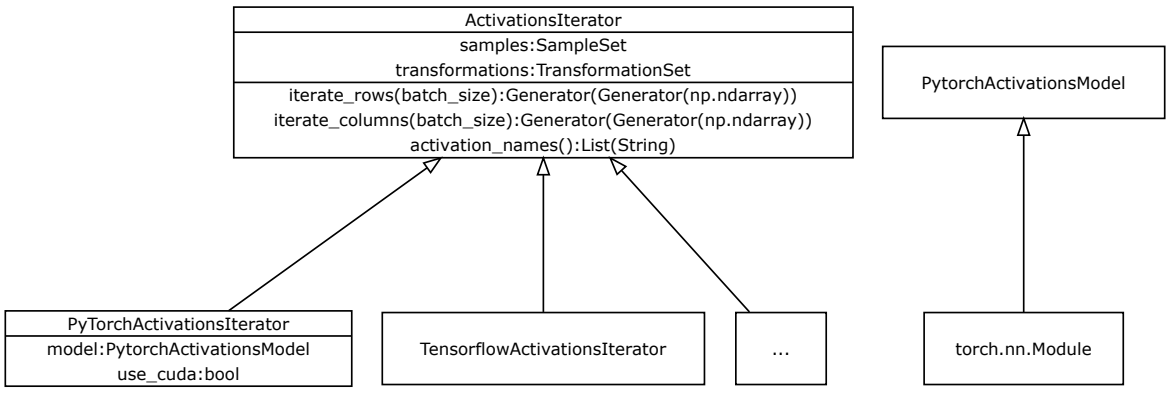


Figura B.1: Diagrama de clases de los iteradores de activaciones. La interfaz `ActivationsIterator` define los métodos básicos para iterar sobre las matrices **MT** de forma eficiente, tanto por filas como por columnas.

```
def eval(self, activations_iterator: ActivationsIterator) ->
    MeasureResult:
        layer_names = activations_iterator.layer_names()
        result = ...
        for t, samples_activations_iterator in activations_iterator.
    transformations_first():

            for batch, batch_activations in samples_activations_iterator:
                for j, layer_activations in enumerate(batch_activations):
                    #layer_activations = activaciones de la capa j del
    modelo para
                    # la transformación "t" y las muestras del lote "
    batch"
                    # Codificado como un vector de Numpy (np.ndarray)
                    result = ...
        ...
     return MeasureResult(result, layer_names, self)
```

Algoritmo B.1: Iteración por columnas de las matrices **MT**.

forma, la implementación puede optimizarse para cada framework específico.

No obstante, los iteradores devuelven las activaciones en arreglos de Numpy, de modo que la implementación de las métricas es agnóstica al framework utilizado(Listado B.1)

Las Figuras B.2 y B.3 muestran las familias de métricas de invarianza y auto-equivarianza, respectivamente. Cada métrica corresponde a una clase. Las métricas muestras y transformacionales pueden calcularse de forma independiente. Las métricas normalizadas utilizan estas clases para calcularse.

Para el cálculo de la auto-equivarianza, se definen también dos iteradores específicos que transforman las activaciones antes de retornarlas.

En el caso de las métricas de auto-equivarianza normalizadas (Auto-Equivarianza Normalizada de Distancia, Auto-Equivarianza Normalizada de Varianza y similares), es necesario iterar sobre la matriz $\mathbf{MT}'$ (sección 4.6), donde las activaciones han sido invertidas, es decir, se les ha aplicado la transformación inversa a la transformación que sufrió la entrada x. La clase `InvertedActivationsPyTorchActivationsIterator` permite recorrer la matriz $\mathbf{MT}'$, con los valores $t_j^{-1} f(t_j(x_i))$ directamente (Figura B.4). De la misma forma, la clase `TransformedActivationsPyTorchActivationsIterator` posibilita iterar sobre activaciones que han sido transformadas de acuerdo a lo necesario para calcular la métrica Auto-Equivarianza de Distancia Simple, de forma de poder comparar $f(t(x))$ con $t(f(x))$ de forma simplificada, y cualquier otra métrica futura que se base en estas cantidades.

Por último, la interfaz del `ActivationsIterator` mezcla las iteraciones de cada capa de las matrices **MT**, complicando el cálculo de las métricas (Listado A.1,

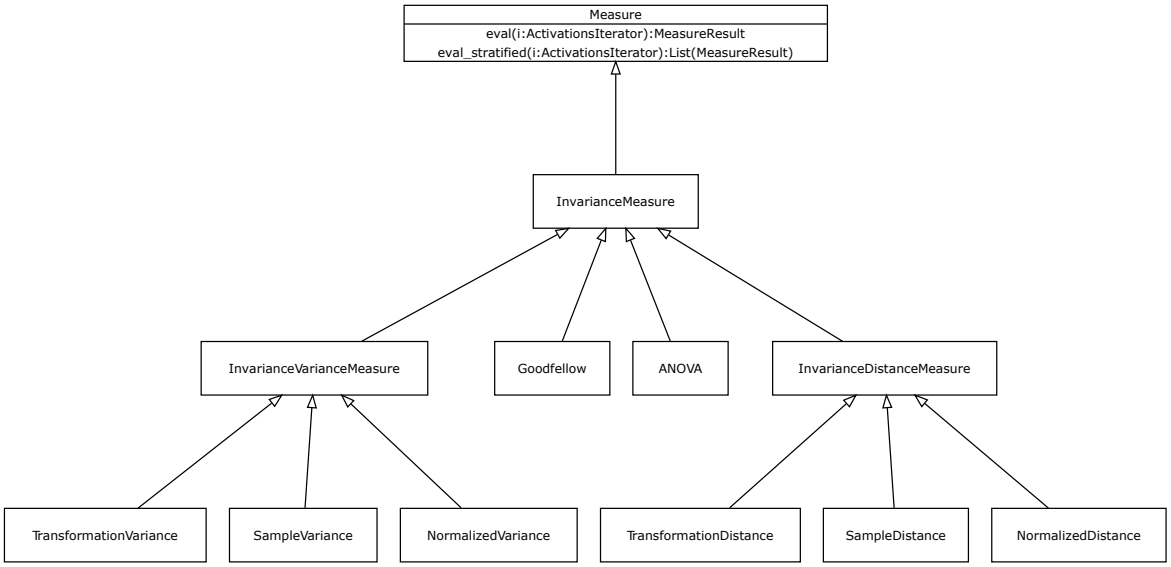


Figura B.2: Diagrama de clases de las métricas de invarianza

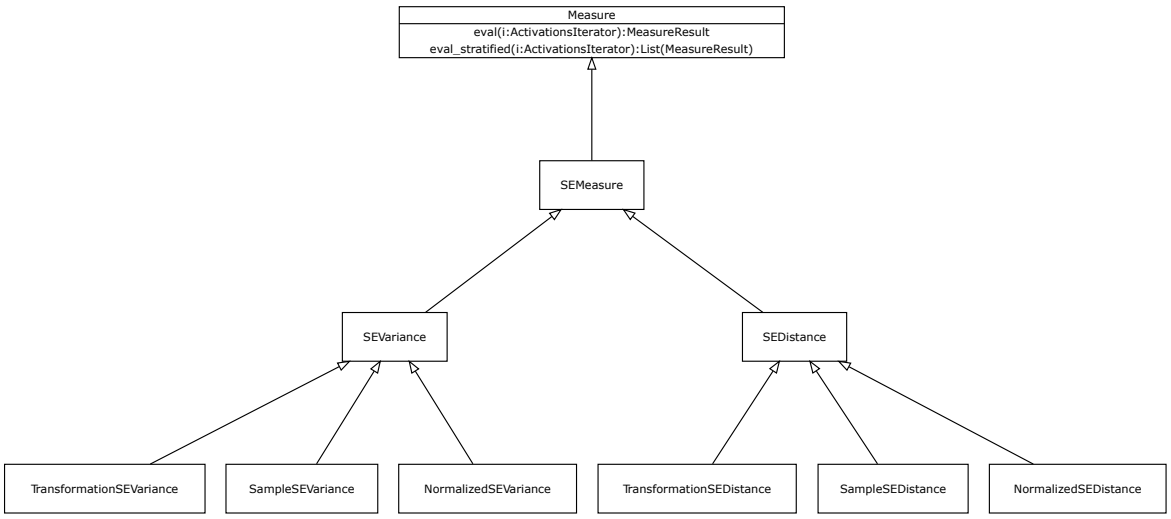


Figura B.3: Diagrama de clases de las métricas de auto-equivarianza

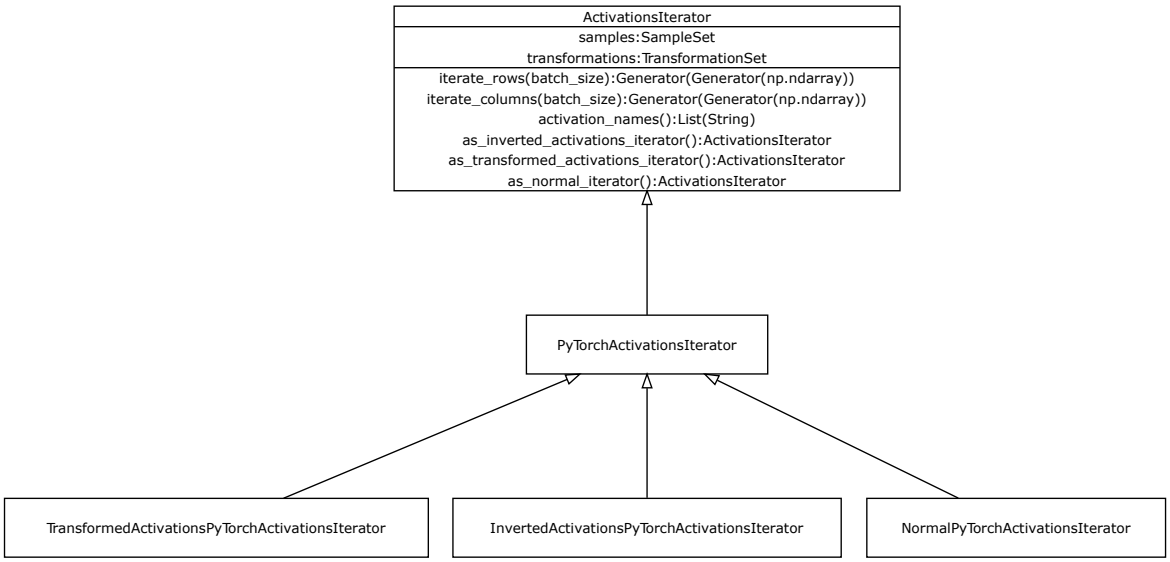


Figura B.4: Diagrama de clases de los iteradores de activaciones especiales para implementar las métricas de auto-equivarianza. Las siglas *SE* se refieren a *Same-Equivariance* (auto-equivarianza en inglés).

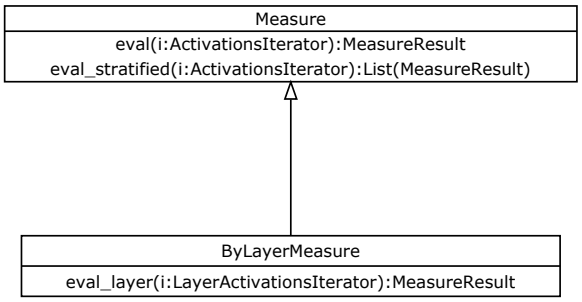


Figura B.5: La clase `ByLayerMeasure` permite hacer el cálculo de la métrica para una sola capa, automatizando la replicación en cada capa y con paralelismo a nivel de procesos.

línea 7).

No obstante, la librería ofrece un grado mayor de automatización para el calculo de métricas en donde el valor de cada capa es independiente de las otras. En este caso, podemos simplemente utilizar la clase `ByLayerMeasure` (Figura B.5) que permite implementar el cómputo para una capa de forma genérica, y aplicar este cálculo a cada capa por separado. Además, dicha clase paraleliza el cómputo de las métricas utilizando un proceso por capa para aumentar el desempeño del cálculo de la métrica.

Apéndice C

Varianza de funciones de activación

Para poder caracterizar la invarianza de los modelos, resulta de utilidad comprender como las funciones de activación afectan a la varianza de las activaciones. Si bien existen varios análisis de la varianza de las funciones de activación, los mismos se han desarrollado en términos de la salida de la capa que alimenta a la función de activación o de la red entera [Kla+17].

Dado que las métricas propuestas son agnósticas a los tipos de capas y la arquitectura de red, debemos realizar la caracterización sólo en términos de la función de activación, ignorando la distribución de su entrada. Estudiamos la varianza de funciones de activaciones comúnmente utilizadas como funciones independientes de la red.

La media E y varianza V de una función continua f de una variable aleatoria real x se definen como:

$$\begin{aligned} E(f(x)) &= \int_{-\infty}^{\infty} f(x)p(x)dx \\ V(f(x)) &= \int_{-\infty}^{\infty} (E(f(x)) - f(x))^2)p(x)dx \\ &= E(f(x)^2) - E(f(x))^2 \end{aligned} \tag{C.1}$$

A continuación, probamos que $V(f(x)) \leq V(x)$ para varias funciones de activación $f(x)$. Esto indica que la varianza de estas funciones es no-creciente con respecto a la varianza de su entrada. En la práctica, esto implica que estas funciones tienden a bajar la varianza de la red. Las condiciones para establecer los casos en que estrictamente $V(f(x)) < V(x)$ dependen de $p(x)$ y no pueden especificarse sin caracterizar la salida de la capa anterior. No obstante, los resultados de la sección 5.5.1 proveen evidencia empírica de que la varianza es más baja o al menos igual para las funciones de activación.

Demostramos la desigualdad para las funciones de activación *ReLU*, *LeakyReLU*, *PReLU* y *ELU*[CNDM19]. Para ello, consideremos la siguiente familia de funciones de la Ecuación [C.2], donde g es una función continua arbitraria que satisface $x \leq g(x)$ para $x < 0$:

$$f(x) = \begin{cases} g(x) & x < 0 \\ x & x \geq 0 \end{cases} \qquad [C.2]$$

Todas las funciones de activación mencionadas pueden definirse en términos de la Ecuación [C.2] con elecciones apropiadas de $g(x)$. Además, todas satisfacen $x \leq g(x)$ para $x < 0$, con elecciones razonables del hiperparámetro α para *LeakyReLU* y *ELU* ($\alpha < 1$) y restricciones en el correspondiente *parámetro* α de las *PReLU* ($\alpha < 1$).

Finalmente, la Ecuación [C.3] demuestra entonces que $V(f(x)) \leq V(x)$ mediante las desigualdades $E(f(x)) \geq E(x)$ y $E(f(x)^2) \leq E(x^2)$ (Ecuaciones [C.4] y [C.5]), y la propiedad de la varianza $V(x) = E(x^2) - E(x)^2$.

$$\begin{aligned} V(f(x)) &= E(f(x)^2) - E(f(x))^2 \\ &\leq E(x^2) - E(f(x))^2 \\ &\leq E(x^2) - E(x)^2 \\ &= V(x) \end{aligned} \qquad [C.3]$$

$$\begin{aligned} E(f(x)) &= \int_{-\infty}^{\infty} f(x)p(x) \\ &= \int_{-\infty}^{0} g(x)p(x)dx + \int_{0}^{\infty} xp(x)dx \qquad [x \leq g(x)] \\ &\geq \int_{-\infty}^{0} xp(x)dx \quad + \int_{0}^{\infty} xp(x)dx \\ &= \int_{0}^{\infty} xp(x)dx \quad = E(x) \end{aligned} \qquad [C.4]$$

$$\begin{aligned} E(f(x)^2) &= \int_{-\infty}^{0} g(x)^2p(x)dx + \int_{0}^{\infty} x^2p(x)dx \\ &\leq \int_{-\infty}^{0} x^2p(x)dx + \int_{0}^{\infty} x^2p(x)dx \qquad [x \leq g(x)] \\ &= \int_{-\infty}^{\infty} x^2p(x)dx = E(x^2) \end{aligned} \qquad [C.5]$$

Bibliografía

[Aba+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard y col. «Tensorflow: A system for large-scale machine learning». En: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, págs. 265-283.

[Abe+10] Thomas Abeel, Thibault Helleputte, Yves Van de Peer, Pierre Dupont e Yvan Saeys. «Robust biomarker identification for cancer diagnosis with ensemble feature selection methods». En: *Bioinformatics* 26.3 (2010), págs. 392-398.

[AH99] Mohamed Azlan Hussain. «Review of the applications of neural networks in chemical process control-simulation and online implementation». En: *Artificial Intelligence in Engineering* 13.1 (1999), págs. 55-68.

[Aim+18] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu y col. «Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps». En: *IEEE transactions on neural networks and learning systems* 30.3 (2018), págs. 644-656.

[Amo+18] M. Amorim, F. Bortoloti, P. M. Ciarelli, E. de Oliveira y A. F. de Souza. «Analysing rotation-invariance of a log-polar transformation in convolutional neural networks». En: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, págs. 1-6. DOI: 10.1109/IJCNN.2018.8489295.

[AW18] Aharon Azulay y Yair Weiss. «Why do deep convolutional networks generalize so poorly to small image transformations?» En: *CoRR* abs/1805.12177 (2018). arXiv: 1805.12177.

[Bra+19] Danielle Bragg, Oscar Koller, Mary Bellard, Larwan Berke, Patrick Boudreault, Annelies Braffort, Naomi Caselli, Matt Huenerfauth, Hernisa Kacorri, Tessa Verhoef y col. «Sign Language Recognition, Genera-

tion, and Translation: An Interdisciplinary Perspective». En: *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*. 2019, págs. 16-31.

[BRW18] Charlotte Bunne, Lukas Rahmann y Thomas Wolf. «Studying invariances of trained convolutional neural networks». En: *arXiv preprint arXiv:1803.05963* (2018).

[Buc+19] Frédéric Bucci, Fabrizio Lillo, Jean-Philippe Bouchaud y Michael Benzaquen. *Are trading invariants really invariant? Trading costs matter*. 2019. arXiv: 1902.03457 [q-fin.TR].

[BZE09] Patrick Buehler, Andrew Zisserman y Mark Everingham. «Learning sign language by watching TV (using weakly aligned subtitles)». En: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, págs. 2961-2968.

[Cad+18] Santiago A. Cadena, Marissa A. Weis, Leon A. Gatys, Matthias Bethge y Alexander S. Ecker. «Diverse feature visualizations reveal invariances in early layers of deep neural networks». En: *The European Conference on Computer Vision (ECCV)*. Sep. de 2018.

[CBM07] Simon Conseil, Salah Bourennane y Lionel Martin. «Comparison of Fourier descriptors and Hu moments for hand posture recognition». En: *2007 15th European Signal Processing Conference*. IEEE. 2007, págs. 1960-1964.

[CET01] Timothy F Cootes, Gareth J Edwards y Christopher J Taylor. «Active appearance models». En: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 6 (2001), págs. 681-685.

[CF+19] Ulises Jeremias Cornejo Fandos, Gastón Gustavo Rios, Facundo Quiroga, Franco Ronchetti, Waldo Hasperue y Laura Cristina Lanzarini. «Recognizing Handshapes using Small Datasets». En: *XXIII Congreso Argentino de Ciencias de la Computación (Rio Cuarto, 2019)*. Ed. por Springer. 2019.

[CGL83] Tony F Chan, Gene H Golub y Randall J LeVeque. «Algorithms for computing the sample variance: Analysis and recommendations». En: *The American Statistician* 37.3 (1983), págs. 242-247.

[CGW19] Taco S Cohen, Mario Geiger y Maurice Weiler. «A general theory of equivariant cnns on homogeneous spaces». En: *Advances in Neural Information Processing Systems*. 2019, págs. 9142-9153.

[CHB11] Helen Cooper, Brian Holt y Richard Bowden. «Sign Language Recognition». En: *Visual Analysis of Humans: Looking at People*. Ed. por Thomas B. Moeslund, Adrian Hilton, Volker Krüger y Leonid Sigal. Springer, oct. de 2011. Cap. 27, págs. 539 -562. DOI: 10.1007/978-0-85729-997-0_27.

[Chi+19] Benjamin Chidester, Tianming Zhou, Minh N Do y Jian Ma. «Rotation equivariant and invariant neural networks for microscopy image analysis». En: *Bioinformatics* 35.14 (jul. de 2019), págs. i530-i537. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz353. eprint: http://oup.prod.sis.lan/bioinformatics/article-pdf/35/14/i530/28913604/btz353.pdf.

[CNDM19] Andrei Ciuparu, Adriana Nagy-Dăbâcan y Raul C. Mureșan. «Soft++, a multi-parametric non-saturating non-linearity that improves convergence in deep neural architectures». En: *Neurocomputing* (2019). ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.12.014.

[Coo+12] Helen Cooper, Eng-Jon Ong, Nicolas Pugeault y Richard Bowden. «Sign Language Recognition using Sub-Units». En: *Journal of Machine Learning Research* 13 (2012). Ed. por Isabelle Guyon y Vassilis Athitsos, págs. 2205-2231.

[CT17] Tony Martinez Christopher Tensmeyer. «Improving Invariance and Equivariance Properties of Convolutional Neural Networks». En: *International Conference on Learning Representations* (2017).

[CW16a] Taco S. Cohen y Max Welling. «Group Equivariant Convolutional Networks». En: *arXiv:1602.07576* [*cs, stat*] (feb. de 2016). arXiv: 1602.07576.

[CW16b] Taco S. Cohen y Max Welling. «Steerable CNNs». En: *arXiv:1612.08498* [*cs, stat*] (dic. de 2016). arXiv: 1612.08498.

[Dai+17] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu y Yichen Wei. «Deformable Convolutional Networks». En: *arXiv:1703.06211* [*cs*] (mar. de 2017). arXiv: 1703.06211.

[DDFK16] Sander Dieleman, Jeffrey De Fauw y Koray Kavukcuoglu. «Exploiting Cyclic Symmetry in Convolutional Neural Networks». En: *arXiv:1602.02660* [*cs*] (feb. de 2016). arXiv: 1602.02660.

[Den+09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei. «ImageNet: A Large-Scale Hierarchical Image Database». En: *CVPR09*. 2009.

[Den+13] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams y col. «Recent advances in deep learning for speech research at Microsoft». En: *ICASSP 2013* (2013).

[Dok+15] Ivan Dokmanic, Reza Parhizkar, Juri Ranieri y Martin Vetterli. «Euclidean distance matrices: essential theory, algorithms, and applications». En: *IEEE Signal Processing Magazine* 32.6 (2015), págs. 12-30.

[DV16] Vincent Dumoulin y Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2016. arXiv: 1603.07285 [stat.ML].

[DWD15] Sander Dieleman, Kyle W. Willett y Joni Dambre. «Rotation-invariant convolutional neural networks for galaxy morphology prediction». En: *Monthly Notices of the Royal Astronomical Society* 450.2 (jun. de 2015). arXiv: 1503.07077, págs. 1441-1459. ISSN: 1365-2966, 0035-8711. DOI: 10.1093/mnras/stv632.

[Eng+17] Logan Engstrom, Dimitris Tsipras, Ludwig Schmidt y Aleksander Madry. «A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations». En: *CoRR* abs/1712.02779 (2017).

[Far+13] Clément Farabet, Camille Couprie, Laurent Najman y Yann LeCun. «Learning hierarchical features for scene labeling». En: *IEEE* (2013).

[FF15] Alhussein Fawzi y Pascal Frossard. «Manitest: Are classifiers really invariant?» En: *CoRR* abs/1507.06535 (2015). arXiv: 1507.06535.

[FLR09] Andrea Finke, Alexander Lenhardt y Helge Ritter. «The MindGame: a P300-based brain–computer interface game». En: *Neural Networks* 22.9 (2009), págs. 1329-1333.

[For+12] Jens Forster, Christoph Schmidt, Thomas Hoyoux, Oscar Koller, Uwe Zelle, Justus Piater y Hermann Ney. «RWTH-PHOENIX-Weather: A Large Vocabulary Sign Language Recognition and Translation Corpus». En: *Language Resources and Evaluation*. Istanbul, Turkey, mayo de 2012, págs. 3785-3789.

[GAeS13] Heba Gamal, Hatem Abd elkader y Elsayed Sallam. «Hand gesture recognition using fourier descriptors». En: nov. de 2013, págs. 274-279. ISBN: 978-1-4799-0078-7. DOI: 10.1109/ICCES.2013.6707218.

[GBC16] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GCC13] Ahana Gangopadhyay, Oindrila Chatterjee y Amitava Chatterjee. «Hand shape based biometric authentication system using radon transform and collaborative representation based classification». En: *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*. IEEE. 2013, págs. 635-639.

[GD14] Robert Gens y Pedro M Domingos. «Deep Symmetry Networks». En: *Advances in Neural Information Processing Systems 27*. Ed. por Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence y K. Q. Weinberger. Curran Associates, Inc., 2014, págs. 2537-2545.

[Gei+19] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann y Wieland Brendel. «ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness». En: *ICLR 2019*. 2019.

[GH92] Wulfram Gerstner y J Leo van Hemmen. «Universality in neural networks: the importance of the 'mean firing rate'». En: *Biological Cybernetics* 67.3 (1992), págs. 195-205.

[Gil+17] Anna C Gilbert, Yi Zhang, Kibok Lee, Yuting Zhang y Honglak Lee. «Towards understanding the invertibility of convolutional neural networks». En: *Proceeding of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)* (2017).

[Goo+09] Ian Goodfellow, Honglak Lee, Quoc V Le, Andrew Saxe y Andrew Y Ng. «Measuring invariances in deep networks». En: *Advances in neural information processing systems*. 2009, págs. 646-654.

[Gra72] Gösta H Granlund. «Fourier preprocessing for hand print character recognition». En: *IEEE transactions on computers* 100.2 (1972), págs. 195-201.

[Hay94] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun. «Delving deep into rectifiers: Surpassing human-level performance on imagenet classification». En: *Proceedings of the IEEE international conference on computer vision*. 2015, págs. 1026-1034.

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun. «Deep residual learning for image recognition». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 770-778.

[Hin+12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath y col. «Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups». En: *Signal Processing Magazine, IEEE* 29.6 (2012), págs. 82-97.

[Ind+19] Piotr Indyk, Ali Vakilian, Tal Wagner y David Woodruff. «Sample-Optimal Low-Rank Approximation of Distance Matrices». En: *arXiv preprint arXiv:1906.00339* (2019).

[IS15] Sergey Ioffe y Christian Szegedy. «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». En: *International Conference on Machine Learning*. 2015, págs. 448-456.

[Jad+15] Max Jaderberg, Karen Simonyan, Andrew Zisserman y Koray Kavukcuoglu. «Spatial Transformer Networks». En: *arXiv:1506.02025* [*cs*] (jun. de 2015). arXiv: 1506.02025.

[Kan17] Can Kanbak. «Measuring Robustness of Classifiers to Geometric Transformations». Tesis de mtría. École polytechnique fédérale de Lausanne, 2017.

[Kan+18] Haribabu Kandi, Ayushi Jain, Swetha Velluva Chathoth, Deepak Mishra y Gorthi R. K. Sai Subrahmanyam. «Incorporating rotational invariance in convolutional neural network architecture». en. En: *Pattern Analysis and Applications* (feb. de 2018), págs. 1-14. ISSN: 1433-7541, 1433-755X. DOI: 10.1007/s10044-018-0689-0.

[Kau18] Eric Kauderer-Abrams. «Quantifying Translation-Invariance in Convolutional Neural Networks». En: *CoRR* abs/1801.01450 (2018). arXiv: 1801.01450.

[Kaw16] Kenji Kawaguchi. «Deep learning without poor local minima». En: *Advances in neural information processing systems*. 2016, págs. 586-594.

[KB14] Diederik P. Kingma y Jimmy Ba. «Adam: A Method for Stochastic Optimization». En: *CoRR* abs/1412.6980 (2014).

[KH00] Kyoung-jae Kim e Ingoo Han. «Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index». En: *Expert systems with applications* 19.2 (2000), págs. 125-132.

[Kha+19] Asifullah Khan, Anabia Sohail, Umme Zahoora y Aqsa Saeed Qureshi. «A survey of the recent architectures of deep convolutional neural networks». En: *arXiv preprint arXiv:1901.06032* (2019).

[Kir12] Roger E Kirk. «Experimental design». En: *Handbook of Psychology, Second Edition* 2 (2012).

[Kla+17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr y Sepp Hochreiter. «Self-normalizing neural networks». En: *Advances in neural information processing systems*. 2017, págs. 971-980.

[KNB16] Oscar Koller, Hermann Ney y Richard Bowden. «Deep hand: How to train a cnn on 1 million hand images when your data is continuous and weakly labelled». En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, págs. 3793-3802.

[Kra91] Mark A Kramer. «Nonlinear principal component analysis using autoassociative neural networks». En: *AIChE journal* 37.2 (1991), págs. 233-243.

[Kri+09] Alex Krizhevsky y col. *Learning multiple layers of features from tiny images*. Inf. téc. Citeseer, 2009.

[KTN15] Byeongkeun Kang, Subarna Tripathi y Truong Q Nguyen. «Real-time sign language fingerspelling recognition using convolutional neural networks from depth map». En: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. IEEE. 2015, págs. 136-140.

[KWT17] Devinder Kumar, Alexander Wong y Graham W. Taylor. «Explaining the Unexplained: A CLass-Enhanced Attentive Response (CLEAR) Approach to Understanding Deep Neural Networks». En: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2017.

[Lan+13] Laura Lanzarini, Franco Ronchetti, Cesar Estrebou, Luciana Lens y Aurelio Fernandez Bariviera. «Face recognition based on fuzzy probabilistic SOM». En: *IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS), 2013 Joint*. IEEE. 2013, págs. 310-314.

[Lap+16] Dmitry Laptev, Nikolay Savinov, Joachim M. Buhmann y Marc Pollefeys. «TI-POOLING: transformation-invariant pooling for feature learning in Convolutional Neural Networks». En: *CoRR* abs/1604.06318 (2016). arXiv: 1604.06318.

[Lar+07] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra y Yoshua Bengio. «An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation». En: *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. New York, NY, USA: ACM, 2007, págs. 473-480. ISBN: 978-1-59593-793-3. DOI: 10.1145/1273496.1273556.

[LB15] Dmitry Laptev y Joachim M. Buhmann. «Transformation-Invariant Convolutional Jungles». en. En: *Proceedings of CVPR 2015*. IEEE, jun. de 2015, págs. 3043-3051. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298923.

[LBH15] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. «Deep learning». En: *Nature* 521.7553 (2015), págs. 436-444.

[Le+11a] Quoc V Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean y Andrew Y Ng. «Building high-level features using large scale unsupervised learning». En: *arXiv preprint arXiv:1112.6209* (2011).

[Le+11b] Quoc V Le, Will Y Zou, Serena Y Yeung y Andrew Y Ng. «Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis». En: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, págs. 3361-3368.

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner y col. «Gradient-based learning applied to document recognition». En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324.

[Leh+07] Christoph Lehmann, Thomas Koenig, Vesna Jelic, Leslie Prichep, Roy E John, Lars-Olof Wahlund, Yadolah Dodge y Thomas Dierks. «Application and comparison of classification algorithms for recognition of Alzheimer's disease in electrical brain activity (EEG)». En: *Journal of neuroscience methods* 161.2 (2007), págs. 342-350.

[LH19] Ilya Loshchilov y Frank Hutter. «Decoupled Weight Decay Regularization». En: *International Conference on Learning Representations*. 2019.

[Lio12] Rudolf Lioutikov. «Machine learning and the brain». En: *Technological University Darmstad* (2012).

[Liu+17] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu y Fuad E Alsaadi. «A survey of deep neural network architectures and their applications». En: *Neurocomputing* 234 (2017), págs. 11-26.

[Low+99] David G Lowe y col. «Object recognition from local scale-invariant features.» En: *iccv*. Vol. 99. 2. 1999, págs. 1150-1157.

[Lua+17] Shangzhen Luan, Baochang Zhang, Chen Chen, Xianbin Cao, Jungong Han y Jianzhuang Liu. «Gabor Convolutional Networks». En: *arXiv:1705.01450 [cs]* (mayo de 2017). arXiv: 1705.01450.

[Lul+12] Dorothée Lulé, Quentin Noirhomme, Sonja C Kleih, Camille Chatelle, Sebastian Halder, Athena Demertzi, Marie-Aurélie Bruno, Olivia Gosseries, Audrey Vanhaudenhuyse, Caroline Schnakers y col. «Probing command following in patients with disorders of consciousness using a brain–computer interface». En: *Clinical Neurophysiology* (2012).

[LV14] Karel Lenc y Andrea Vedaldi. «Understanding image representations by measuring their equivariance and equivalence». En: *arXiv:1411.5908 [cs]* (nov. de 2014). arXiv: 1411.5908.

[LV15] Karel Lenc y Andrea Vedaldi. «Understanding image representations by measuring their equivariance and equivalence». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, págs. 991-999.

[Mas+18] Sarfaraz Masood, Adhyan Srivastava, Harish Chandra Thuwal y Musheer Ahmad. «Real-time sign language gesture (word) recognition from video sequences using CNN and RNN». En: *Intelligent Engineering Informatics*. Springer, 2018, págs. 623-632.

[MD00] Holger R Maier y Graeme C Dandy. «Neural networks for the prediction and forecasting of water resources variables: a review of modelling issues and applications». En: *Environmental modelling & software* 15.1 (2000), págs. 101-124.

[Mel+09] Andrew N Meltzoff, Patricia K Kuhl, Javier Movellan y Terrence J Sejnowski. «Foundations for a new science of learning». En: *science* 325.5938 (2009), págs. 284-288.

[Mit97] Thomas M. Mitchell. *Machine Learning*. 1.ª ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077, 9780070428072.

[Mos+09] Ahmed Mostayed, Md Ekramul Kabir, Saurav Zaman Khan y Md Mynuddin Gani Mazumder. «Biometric authentication from low resolution hand images using radon transform». En: *2009 12th International Conference on Computers and Information Technology*. IEEE. 2009, págs. 587-592.

[MVT16] Diego Marcos, Michele Volpi y Devis Tuia. «Learning rotation invariant convolutional filters for texture classification». En: *arXiv:1604.06720 [cs]* (dic. de 2016). arXiv: 1604.06720, págs. 2012-2017. DOI: 10.1109/ICPR.2016.7899932.

[NW01] Frank Natterer y Frank Wübbeling. *Mathematical methods in image reconstruction*. Vol. 5. Siam, 2001.

[OB04] Eng-Jon Ong y Richard Bowden. «A boosted classifier tree for hand shape detection». En: *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.* IEEE. 2004, págs. 889-894.

[Oli18] Marlon Oliveira. «Handshape recognition using principal component analysis and convolutional neural networks applied to sign language». Tesis doct. Dublin City University, 2018.

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga y col. «PyTorch: An imperative style, high-performance deep learning library». En: *Advances in Neural Information Processing Systems*. 2019, págs. 8024-8035.

[PB11] N. Pugeault y R. Bowden. «Spelling It Out: Real-Time ASL Fingerspelling Recognition». En: *1st IEEE Workshop on Consumers Depth Cameras for Computer Vision, in conjunction with ICCV'2011*. 2011.

[Pen+14] Xingchao Peng, Baochen Sun, Karim Ali y Kate Saenko. «Exploring invariances in deep convolutional neural networks using synthetic images». En: *CoRR, abs/1412.7122* 2.4 (2014).

[Pom91] Dean A Pomerleau. «Efficient training of artificial neural networks for autonomous navigation». En: *Neural Computation* 3.1 (1991), págs. 88-97.

[Pom93] Dean A Pomerleau. «Knowledge-based training of artificial neural networks for autonomous robot driving». En: *Robot learning*. Springer, 1993, págs. 19-43.

[QC13] Facundo Quiroga y Leonardo César Corbalán. «A novel competitive neural classifier for gesture recognition with small training sets». En: *XVIII Congreso Argentino de Ciencias de la Computación (CACIC)*. Red de Universidades con Carreras en Informática (RedUNCI). 2013.

[Qui+16a] Facundo Quiroga, Franco Ronchetti, César Armando Estrebou y Laura Cristina Lanzarini. «Handshape recognition for argentinian sign language using probsom». En: *Journal of Computer Science & Technology* 16.1 (2016). ISSN: 1666-6038.

[Qui+16b] Facundo Quiroga, Franco Ronchetti, César Armando Estrebou, Laura Cristina Lanzarini y Alejandro Rosete. «LSA64: An Argentinian Sign Language Dataset». En: *XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016)*. Ed. por Springer. Red de Universidades con Carreras en Informática (RedUNCI). 2016, págs. 794-803.

[Qui+16c] Facundo Quiroga, Franco Ronchetti, César Estrebou, Laura Lanzarini y Alejandro Rosete. «Sign languague recognition without frame-sequencing constraints: A proof of concept on the argentinian sign language». En: *Ibero-American Conference on Artificial Intelligence*. Springer International Publishing. 2016, págs. 338-349.

[Qui+17] Facundo Quiroga, Ramiro Antonio, Franco Ronchetti, Laura Cristina Lanzarini y Alejandro Rosete. «A study of convolutional architectures for handshape recognition applied to sign language». En: *XXIII Congreso Argentino de Ciencias de la Computación (La Plata, 2017)*. Springer International Publishing. 2017.

[Qui+18] Facundo Quiroga, Franco Ronchetti, Laura Lanzarini y Aurelio F Bariviera. «Revisiting Data Augmentation for Rotational Invariance in Convolutional Neural Networks». En: *International Conference on Modelling and Simulation in Management Sciences*. Springer. 2018, págs. 127-141.

[Qui+19] Facundo Quiroga, Jordina Torrents-Barrena, Laura Lanzarini y Domenec Puig. «Measuring (in) variances in Convolutional Networks». En: *Conference on Cloud Computing and Big Data*. Springer. 2019, págs. 98-109.

[RBS12] Akbar Rahideh, AH Bajodah y M Hasan Shaheed. «Real time adaptive nonlinear model inversion control of a twin rotor MIMO system using neural networks». En: *Engineering Applications of Artificial Intelligence* 25.6 (2012), págs. 1289-1297.

[RMG14] Lucas Rioux-Maldague y Philippe Giguere. «Sign Language Fingerspelling Classification from Depth and Color Images using a Deep Belief Network». En: *Computer and Robot Vision (CRV), 2014 Canadian Conference on*. IEEE. 2014, págs. 92-97.

[Ron+15] Franco Ronchetti, Facundo Quiroga, Laura Lanzarini y Cesar Estrebou. «Distribution of Action Movements (DAM): a Descriptor for Human Action Recognition». En: *Frontiers of Computer Science* 9.6 (2015), págs. 956-965. ISSN: 2095-2236. DOI: 10.1007/s11704-015-4320-x.

[Ron18] Franco Ronchetti. «Reconocimiento de gestos dinámicos y su aplicación al lenguaje de señas». En: *XX Workshop de Investigadores en Ciencias de la Computación (WICC 2018, Universidad Nacional del Nordeste)*. 2018.

[Rou+10a] Anastasios Roussos, Stavros Theodorakis, Vassilis Pitsikalis y Petros Maragos. «Hand Tracking and Affine Shape-Appearance Handshape Sub-units in Continuous Sign Language Recognition». En: *Trends and Topics in Computer Vision - ECCV 2010 Workshops, Heraklion, Crete, Greece, September 10-11, 2010, Revised Selected Papers, Part I*. 2010, págs. 258-272. DOI: 10.1007/978-3-642-35749-7_20.

[Rou+10b] Anastasios Roussos, Stavros Theodorakis, Vassilis Pitsikalis y Petros Maragos. «Hand tracking and affine shape-appearance handshape sub-units in continuous sign language recognition». En: *European Conference on Computer Vision*. Springer. 2010, págs. 258-272.

[SG18] Megha Srivastava y Kalanit Grill-Spector. «The Effect of Learning Strategy versus Inherent Architecture Properties on the Ability of Convolutional Neural Networks to Develop Transformation Invariance». En: *CoRR* abs/1810.13128 (2018). arXiv: 1810.13128.

[SGL12] Yeou-Ren Shiue, Ruey-Shiang Guh y Ken-Chun Lee. «Development of machine learning-based real time scheduling systems: using ensemble based on wrapper feature selection approach». En: *International Journal of Production Research* 50.20 (2012), págs. 5887-5905.

[Sha+16] Wenling Shang, Kihyuk Sohn, Diogo Almeida y Honglak Lee. «Understanding and improving convolutional neural networks via concatenated rectified linear units». En: *International Conference on Machine Learning*. 2016, págs. 2217-2225.

[Sha53] Claude E Shannon. «Computers and automata». En: *Proceedings of the IRE* 41.10 (1953), págs. 1234-1241.

[Sid+12] Gagan S Sidhu, Nasimeh Asgarian, Russell Greiner y Matthew RG Brown. «Kernel Principal Component Analysis for dimensionality reduction in fMRI-based diagnosis of ADHD». En: *Frontiers in systems neuroscience* 6 (2012).

[SK08] Neal Schmitt y Goran Kuljanin. «Measurement invariance: Review of practice and implications». En: *Human Resource Management Review* 18.4 (2008). Research Methods in Human Resource Management, págs. 210 -222. ISSN: 1053-4822. DOI: https://doi.org/10.1016/j.hrmr.2008.03.003.

[Smi99] Kate A Smith. «Neural networks for combinatorial optimization: a review of more than a decade of research». En: *INFORMS Journal on Computing* 11.1 (1999), págs. 15-34.

[Spr+15] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox y Martin A. Riedmiller. «Striving for Simplicity: The All Convolutional Net». En: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. 2015.

[SZ14] Karen Simonyan y Andrew Zisserman. «Very Deep Convolutional Networks for Large-Scale Image Recognition». En: *arXiv e-prints*, arXiv:1409.1556 (sep. de 2014), arXiv:1409.1556. arXiv: 1409.1556 [cs.CV].

[Sze+15a] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke y Andrew Rabinovich. «Going deeper with convolutions». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, págs. 1-9.

[Sze+15b] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens y Zbigniew Wojna. «Rethinking the Inception Architecture for Computer Vision». En: *CoRR* abs/1512.00567 (2015).

[Tan+14] Wenjun Tan, Zijiang Bian, Jinzhu Yang, Huang Geng, Zhaoxuan Gong y Dazhe Zhao. «Hand Gesture Shape Descriptor Based on Energy-Ratio and Normalized Fourier Transform Coefficients». En: *Advances in Swarm Intelligence*. Ed. por Ying Tan, Yuhui Shi y Carlos A. Coello Coello. Cham: Springer International Publishing, 2014, págs. 34-41. ISBN: 978-3-319-11897-0.

[TC16] Chuan-Yung Tsai y DD Cox. «Characterizing visual representations within convolutional neural networks: Toward a quantitative approach». En: *ICML Workshop on Vis for Deep Learning*. 2016.

[TL19] Mingxing Tan y Quoc V Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». En: *arXiv preprint arXiv:1905.11946* (2019).

[VA+08] Ulrich Von Agris, Jörg Zieren, Ulrich Canzler, Britta Bauer y Karl-Friedrich Kraiss. «Recent developments in visual sign language recognition». En: *Universal Access in the Information Society* 6.4 (2008), págs. 323-362.

[Vel08] Rosemarie Velik. «Discrete Fourier Transform Computation Using Neural Networks». En: *Computational Intelligence and Security, 2008. CIS'08. International Conference on*. Vol. 1. IEEE. 2008, págs. 120-123.

[VL00] Robert J. Vandenberg y Charles E. Lance. «A Review and Synthesis of the Measurement Invariance Literature: Suggestions, Practices, and Recommendations for Organizational Research». En: *Organizational Research Methods* 3.1 (2000), págs. 4-70. DOI: 10.1177/109442810031002. eprint: https://doi.org/10.1177/109442810031002.

[WHK15] Fa Wu, Peijun Hu y Dexing Kong. «Flip-Rotate-Pooling Convolution and Split Dropout on Convolution Neural Networks for Image Classification». En: *arXiv:1507.08754 [cs]* (jul. de 2015). arXiv: 1507.08754.

[WHS18] Maurice Weiler, Fred A Hamprecht y Martin Storath. «Learning steerable filters for rotation equivariant CNNs». En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, págs. 849-858.

[WRL94] Bernard Widrow, David E Rumelhart y Michael A Lehr. «Neural networks: Applications in industry, business and science». En: *Communications of the ACM* 37.3 (1994), págs. 93-105.

[Xie+17] Qizhe Xie, Zihang Dai, Yulun Du, Eduard Hovy y Graham Neubig. «Controllable invariance through adversarial feature learning». En: *Advances in Neural Information Processing Systems*. 2017, págs. 585-596.

[Yan10] Quan Yang. «Chinese sign language recognition based on video sequence appearance modeling». En: *2010 5th IEEE Conference on Industrial Electronics and Applications*. IEEE. 2010, págs. 1537-1542.

[Yua+10] Ruixi Yuan, Zhu Li, Xiaohong Guan y Li Xu. «An SVM-based machine learning method for accurate internet traffic classification». En: *Information Systems Frontiers* 12.2 (2010), págs. 149-156.

[ZD19] Y. Zhong y W. Deng. «Exploring Features and Attributes in Deep Face Recognition Using Visualization Techniques». En: *2019 14th IEEE International Conference on Automatic Face Gesture Recognition (FG 2019)*. 2019, págs. 1-8. DOI: 10.1109/FG.2019.8756546.

[ZF14] Matthew D Zeiler y Rob Fergus. «Visualizing and understanding convolutional networks». En: *European conference on computer vision*. Springer. 2014, págs. 818-833.

[Zha+17] Xin Zhang, Li Liu, Yuxiang Xie, Jie Chen, Lingda Wu y Matti Pietikainen. «Rotation Invariant Local Binary Convolution Neural Networks». en. En: *Proceedings of the ICCV 2017*. IEEE, oct. de 2017, págs. 1210-1219. ISBN: 978-1-5386-1034-3. DOI: 10.1109/ICCVW.2017.146.

[Zho+17] Yanzhao Zhou, Qixiang Ye, Qiang Qiu y Jianbin Jiao. «Oriented Response Networks». En: *arXiv:1701.01833 [cs]* (ene. de 2017). arXiv: 1701.01833.

[ZW12] Xiaolong Zhu y Kenneth KY Wong. «Single-frame hand gesture recognition using color and depth kernel descriptors». En: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE. 2012, págs. 2989-2992.

[ZYT13] Chenyang Zhang, Xiaodong Yang y YingLi Tian. «Histogram of 3D facets: A characteristic descriptor for hand gesture recognition». En: *Automatic Face and Gesture Recognition (FG), 2013 10th IEEE International Conference and Workshops on*. IEEE. 2013, págs. 1-8.