UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Integración de diferentes técnicas para visualizar la influencia de regiones de una imagen en su clasificación por una red neuronal.

AUTORES: Andrés Mauro Gardella Ruiz

DIRECTOR/A: Claudia Pons

CODIRECTOR/A: Gabriela Pérez

CARRERA: Licenciatura en informática

Resumen

Hoy en día es común la utilización, en múltiples ámbitos, de redes neuronales que permiten realizar actividades complejas como clasificación de imágenes. Tecnología muy útil debido a la información que provee, aunque en contextos sensibles, como el de la salud pública, es necesario poder entender y confiar en dicha información, ya que la falta de precisión puede acarrear consecuencias negativas en términos generales.

Debido a la necesidad de comprender el funcionamiento y la toma de decisiones de las redes neuronales comienzan a surgir métodos y técnicas de visualización, que permiten visualizar neuronas para comprender mejor las decisiones tomadas por éstas en base a la información ingresada.

El propósito de este trabajo es analizar varios de dichos métodos y en base a eso realizar un programa que simplifique tanto su utilización como la visualización de dichas explicaciones, haciendo que las técnicas para comprender el funcionamiento y decisiones de la red sean más accesibles.

Palabras Clave

Red Neuronal Convolucional, Análisis de Imágenes, Clasificación, verificación, Visualización, Decisiones, LIME, SHAP, Grad-CAM

Trabajos Realizados

Se realizó una investigación para comprender y caracterizar el ámbito del machine learning desde su base teórica hasta llegar a redes neuronales convolucionales.

Se realizaron análisis teóricos de múltiples métodos de visualización de redes neuronales para luego implementarlos en software y poder realizar comparaciones entre los resultados obtenidos.

Se investigaron múltiples Frameworks y herramientas de software para llevar a cabo el desarrollo de un programa que integre implementaciones de redes neuronales convolucionales y métodos de visualización.

Conclusiones

Se desarrolló un componente de software que incorpora múltiples redes neuronales para clasificación de imágenes junto a métodos de visualización que permiten verificar los resultados obtenidos, haciéndolos más comprensibles.

En base a la investigación realizada se infieren y resuelven algunas complicaciones relacionados con este tipo de aplicación y se identifican problemas a solucionar en caso de buscar expandir sus capacidades.

Trabajos Futuros

Algunos trabajos futuros que se desprenden de la tesina son:

- Agregar redes neuronales convolucionales para clasificación de imágenes.
- Agregar métodos de visualización.
- Mejorar las implementaciones actuales de métodos de visualización para facilitar la expansión del programa a futuro.
- Mejorar la capacidad de configuración tanto de redes como de métodos de visualización

Fecha de la presentación:

Capítulo 1: Introducción

Hoy en día la inteligencia artificial está presente en la vida de prácticamente todas las personas, ya que cada vez más dispositivos incorporan alguna forma de software “inteligente” para realizar sus tareas. Además, estos tipos de sistemas también son encontrados de manera más frecuente en ámbitos críticos como la medicina o finanzas, los cuales requieren la mayor precisión posible tanto en la información que se maneja como en la forma de gestionarla.

Los sistemas “inteligentes” son de gran ayuda y muchas veces son más eficientes que las personas mismas. Sin embargo, no son infalibles, no se puede confiar ciegamente en la información que proveen, mucho menos en ámbitos donde esta información es crítica y un error puede causar algún tipo de daño a las personas. Debido a esto es necesario que estos sistemas puedan presentar información acerca de cómo toman sus decisiones, permitiendo corroborar que estas tengan sentido. Por ejemplo, si un médico utiliza una red neuronal para analizar una imagen de un paciente y realizar un diagnóstico, es importante que pueda verificar luego que la decisión tomada por la red se corresponda con la información analizada, por ejemplo identificando la región de la imagen que afecta a esa decisión para poder analizarla luego más detalladamente.

Estos sistemas crecen en complejidad a medida que aumentan sus capacidades, por lo que llega un punto en que se vuelve imposible el entender los motivos de las decisiones que toma. Por esta razón se los considera una caja negra o “black box”, en las que un usuario introduce información y obtiene un resultado sin conocimiento del proceso que ocurre en el sistema más que la base matemática bajo la cual se crea. Debido a esto surgen problemas relacionados a la falta de confianza en los resultados o problemas para encontrar errores en el modelo en sí a modo de poder corregirlos. Por ejemplo, se han documentado casos en los cuales las redes clasificaban imágenes de forma errónea. En uno de ellos, la red detectaba perros Husky como lobos. Este error se debió a que muchas de las imágenes de lobos usadas como datos de entrenamiento contenían nieve y por esta razón ante la presencia de nieve la red lo clasificaba como lobo[1]. En otro caso, debido a la presencia de un “sticker” en una imagen, el sistema reconocía el contenido como una tostadora [2].

Actualmente el procesamiento de imágenes para clasificación o detección de objetos es utilizado en múltiples sectores, como en la industria automotriz, electrónica, o demás ámbitos en los que ocurre algún tipo de fabricación y/o ensamblaje. Estas redes se utilizan para inspeccionar los componentes, controlar la calidad de los alimentos en el sector alimenticio, soporte en la realización de diagnósticos, identificación de tumores o enfermedades con síntomas visibles en medicina[3], etc. Por esta razón sería de vital importancia poder entender y explicar las decisiones tomadas por estos sistemas.

Las redes neuronales convolucionales son ampliamente utilizadas para el tratamiento de imágenes por su gran rendimiento, reconociendo patrones en imágenes, detección de objetos, etc. Sin embargo carecen de interpretabilidad debido a su complejidad. Afortunadamente hoy en día existen técnicas y métodos que aportan algo de

interpretabilidad a estos modelos permitiendo reconocer qué partes de la imagen procesada afectan a su decisión.

En este trabajo se llevará a cabo un análisis de varias arquitecturas de redes neuronales, así como métodos de visualización de decisiones. Posteriormente se seleccionarán algunas arquitecturas y métodos de visualización en base a sus características, tales como facilidad de uso o capacidad de ser utilizados en conjunto. Finalmente, se los integrará en un programa que provea una versión funcional de dichos elementos, con el objetivo de facilitar el acceso a estos métodos de visualización y simplificar su utilización por parte del usuario.

1.1 Motivación

La inteligencia artificial y aprendizaje profundo han tenido grandes avances en los últimos años y como consecuencia, surgieron modelos capaces de realizar actividades muy complejas con éxito, como por ejemplo detección de objetos y clasificación de imágenes. Este avance implica que los modelos sean cada vez más complejos, generando como consecuencia el inconveniente intrínseco de la falta de transparencia. Por esta razón, han surgido métodos para visualizar las predicciones de las redes neuronales. Sin embargo, la información sobre estos métodos a veces es insuficiente, no están documentados adecuadamente, son muy diferentes entre sí y/o es difícil encontrar implementaciones útiles. En muchos casos crear una implementación desde cero solamente en base a la documentación es particularmente dificultoso. Por esta razón, en este trabajo se busca desarrollar una herramienta que integre un conjunto de redes neuronales y métodos de visualización de manera de simplificar su uso y facilitar el análisis de los fundamentos de las predicciones realizadas.

1.2 Objetivos

1.2.1 Generales:

El objetivo de esta tesina es profundizar en el estudio de las técnicas de visualización utilizadas para analizar las predicciones de las redes neuronales convolucionales que permiten explicar las predicciones de la red. Además, se busca integrar las diversas técnicas de justificación de toma de decisiones en redes neuronales destinadas a la clasificación de imágenes, construyendo una aplicación. Esta aplicación simplificará la visualización de estas explicaciones, haciendo que las técnicas sean más accesibles y que el funcionamiento de la red sea más comprensible

1.2.2 Específicos:

1. Estudiar redes neuronales para clasificación de imágenes, como las redes neuronales convolucionales
2. Estudiar distintas arquitecturas de redes neuronales convolucionales ya definidas.
3. Analizar distintas técnicas de interpretación para explicar el funcionamiento de redes neuronales para entender las decisiones respecto a clasificación de imágenes
4. Integrar las diferentes técnicas y redes estudiadas desarrollando una aplicación que simplifique su utilización y la visualización de las explicaciones.

1.3 Resultados esperados

Como resultado general se espera la creación e implementación exitosa de un programa que permita reunir múltiples redes neuronales convolucionales y métodos de visualización, para que de esta manera sea posible realizar clasificación de imágenes mediante inteligencia artificial y al mismo tiempo obtener un feedback que permita explicarlas.

Los resultados particulares esperados son:

- Análisis de métodos de visualización y desarrollo del código para su implementación
- Desarrollo de un programa que integre las redes neuronales con métodos de visualización
- Mejora en el entendimiento acerca de las decisiones tomadas por redes neuronales en la clasificación de imágenes
- Mejora en la facilidad de uso y acceso a métodos de visualización de redes neuronales convolucionales

1.4 Estructura de la tesina

En este apartado se realiza una breve descripción de los capítulos que componen esta tesina.

En el primer capítulo se explica la motivación y los objetivos propuestos. El segundo capítulo “Información preliminar” presenta el marco teórico, la información necesaria para entender varios de los conceptos a ser tratados durante este trabajo, explicando machine learning, las redes neuronales y el deep learning, para luego mostrar elementos importantes en el entrenamiento de las redes y por último centrarse en redes neuronales convolucionales, un subconjunto de redes neuronales las cuales serán utilizadas en este trabajo.

En el tercer capítulo “Visualización de redes neuronales convolucionales” se presentan métodos de visualización de redes neuronales convolucionales, explicando qué son, por qué es posible su funcionamiento y qué información permiten obtener, para terminar mostrando múltiples métodos de visualización particulares explicando cómo funcionan, junto a arquitecturas de redes neuronales que podrían ser analizadas mediante dichos métodos.

En el cuarto capítulo “Desarrollo” se realizan implementaciones de los métodos de visualización para posteriormente realizar un análisis de dichos métodos y de los resultados obtenidos al trabajar junto a diferentes redes neuronales. En base a eso se seleccionan algunos (tanto redes como métodos) para incorporarlos a la aplicación a desarrollar.

En el quinto capítulo “Herramienta propuesta” se presenta el proceso de desarrollo de la aplicación, explicando herramientas y frameworks utilizados, decisiones respecto a su construcción y por último se muestra su funcionamiento.

Finalmente, en el sexto capítulo “Conclusiones y trabajos futuros” se presentan las conclusiones obtenidas con respecto a la aplicación desarrollada para incorporar múltiples métodos de visualización para justificar la toma de decisiones en redes neuronales destinadas a la clasificación de imágenes y el proceso de desarrollo de la misma, además de proponer ideas para posibles trabajos futuros respecto a mejoras y modificaciones posibles para la herramienta desarrollada.

Capítulo 2: Información preliminar

En este capítulo se presenta el marco teórico, la información necesaria para entender los conceptos tratados en este trabajo, explicando qué es el machine learning, las redes neuronales y el deep learning, para luego mostrar elementos importantes en el entrenamiento de las redes, representación de imágenes, y por último centrarse en redes neuronales convolucionales, un subconjunto de redes neuronales que serán utilizadas en este trabajo.

2.1 Machine Learning

El primer apartado se centra en la definición de machine learning y la taxonomía de estos algoritmos. Luego se clasifican en base a los tipos de problema que pueden resolver, y se explican algunos conceptos a tener en cuenta durante el entrenamiento, como la separación de datos en los conjuntos de entrenamiento, prueba y validación, y métricas para poder determinar su capacidad de cumplir con lo que se requiere de los mismos.

El Machine Learning, o aprendizaje automático es una disciplina dentro del campo de la inteligencia artificial que abarca a los sistemas que, mediante algoritmos, pueden aprender automáticamente. Esto significa que pueden identificar patrones en datos masivos, y en base a estos consiguen realizar predicciones. De esta manera, los sistemas son capaces de realizar tareas específicas de forma autónoma sin intervención humana respecto a tener que programar explícitamente reglas o información en base a las cuales obtengan la capacidad de producir dichas predicciones.

Respecto a la diferencia entre machine learning y programación tradicional, se puede considerar que la ingeniería de software es el arte de automatizar una tarea escribiendo reglas para que una computadora las siga, mientras que en el caso de machine learning se avanza aún más al automatizar la tarea de obtener las reglas.

La programación tradicional es un proceso manual, en el que se deben formular y codificar reglas específicas. El programador codifica un programa que se ejecuta utilizando los datos de entrada para producir una salida deseada. Mientras tanto, en el aprendizaje automático la computadora recibe los datos de entrada y las salidas esperadas, y no hace falta que nadie programe la lógica, ya que dichas reglas son generadas a partir de los datos ingresados al sistema. En la parte superior de la figura 1, puede observarse que la computadora recibe datos de entrada, y las reglas o el programa, los cuales son procesados para obtener una salida, mientras que en el segundo caso la computadora recibe datos de entrada y la salida esperada para dar como resultado el propio programa

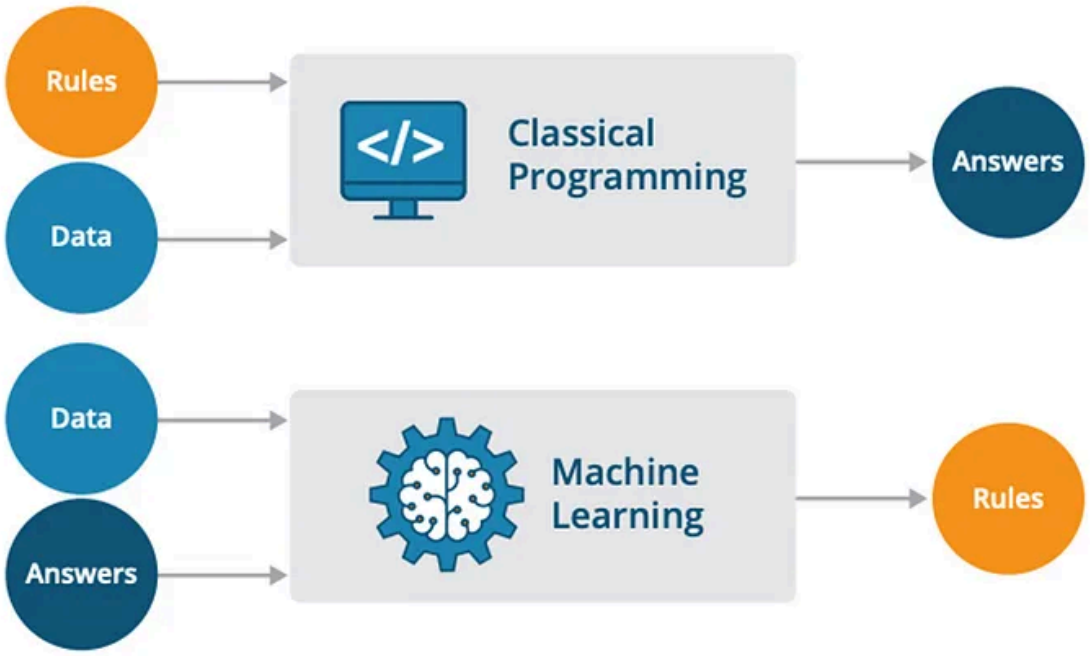


Figura 1: Programación tradicional vs. Aprendizaje automático.[4]

Si bien muchos modelos de Aprendizaje Automático existen desde hace varias décadas, estos recobraron importancia en los últimos años debido principalmente a dos motivos. Por un lado, al aumento en la capacidad computacional de las computadoras, lo que permite tratar problemas que antes eran inviables; y por otro lado, al aumento exponencial en la cantidad de datos disponibles debido a la digitalización, los cuales pueden ser procesados y modelizados para obtener información de los mismos.

Los algoritmos en Machine Learning, se agrupan en tres categorías:

- Aprendizaje Supervisado: Es una técnica para deducir una función a partir de datos de entrenamiento, los cuales consisten de datos de entrada y las etiquetas o resultados deseados. En base a estos, la salida de la función puede ser un valor numérico en el caso de problemas de regresión, o una etiqueta en el caso de problemas de clasificación. El objetivo del aprendizaje supervisado es crear una función que pueda predecir el valor correspondiente a cualquier entrada luego de haber visto una serie de ejemplos (los datos de entrenamiento)
- Aprendizaje no supervisado: En este caso no existe un conocimiento del resultado esperado. Únicamente se tratan datos de entrada los cuales son considerados como un conjunto de variables aleatorias. De esta forma los algoritmos descubren agrupaciones de datos o patrones ocultos sin necesidad de intervención humana. Estas características lo hacen muy útil para el análisis de datos exploratorios, estrategias de venta cruzada o segmentación de clientes.
- Aprendizaje por refuerzo: El algoritmo aprende observando el mundo que le rodea. Su información de entrada es la retroalimentación que obtiene del mundo exterior como respuesta a sus acciones. De esta manera el sistema aprende por prueba y error. Esta categoría es la más general. En vez de que el agente tenga instrucciones,

aprende cómo se comporta el entorno mediante refuerzos o castigos, derivados del éxito o fracaso. De esta manera el objetivo general del sistema es encontrar la función que maximice la recompensa para optimizar la toma de buenas decisiones.

Los sistemas analizados en este trabajo son de aprendizaje supervisado. Como se mencionó previamente, estos sistemas pueden resolver problemas de regresión o de clasificación, cuya principal diferencia es el tipo de salida resultante, ya que en los problemas de regresión se obtiene un número real, mientras que en el de clasificación se obtienen valores discretos, que representan clases.

2.1.1 Problemas de Clasificación y Regresión

Los ejemplos más simples de algoritmos para resolver estos tipos de problemas son regresión lineal simple, para el caso de problemas de regresión, y regresión logística para los problemas de clasificación.

La regresión lineal simple, es una técnica de modelado estadístico que se emplea para relacionar una variable independiente x, que es la entrada, con una variable dependiente y, que es la salida. Para esto, se asume que hay una relación funcional entre ambas variables, siendo w y b estimadores, parámetros que hay que entrenar (adaptar sus valores) para que se obtenga la salida esperada en base a los diferentes datos de entrada que se posean.

La fórmula resultante es $y = w.x + b$, donde w,b $\in$ R son constantes desconocidos llamadas coeficientes de regresión. *w* puede considerarse la pendiente de la recta, y *b* la ordenada al origen. En la figura 2 puede verse en rojo, gráficamente, la función.

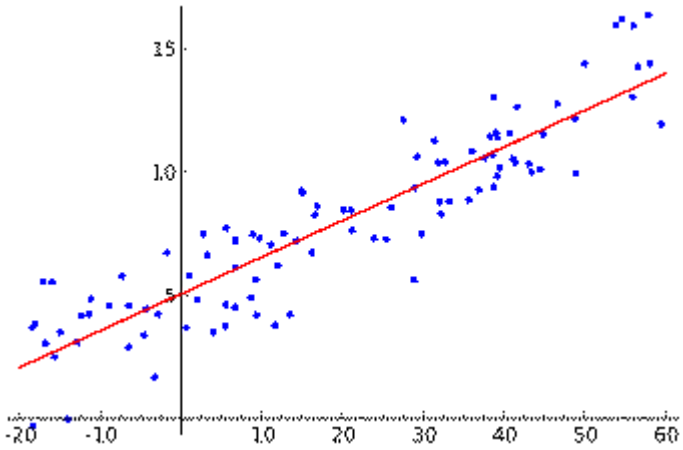


Figura 2: Ejemplo de una regresión lineal con una variable dependiente y una variable independiente

Para poder entrenar dichos parámetros se tienen ejemplos de entrenamiento, es decir, conjuntos de datos de entrada con sus correspondientes etiquetas o salidas esperadas, de esta forma se puede utilizar dicha información para modificar b y w hasta

encontrar los valores que consiguen que la salida arrojada por el modelo sea lo más parecida posible a los valores esperados de salida (los valores de y) en relación a sus valores de entrada x. Para esto, se tiene que medir que tan cercanos son los resultados obtenidos con los deseados, y esto se consigue mediante una medida de error que suele ser la diferencia elevada al cuadrado (método de mínimos cuadrados), es decir $(y - d)^2$ donde y es la salida actual obtenida por el modelo y d es la salida deseada (los valores de salida etiquetados como Y en los datos de entrenamiento), con todo esto, el algoritmo de aprendizaje se centra en minimizar el resultado de la sumatoria de los errores de cada ejemplo, para de esta manera minimizar el error global.

Esta minimización del error se puede realizar de dos maneras. La primera es la forma analítica, en la que se iguala a 0 el gradiente de la función de error global con respecto a cada parámetro del modelo, para luego resolver el sistema de ecuaciones resultante. Aunque este método suele ser empleado al realizar este tipo de procedimiento manualmente, se vuelve inviable computacionalmente a medida que aumenta la cantidad de variables a procesar.

La segunda forma, viable computacionalmente sin importar la cantidad de variables, es el método de descenso por gradiente. Este método resuelve el problema iterativamente partiendo de valores aleatorios para los parámetros a entrenar, calculando el gradiente para ese punto, y usando esa información para actualizar los valores de los parámetros. El objetivo es que converjan hacia el valor en el que el gradiente del error con respecto al parámetro sea 0. Esto se representa mediante la fórmula $\Theta = \Theta - \alpha\frac{\partial J}{\partial \Theta}$ donde Θ es el parámetro a actualizar, J es la función de error que se pretende minimizar y α es el ratio de aprendizaje, un parámetro que se utiliza para controlar la velocidad con la que se desciende el gradiente, de manera de garantizar una convergencia eficiente. En la figura 3 se muestra como, en una iteración de descenso por el gradiente, Θ disminuye, pues la pendiente de la recta tangente a la función de error en ese punto (derivada) es positiva (la función es creciente en ese punto), luego el valor mínimo de J se encuentra a la izquierda del punto

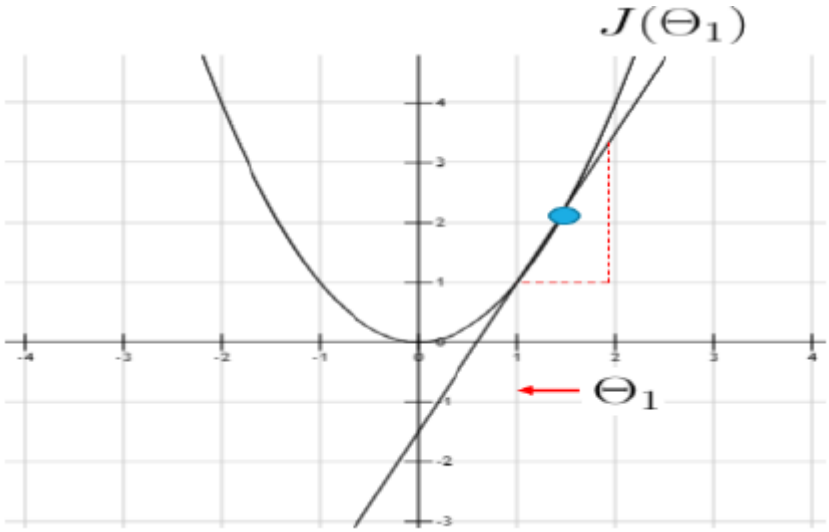


Figura 3: Una iteración de descenso por gradiente. [5 p.8]

En el caso de la regresión logística, se trata de un modelo estadístico que se utiliza para determinar la probabilidad de que ocurra un evento con el objetivo de predecir el resultado de una variable categórica, es decir, una variable que puede adoptar un número limitado de valores posibles o categorías. Es similar a la regresión lineal, salvo por el hecho de que el resultado es binario en este caso, a diferencia de la regresión lineal donde es un número real, siendo 1 si la salida pertenece a la clase o 0 en caso contrario. Más específicamente, el resultado es un valor entre 0 y 1 que indica la probabilidad de pertenecer a determinada clase. Cuando ese valor es mayor o igual a 0.5 se considera que pertenece a la clase, por lo que el valor se considera 1, mientras que en el caso contrario se considera 0.

Durante este procedimiento, se inicia con un vector X de valores de entrada, un vector de pesos W y un parámetro independiente b, que corresponden a los coeficientes o parámetros del modelo que se obtienen durante el proceso de entrenamiento. de esta forma la salida z es: $z = X \cdot W + b$

Una vez transformados los datos, este resultado se tiene que ingresar a una función de activación, ya que z tiene un rango continuo de valores, pero como se vio previamente, el resultado tiene que ser 0 o 1.

Esta función de activación tiene un comportamiento no lineal, y en el caso de la regresión logística se usa la función sigmoide $\sigma(z) = \frac{1}{1+e^{-z}}$ en la cual la entrada puede tener cualquier valor tanto positivo como negativo, pero la salida siempre estará en el rango de 0 a 1. De esta manera la función devuelve una probabilidad de que el dato pertenezca a una de las dos categorías. Luego, se considera el valor 1 o 0 según si el valor es mayor o igual a 0.5 o no, respectivamente.

Respecto al entrenamiento de este tipo de modelos, el procedimiento general es similar al analizado en el caso de la regresión lineal, con un pequeño cambio en relación a la función de error, la cual no puede ser el error cuadrático medio, porque se tendría una función con múltiples mínimos y sería muy probable que el método de descenso por gradiente no se detenga en el mínimo absoluto de la función sino en uno local. Además, se necesita considerar el error en un intervalo reducido pero conservando las proporciones, ya que si la salida esperada es 1 y se obtiene 0, el error no es realmente 1 sino infinito, ya que es lo más alejado que se puede estar de la salida esperada. De esta forma se usa una función de error conocida como entropía cruzada (cross-entropy) que se define matemáticamente como $error = -\frac{1}{N}\sum_{i=1}^{N}[D_i . ln(Y_i) + (1 - D_i).ln(1 - Y_i)]$ donde D corresponde a la salida deseada, Y a la obtenida durante el entrenamiento, N representa el número total de datos usados en la regresión y ln es el logaritmo natural.

En la figura 4 puede verse una representación gráfica de ambos modelos analizados previamente representados por líneas rojas discontinuas.

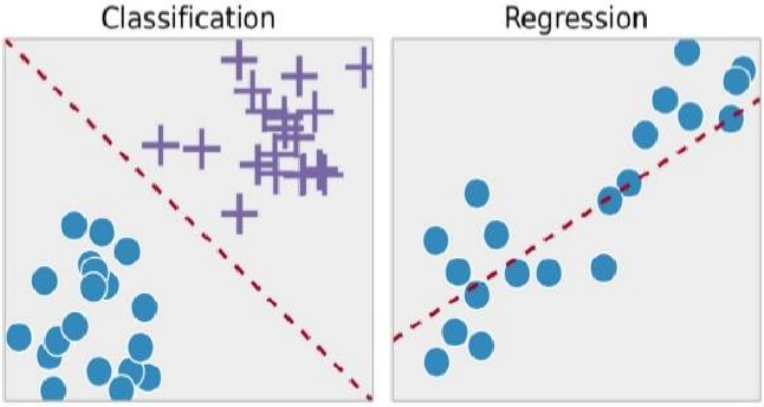


Figura 4: Regresión logística y lineal en problemas de clasificación y regresión. [5 p.9]

2.1.2 Datos de Entrenamiento y de Prueba

Hasta ahora se ha mencionado múltiples veces la existencia de parámetros a "entrenar" de manera que permitan representar correctamente las relaciones entre las entradas al sistema y sus salidas. Para poder utilizar dichos parámetros para predecir salidas de nuevas entradas nunca antes vistas.

Para el proceso de entrenamiento es necesario tener un conjunto de datos en base a los cuales el sistema puede aprender los parámetros necesarios. El conjunto original de datos disponibles se suele dividir en dos grupos: de entrenamiento (train set) y de prueba (test set). Los datos de entrenamiento se utilizan para actualizar dichos parámetros. Los datos de prueba se utilizan para evaluar la capacidad predictiva del modelo entrenado en datos no vistos anteriormente. Existe un tercer conjunto de datos que puede entrar en juego denominado conjunto de validación (validation set), el cual se utiliza para ajustar los hiperparámetros del modelo y evitar el sobreajuste al conjunto de entrenamiento.

Generalmente estos conjuntos de datos se establecen dividiendo un conjunto inicial en tres subconjuntos, siendo un 60% de los datos utilizados para entrenamiento, 20% para prueba y 20% para validación.

2.1.3 Métricas

Al momento de entrenar modelos, hace falta alguna medida para poder determinar su capacidad de cumplir con lo que se requiere de los mismos. Estas evaluaciones son fundamentales en el campo de machine learning en general, ya que se centran en analizar el error de los modelos y su desempeño en la tarea específica.

La medida más básica es el ratio de clasificación, que consiste en calcular el porcentaje de aciertos del modelo para los datos contemplados. El problema de esta métrica es que no tiene en cuenta el número de ejemplos de cada clase, por lo que no se puede reflejar fielmente la calidad del clasificador.

Por otro lado, existen métricas que abordan este problema, como la precisión y el recall. La precisión calcula el porcentaje de aciertos en las predicciones (que porcentaje de predicciones positivas fueron correctas) y puede considerarse como el ratio entre los los verdaderos positivos y todos los positivos (verdaderos positivos más falsos positivos). El recall calcula el porcentaje de aciertos en los ejemplos que realmente son de determinada clase, es decir que porcentaje de los casos positivos fueron capturados, siendo en este caso el ratio entre los verdaderos positivos y los verdaderos positivos más falsos negativos.

Por último existe una métrica conocida como F1-Score que combina precisión y recall, de manera de poder obtener un parámetro que puede indicar un buen valor para ambas métricas mencionadas previamente, siendo este un coeficiente entre los resultados capturados y los resultados relevantes. F1-Score se calcula como $F1\,Score = 2 * \frac{Precisión*Recall}{Precisión+Recall}$. Proveyendo un resultado entre 0 y 1, siendo 1 la indicación de un algoritmo perfecto, y 0 un algoritmo que ha fallado completamente en el recall, precision o ambos

En este apartado se ha explicado el concepto de Machine Learning, o aprendizaje automático: sistemas que, mediante algoritmos, pueden aprender automáticamente. En este trabajo se utilizarán sistemas de aprendizaje supervisado, específicamente sistemas entrenados para resolver problemas de clasificación. Además se han presentado conceptos clave durante la fase de entrenamiento, como la separación de datos en los conjuntos de entrenamiento, prueba y verificación, y métricas para poder determinar su capacidad de cumplir con lo que se requiere de los mismos.

2.2 Redes Neuronales y Deep Learning

En esta sección se muestra la estructura y funcionamiento básicos de las redes neuronales.

Las redes neuronales son modelos computacionales dentro del ámbito de machine learning en los que las computadoras procesan datos de una manera análoga o inspirada en la forma de procesar información que posee el cerebro humano. Este proceso, llamado aprendizaje profundo (deep learning) utiliza un conjunto de unidades interconectadas entre sí, neuronas artificiales, las cuales forman una estructura de capas en las que cada neurona perteneciente a una capa se encuentra conectada a las neuronas de la capa siguiente, a excepción de las pertenecientes a la última capa, o capa de salida, en la cual simplemente se encuentran los resultados obtenidos.

Estas redes existen desde los años 40s - 50s, pero debido a la potencia computacional y cantidades de datos que requieren no tuvieron mucho éxito en esa época, sin embargo, como hoy en día ambos recursos vieron un crecimiento exponencial en comparación a esos tiempos, este tipo de redes se considera una de las mejores técnicas para el ámbito de machine learning en relación a clasificación de imágenes o procesamiento de lenguaje natural.

En el funcionamiento básico de este tipo de redes, las neuronas están conectadas entre sí a través de enlaces, en los cuales la salida de una neurona previa es multiplicada por un valor de peso, lo cual puede aumentar o inhibir el estado de activación de las neuronas adyacentes. Al mismo tiempo, a la salida de la neurona puede existir una función de activación, que puede modificar el resultado obtenido, o imponer un límite que no se puede sobrepasar, antes de transferir la información a otra neurona.

En general, puede verse a la red neuronal como una función de funciones que permite procesar grandes cantidades de información, detectando relaciones entre los datos, y tras realizar un entrenamiento como los mencionados previamente, modificando pesos que afectan a los parámetros de entrada con el objetivo de minimizar una función de error, conseguir un sistema capaz de realizar predicciones correctas para datos de entrada previamente desconocidos. Siempre teniendo en cuenta, que el entrenamiento tiene que ser de la red como conjunto completo, ya que no es posible entrenar cada neurona individualmente, porque al estar interconectadas, la salida de una se convierte en la entrada de otra.

Cada neurona actúa como un clasificador con sus propios parámetros y función de activación, siendo una de las más utilizadas la función ReLU: $ReLU(x) = max(0, x)$.

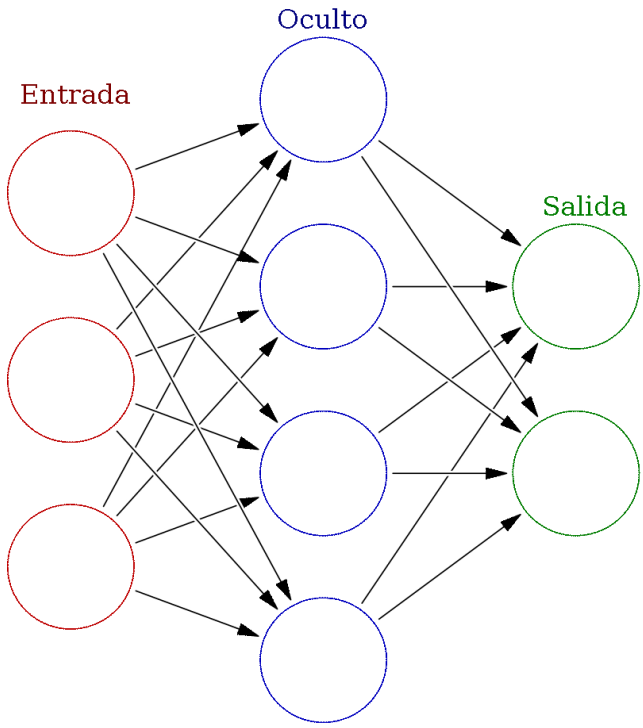


Figura 5: Diagrama de una red neuronal

En la figura 5 puede observarse un ejemplo de una red neuronal con una única capa oculta, cada nodo circular representa una neurona artificial y cada flecha representa una conexión desde la salida de una neurona a la entrada de otra. Se denomina capa oculta a aquellas conectadas por la entrada y salida con otras capas. Puede verse que la

arquitectura de esta red es totalmente conectada o “fully connected”, es decir, cada neurona está conectada a cada neurona de la capa siguiente.

A mayor cantidad de neuronas y capas, mayor capacidad de aprendizaje y predicción, pero al mismo tiempo se ve incrementada su complejidad y dificultad para ser entrenada y entendida, ya que los recursos computacionales y datos necesarios se ven incrementados en gran medida. A estas redes con múltiples capas intermedias se las considera en el marco del aprendizaje profundo (deep learning).

Para este tipo de redes con múltiples capas ocultas el entrenamiento se vuelve más complejo, ya que no es posible actualizar todos los parámetros o “pesos” del modelo al mismo tiempo, sino que tiene que hacerse por etapas, actualizando los de cada capa una a la vez. Para esto se utiliza el algoritmo de Backpropagation, en el cual se aplica la regla de la cadena para minimizar el error de la red, intentando conseguir que el gradiente de la función de error con respecto a cada peso sea 0, aprovechando los cómputos de gradientes anteriores para calcular los actuales, aunque recorriendo la red en sentido inverso, desde las últimas capas hacia las primeras.

De esta forma, se ve que las redes neuronales poseen un conjunto de características comunes en todas las implementaciones: son un conjunto de unidades interconectadas entre sí que forman una estructura de capas, tienen que entrenarse como conjunto, pueden procesar grandes cantidades de información, y requieren un funcionamiento en etapas para modificar parámetros internos a modo de reducir el error en sus predicciones.

2.3 Entrenamiento

A continuación se presentan una serie de aspectos, como variables, métodos e hiperparámetros, a tener en cuenta durante el entrenamiento de las redes.

Teniendo ya los datos a utilizar en el entrenamiento de la red correctamente procesados y separados, y habiendo elegido la arquitectura de la red, el objetivo de este proceso es minimizar la función de error que nos permita ajustar el modelo a los ejemplos del conjunto de entrenamiento. Esta no es una tarea sencilla debido a la cantidad de parámetros que se tienen que actualizar durante cada paso del proceso y la cantidad de información que debe manejarse.

Existen una serie de aspectos clave a considerar en la fase de entrenamiento, los cuales afectan la capacidad de la red de llevar a cabo su trabajo de manera tanto eficaz como eficiente, de los que se hablará a continuación.

2.3.1 Inicialización de pesos:

Al inicio de la etapa de entrenamiento de redes neuronales se tienen que establecer los valores iniciales de los pesos. Estos se suelen inicializar en valores aleatorios cercanos a 0, ya que en caso contrario no se verían modificados nunca[6].

En general, se elige un método de inicialización de pesos acorde a la función de activación que se vaya a utilizar, ya que eso permite minimizar las dos tendencias más comunes de los gradientes, como el desvanecimiento a medida que se avanza en el proceso de backpropagation o el aumento de forma desproporcionada.

2.3.2 Regularización:

La regularización es una técnica que ayuda a prevenir el sobreajuste, el cual ocurre cuando una red neuronal aprende demasiado de los datos de entrenamiento y no generaliza bien a los nuevos datos. Es una forma de agregar algunas restricciones o penalizaciones al modelo, y si bien existen diferentes métodos, todos tienen como objetivo reducir la varianza del modelo (cuán sensible es el modelo a pequeños cambios en los datos) y aumentar el sesgo (que tán lejos está el modelo de la verdadera relación)

Un buen modelo debe tener baja varianza y bajo sesgo, pero generalmente hay una compensación entre ellos. La regularización ayuda a encontrar un equilibrio entre ellos reduciendo o podando los parámetros del modelo, agregando ruido o caída a las capas, o aumentando los datos con transformaciones.

Entre los métodos de regularización se encuentran L1 y L2, también conocida como decaimiento de peso o regularización de lazo y de cresta respectivamente, donde se agrega un término a la función de pérdida que penaliza los pesos grandes en el modelo. L1 tiende a hacer que algunos pesos sean 0, mientras que L2 hace que todos los pesos sean más pequeños.

En L1:

$$\Sigma\left(y-\widehat{y}\right)^2+\lambda\left(|m_1|+|m_2|+.....+|m_n|\right)$$

En L2:

$$\Sigma\left(y-\widehat{y}\right)^2+\lambda\left(m1^2+m2^2+.....+mn^2\right)$$

Siendo Ŷ la variable de predicción, m1 … mn las pendientes y λ un parámetro que indica cuánto se quiere penalizar a los coeficientes, pudiendo tener cualquier valor mayor a 0.

Otra técnica es el abandono, o dropout, que elimina aleatoriamente algunas unidades o neuronas en las capas ocultas durante el entrenamiento en las diferentes iteraciones, evitando la coadaptación de las características.[7]

2.3.3 Batch size:

Este parámetro indica el tamaño de un subconjunto de los datos de entrada que serán utilizados para entrenar a la red, ya que normalmente la cantidad de datos disponibles son demasiados para ser utilizados al mismo tiempo.

Si se tienen 1000 datos de entrenamiento, y se utiliza un batch size de 100, hará falta pasar en total 10 subconjuntos de datos para que haya pasado todo el conjunto de

entrenamiento por la red, lo que se considera una época, y como el proceso de minimización de error es iterativo, hacen falta múltiples épocas para entrenar a la red.

El batch size es un hiperparámetro más del modelo. Generalmente si el conjunto de entrenamiento no es muy grande (<=2000) conviene que el tamaño del batch sea igual a dicho conjunto, pero en caso contrario se suelen utilizar potencias de 2, y se tiene que tener en cuenta que la cantidad de información en cada batch tiene que caber en la memoria de la CPU/GPU para beneficiarse de la ganancia en eficiencia.

2.3.4 Ratio de aprendizaje:

El ratio de aprendizaje **α** es un hiperparámetro que controla la velocidad del proceso de aprendizaje. Este debe tener un valor que permita una convergencia, y que sea rápida. Si se elige un valor muy grande de **α** entonces el método utilizado oscilará en torno a un punto de error mínimo pero no convergerá, mientras que si se utiliza un valor muy pequeño, el aprendizaje resultará extremadamente lento.

Para cada problema el ratio de aprendizaje idóneo es diferente, y no existe una manera de conocerlo de antemano, por eso existen técnicas que proponen que no sea un valor prefijado, sino que decrezca conforme nos acercamos al mínimo buscado (Learning rate Decay)

2.3.5 Métodos de optimización:

Como se mencionó previamente, en el entrenamiento de redes neuronales se busca minimizar la función de error, normalmente utilizando el método de descenso por gradiente, sin embargo, este no es el único método de optimización disponible.

Los optimizadores pueden ser tan sencillos como restar los gradientes de los pesos, o pueden ser funciones complicadas. Los mejores optimizadores se centran en ser más rápidos y eficientes pero también suelen generalizar mejor, reduciendo el sobreajuste. Por esto, es posible que la elección de un optimizador pueda influir dramáticamente en el rendimiento del modelo.

A continuación se presentan algunos de los optimizadores más utilizados.

SGD

Stochastic Gradient Descent (Descenso estocástico por gradiente) se refiere al caso de aplicar el descenso por gradiente básico cuando el batch size es 1, sustrayendo de los pesos el gradiente multiplicado por el ratio de aprendizaje.

Al empezar una nueva época se mezcla aleatoriamente los ejemplos del conjunto de entrenamiento

Momentum

En este método los pesos son modificados a través de un término de momento, el cual es calculado como la media móvil de los gradientes. Se recuerda el incremento

aplicado a los pesos en cada iteración y se determina la siguiente actualización como una combinación lineal entre el gradiente y esos incrementos anteriores. De esta forma se suavizan las oscilaciones en torno al mínimo durante la convergencia del algoritmo de descenso por gradiente.[5 p.22]

RMSProp

RMSProp o Root Mean Square Propagation es una modificación de Stochastic Gradient Descent en la que se utilizan diferentes tasas de aprendizaje para las variables teniendo en cuenta el gradiente acumulado en cada una de ellas, pero utilizando el concepto de "ventana" para considerar solo los gradientes más recientes, aplicándose a éstos una media exponencial ponderada para suavizar los cambios aplicados a los parámetros. Este enfoque puede ayudar a evitar que las tasas de aprendizaje se vuelvan excesivamente pequeñas.[8]

Adam

Este optimizador combina RMSProp y Momentum, almacenando el ratio de aprendizaje individual de RMSProp y el promedio pesado de momentum.

2.3.6 Batch Normalization:

Es una técnica que se emplea en Deep Learning para agilizar el proceso de aprendizaje[9].

Batch Normalization pretende reducir el cambio de covariables internos, lo que hace a la red más robusta ante malas inicializaciones. El cambio de covariables internos se define como el cambio en la distribución de las activaciones de las redes, debido a que la distribución de los datos de entrada es diferente entre batches. Cuando menor sea esta diferencia entre batches, más similares serán los datos que llegan a los filtros de la red, más parecidos los mapas de activación también, y, además, mejor funcionará el entrenamiento de la red.

Esto lo consigue forzando las activaciones de la red a tener un valor escogido de una distribución gaussiana unitaria al principio del entrenamiento. Este proceso es posible gracias a que la normalización es una operación diferenciable.

Matemáticamente se centra y normaliza cada batch que llega a la red con una media y desviación estándar calculadas con el batch, para luego re-escalar y descentrar los datos de nuevo con parámetros aprendidos por la red a través del entrenamiento.

Además, como se calcula la media y desviación típica para cada batch, en vez de para todo el dataset, también se introduce cierto ruido que actúa como regularización y ayuda a reducir el sobreajuste.

Esta técnica se ha mostrado muy eficiente para entrenar redes más rápidamente (necesitando menos épocas)[10].

Puede verse que existe una gran cantidad de variables e hiperparámetros a tener en cuenta al momento de entrenar una red neuronal los cuales afectan a la capacidad de la red de realizar su trabajo además de la eficiencia con la que lo realiza, por desgracia no existe una fórmula precisa para seleccionar los parámetros de manera óptima, requiriendo muchas veces prueba y error hasta encontrar una combinación que funcione bien para realizar la tarea deseada.

En base a lo analizado en este apartado, es importante tener en cuenta que en el caso de procesamiento de imágenes cada píxel de la imagen de entrada está conectado a cada neurona de la primera capa oculta, y con el tamaño y resolución de imágenes manejado actualmente la cantidad de pesos con los que se tiene que trabajar vuelven esto inviable. Es por eso que en aplicaciones de procesamiento de imágenes, se prefieren arquitecturas de red más especializadas y eficientes, como las redes neuronales convolucionales. En la siguiente sección, explicaremos cómo se representan las imágenes en escala de grises y en color, y luego profundizaremos en el funcionamiento de las redes convolucionales para el procesamiento de este tipo de imágenes.

2.4 Imágenes

Las imágenes son representadas por el valor de cada uno de sus píxeles como un arreglo bidimensional. En la figura 6 puede observarse una imagen en escala de grises. Abajo en la misma imagen pueden verse los valores de cada uno de los píxeles que corresponden a una sección de la imagen formando una matriz bidimensional que representa la información contenida en dicha imagen. Al lado de la imagen puede verse la escala que representa los diferentes niveles de intensidad de los píxeles, iniciando en 0 (ausencia total de intensidad, es decir negro) hasta 255 (presencia total de intensidad, blanco).

90	67	68	75	78	98	185	180	153	139	132	106	70	80	81	69	69	67	35	34
92	87	73	78	82	132	180	152	134	120	102	106	95	75	72	63	75	42	19	29
63	102	89	76	98	163	166	164	175	159	120	103	132	96	68	42	49	46	17	22
45	83	109	80	130	158	166	174	158	134	105	71	82	121	80	51	12	50	31	17
39	69	92	115	154	122	144	173	155	105	98	86	82	106	83	76	17	29	41	19
34	80	73	132	144	110	142	181	173	122	100	88	141	142	111	87	33	18	46	36
37	93	88	136	171	164	137	171	190	149	110	137	168	161	132	96	56	23	48	49
66	117	106	147	188	202	198	187	187	159	124	151	167	158	138	105	80	55	59	54
127	136	107	144	188	197	188	184	192	172	124	151	138	108	116	114	84	46	67	54
143	134	99	143	188	172	129	127	179	167	106	118	111	54	70	95	90	46	69	52
141	137	96	146	167	123	91	90	151	156	121	93	78	82	97	91	87	45	66	39
139	137	80	131	162	145	131	129	154	161	158	149	134	122	115	99	84	35	52	30
137	133	56	104	165	167	174	181	175	169	165	162	158	142	124	103	67	19	31	23
135	132	65	86	173	186	200	198	181	171	162	153	145	135	121	104	53	14	15	33
132	132	88	50	149	182	189	191	186	178	166	157	148	131	106	78	28	10	15	44

Figura 6: Imagen en escala de grises

En el caso de imágenes a color, el formato más utilizado es conocido como R, G y B (*Red*, *Green* y *Blue*). De esta forma, una imagen se puede representar como un arreglo tridimensional (m,n,c) donde m representa las filas en la imagen y n las columnas determinadas por la dimensión de la imagen. Mientras que c es una variable que determina el canal (R,G o B).

En la figura 7 puede observarse una imagen a color y debajo de esta los resultados de extraer sus canales R, G y B en ese orden. Cada uno de estos canales se representa mediante valores enteros, siguiendo una estructura similar a la explicada para las imágenes en escala de grises.

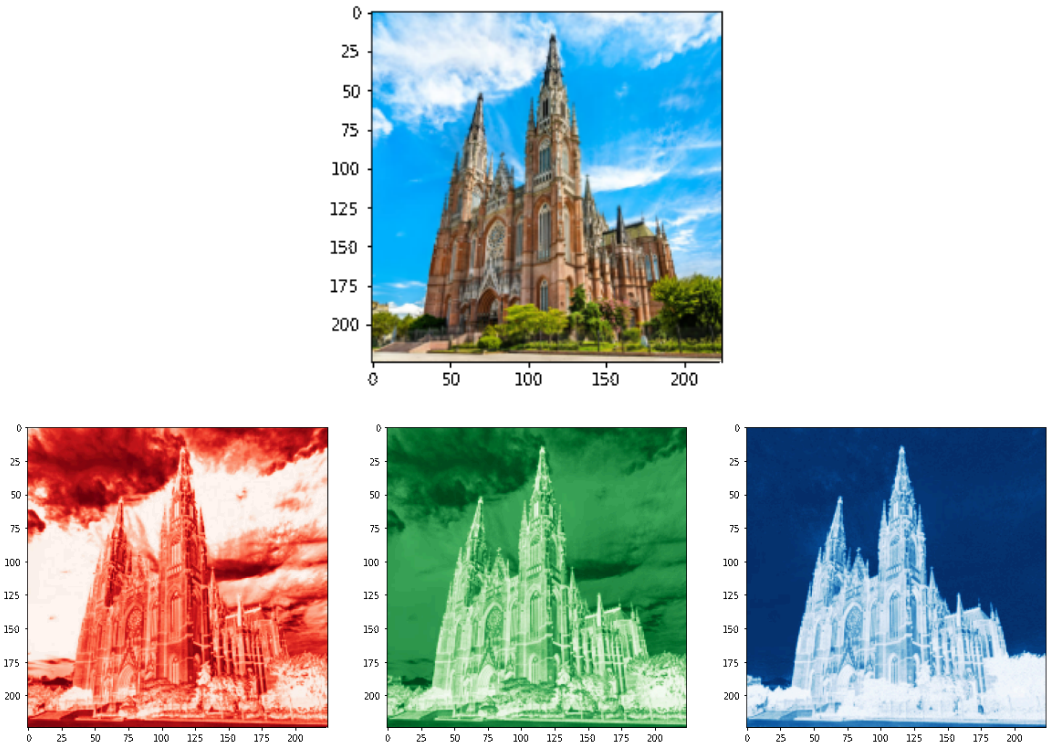


Figura 7: Imagen a color y la extracción de sus canales rojo, verde y azul

Al tratar con imágenes a color la manipulación puede volverse más compleja debido a la necesidad de trabajar con múltiples canales. En estos casos convertir las imágenes a escala de grises puede servir para mantener los valores de intensidad de los píxeles, pero utilizando una matriz bidimensional simplificando su procesamiento. Este tipo de imágenes usualmente capturan la luminancia o intensidad de cada pixel en base a los valores de los canales R,G y B.[11]

2.5 Redes Neuronales Convolucionales (CNNs)

En las redes neuronales convolucionales[12] (CNN por sus siglas en inglés, Convolutional Neural Network) las neuronas artificiales corresponden a campos receptivos

de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico. Este modelo surge por la importancia de la visión por computador en el mundo del Deep Learning, siendo muy utilizadas para tareas de clasificación y “computer vision” debido a su eficacia en tareas como reconocimiento de patrones, detección de objetos o segmentación de imágenes.

2.5.1 Estructura de las redes convolucionales

En las redes neuronales convolucionales (CNN), se utilizan dos tipos principales de capas: las capas convolucionales y las capas de pooling (o agrupación). Estas capas se alternan en la arquitectura de la red (como puede verse en la figura 8), lo que permite una reducción en la cantidad de parámetros (pesos) que deben ser procesados en las capas siguientes. Esta reducción en la dimensionalidad permite utilizar capas fully connected (completamente conectadas) al final de la red, y que estas pueden operar de manera más eficiente en términos de recursos computacionales, ya que la cantidad de parámetros es considerablemente menor.

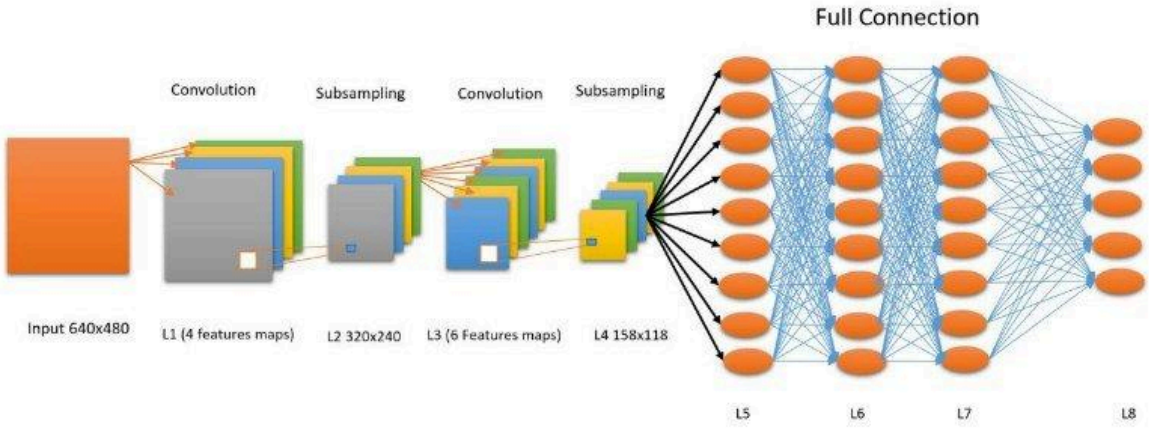


Figura 8: Diagrama de una red neuronal convolucional [13]

2.5.1.1 Capa convolucional

Estas capas son donde se realizan la mayoría de los cálculos, y requieren de datos de entrada y filtros, para luego producir un resultado denominado mapa de características.

La entrada en estas capas suele ser una imagen de tamaño m x n x c donde m y n son el ancho y alto de la imagen, y c la cantidad de canales (3 en las imágenes rgb), y sobre estos datos es donde el o los filtros (o detectores de características, o kernel) realizan la convolución. Estos filtros de tamaño f x f x c (notar que el número de canales de un filtro debe coincidir con el número de canales de la entrada), durante el proceso de convolución , se aplican a un área de la imagen y se calcula un producto escalar entre los valores de los pixeles de entrada y el filtro, el cual es introducido en una matriz de salida, acto seguido, el filtro se desplaza en una distancia denominada “stride” y repite el proceso hasta que el

kernel haya recorrido toda la imagen, de esta manera se obtiene como resultado final un conjunto de datos denominado mapa de activación, el cual se puede considerar como una nueva imagen con un alto y ancho menor al de la imagen de entrada original y una cantidad de canales igual a la cantidad de filtros empleados en la capa convolucional particular (también se los puede considerar como múltiples mapas de características de un solo canal).

Una característica adicional que se puede modificar en este proceso es el padding, que consiste en rellenar con 0 los bordes de la entrada, lo que permite controlar el tamaño que se desea que tenga la salida resultante. Con este hiperparámetro se indica la cantidad de filas o columnas de ceros a añadir alrededor de los bordes de la entrada.

Puede verse un diagrama de este proceso en la figura 9, realizando una convolución en una imagen RGB a la que se le aplica un padding de una fila y una columna.

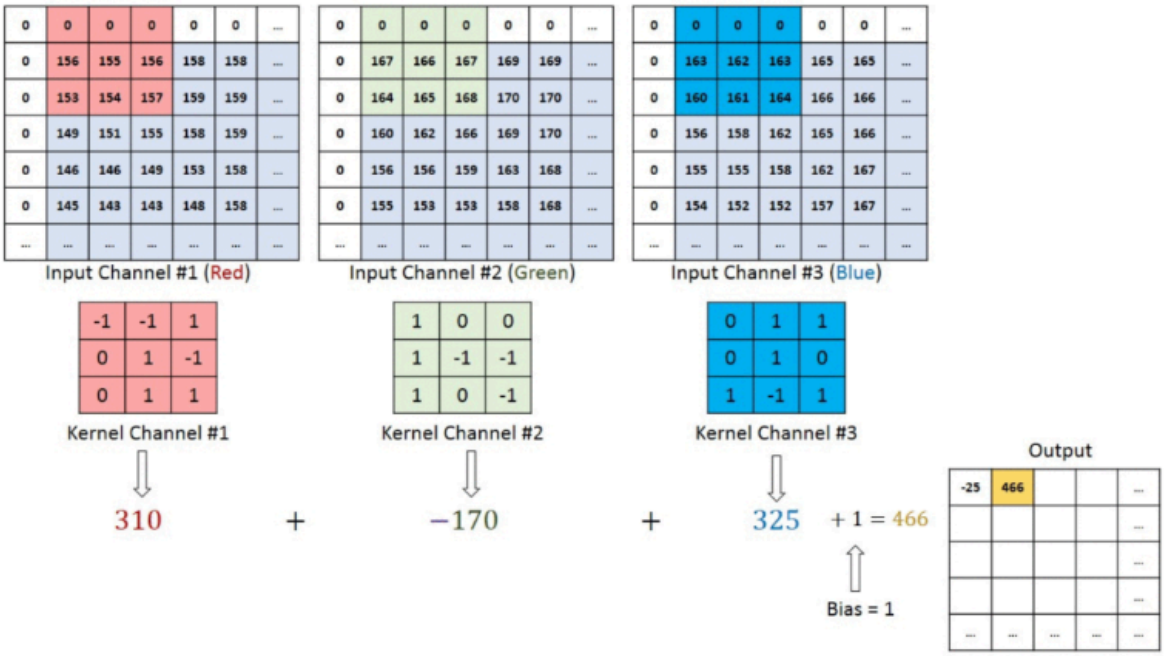


Figura 9: Convolución de una imagen RGB (con 3 capas), a la que se le ha aplicado un padding de una fila y una columna [14]

La nueva imagen generada por los filtros pasará a ser la entrada de la siguiente capa luego de que se le haya sumado un bias o término independiente y haber aplicado una transformación mediante una función de activación (que como en las redes convencionales, suele ser la ReLU) la cual introduce la no linealidad en el modelo, para de esta manera obtener lo que se conoce como “mapa de activaciones”.

2.5.1.2 Capa de agrupación

También conocida como capa de pooling, submuestreo o subsampling, permite reducir la dimensión mediante la reducción del número de parámetros de entrada. En esta capa se recorre la entrada con un filtro, de manera similar a la capa convolucional, pero en

este caso se utiliza un filtro sin pesos, en su lugar el kernel aplica una función de agregación a los valores dentro del campo receptivo.

Hay dos tipos principales de agrupación:

- Máxima, o “max pooling” en la que a medida que el filtro recorre la entrada, selecciona el píxel con el valor más alto para enviarlo a la matriz de salida
- Media, o “mean pooling” en la que el filtro avanza calculando el promedio dentro del campo receptivo, para luego enviarlo a la matriz de salida

Si bien esta capa causa que se pierda información, también presenta una serie de beneficios para la red neuronal convolucional, ya que ayuda a reducir la complejidad, mejora la eficiencia y reduce el riesgo de sobreajuste.

2.5.1.3 Capa totalmente conectada

Esta capa es como las mencionadas previamente, cada neurona está conectada con todas las neuronas de la capa siguiente, de esta manera puede realizar la tarea de clasificación basándose en las características que las capas anteriores extrajeron.

Una diferencia a tener en cuenta es que las capas previas usan en su mayoría funciones de activación ReLU, mientras que las capas totalmente conectadas suelen utilizar una función de activación softmax para clasificar las entradas adecuadamente y generar probabilidades entre 0 y 1.

Estas redes poseen una gran capacidad para identificar características en imágenes, ya que a medida que la información entra en las capas más profundas, los filtros por los que los datos atraviesan pueden detectar características cada vez más complejas, pudiendo verse que en los primeros filtros solo se detectan rasgos como dirección y color, mientras que en los más profundos se pueden detectar estructuras cada vez más complejas, pasando por texturas y patrones en los filtros intermedios, hasta características complejas que se asemejan a partes de objetos, como puede observarse en la figura 10.

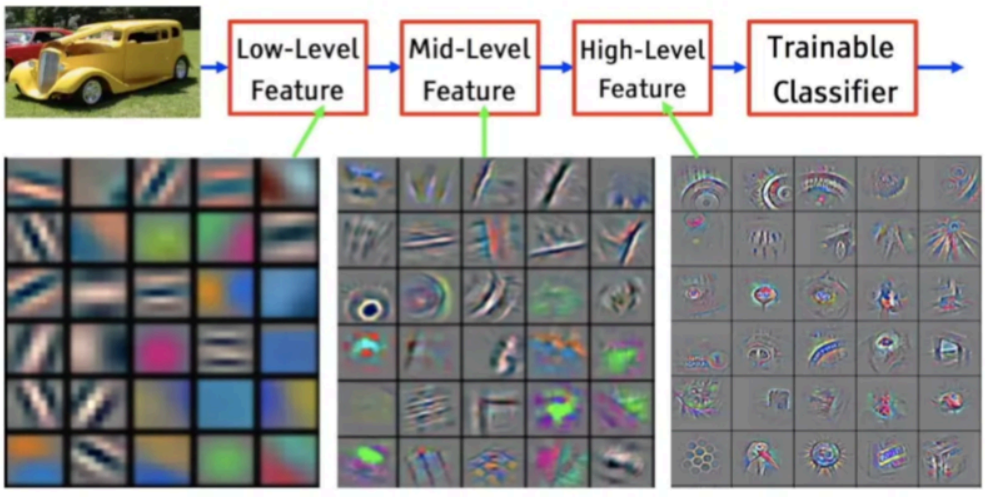


Figura 10: Mapas de características en diferentes etapas de la red [15]

Esta no es la única característica de las redes neuronales convolucionales que las hacen útiles para trabajar sobre imágenes, ya que hay dos propiedades que pueden considerarse esenciales para este trabajo: la conservación de la espacialidad y eficiencia computacional.

La primera de estas características se debe a la utilización de matrices en lugar de datos aislados, cosa que ocurre en las redes completamente conectadas. De esta manera se conserva a lo largo de todo el proceso la información sobre la localidad de las activaciones calculadas, lo cual es importante al trabajar con imágenes ya que la localidad de los pixeles es igual de importante que las relaciones entre estos, y es información que normalmente se pierde en el procesamiento de los datos al trabajar con otros tipos de redes neuronales.

Respecto a la eficiencia computacional, hay nuevamente dos elementos que afectan a esta característica: Por un lado. el algoritmo puede aprovechar el hecho de que en las imágenes suelen existir características que se repiten en diferentes regiones de la misma, utilizando los mismos pesos para computar activaciones en zonas diferentes, lo cual puede verse en el deslizamiento de la misma matriz de convolución por toda la imagen, de esta manera se reducen las conexiones y parámetros a entrenar.

Por otra parte, los valores de salida dependen de un conjunto reducido de valores de entrada, por lo que no se establecen conexiones entre cada valor de entrada y de salida, lo que nuevamente reduce la cantidad de parámetros a entrenar.

Es interesante notar que estos últimos puntos, referentes a la eficiencia computacional, están muy relacionados al funcionamiento de las capas de pooling explicado previamente.

Como resumen de lo visto a lo largo de este capítulo, este tipo de redes neuronales presentan una estructura similar a las neuronas en la corteza visual primaria de un cerebro biológico y son especialmente útiles para el procesamiento de imágenes debido a la conservación de la espacialidad y eficiencia computacional.

Estas redes tienen como una de sus principales características la presencia de capas de convolución, donde se realizan la mayoría de los cálculos y capas de agrupación, que ayudan a reducir la complejidad y mejorar la eficiencia.

Si bien las CNN permiten realizar diferentes tipos de análisis sobre imágenes, debido a su complejidad no es posible seguir un paso a paso y entender el cómo llegan a sus conclusiones (se comportan como cajas negras). Debido a esto, en el próximo capítulo explicaremos métodos de visualización que nos permitirán comprender e interpretar los resultados obtenidos por este tipo de redes.

Capítulo 3: Visualización de Redes Neuronales Convolucionales

En este capítulo se presentan métodos de visualización de redes neuronales convolucionales como concepto, explicando qué son, por qué es posible su funcionamiento y que información permiten obtener. Luego se muestran múltiples métodos de visualización explicando cómo funcionan y por último se presentan arquitecturas de redes neuronales convolucionales consideradas para analizar en este trabajo con dichos métodos.

El rendimiento de las redes neuronales convolucionales (CNN) se debe principalmente a su capacidad para aprender características de las imágenes de entrada. Sin embargo, es difícil para un humano identificar e interpretar dichas características y el funcionamiento del mecanismo interno de las redes en sí, debido a esto se las considera como “cajas negras”.

La interpretabilidad de una CNN está ligada a la capacidad del ser humano de comprenderla, la cual puede mejorarse demostrando las características internas que aprenden estas redes. La visualización contribuye en gran medida a esto, siendo un método cualitativo que permite analizar el funcionamiento interno de la red mediante la traducción de sus características internas a patrones visuales perceptibles. Esto es posible debido a que las características internas de las redes neuronales convolucionales se basan en el córtex visual humano, el cual procesa las características de los objetos percibidos de manera jerárquica a través de diferentes zonas neuronales [16]

Primero, en el área neuronal de más bajo nivel visual se detectan las características visuales más básicas como bordes o líneas y posteriormente en las más altas se empiezan a detectar características más complejas como formas u objetos, hasta llegar a detectar imágenes completas.

En comparación, las redes neuronales convolucionales comienzan extrayendo pequeñas características como bordes en la primera capa y a medida que se avanza por las capas más profundas se extraen características cada vez más complejas (formas y objetos parciales), hasta llegar a la clasificación final en las capas “fully connected”. De esta forma comparando las neuronas cerebrales visuales de un humano y las de las CNNs, la visualización permite observar las funcionalidades de cada componente en las redes neuronales convolucionales, como por ejemplo, que es lo que aprende cada neurona.

Al emplear el método de visualización es posible intuir lo que hace el modelo y cómo toma decisiones en las predicciones, lo cual también permite averiguar fallos en el mismo e intuir por qué fallan o explicar sus decisiones a clientes (usuarios en general), y ayuda a elegir los hiperparametros más adecuados o la arquitectura ideal para el problema concreto que se tenga.

Existen múltiples métodos de visualización de CNNs. Todos ellos tienen el objetivo de demostrar el campo receptivo de una neurona, resaltando las secciones de una imagen que más estimulan a la neurona o conjunto de ellas, de manera de poder observar qué patrones de dicha imagen consiguen una mayor activación. Básicamente, ver qué patrones

visuales tienen impacto en el mapa de activaciones de dicha neurona. De esta forma es posible averiguar, en los casos más puntuales por ejemplo, si la neurona detecta bordes, formas, objetos, etc.

Si bien se puede investigar respecto a las capas intermedias de la red para diseccionarla y posiblemente descubrir aspectos nuevos, suele ser más habitual el visualizar neuronas de salida. Estas neuronas computan la clasificación final, que en general suele ser lo más importante y no tanto el proceso completo para conseguirla. Esto se debe a la necesidad de poder explicar una decisión del clasificador en el caso de que no resulte convincente, o si un experto utiliza estos algoritmos para tomar decisiones y requiere evidencias que las respalden, o poder explicar resultados a un cliente.

Visualizar las predicciones sirve para analizar en qué partes de la imagen de entrada se fundamenta la decisión. Esto permite verificar que la red haya aprendido correctamente, por ejemplo, si en el entrenamiento se usan imágenes de lobos y todas tienen un fondo de nieve, es probable que la red tenga una buena precisión durante el entrenamiento, pero que en realidad no se esté enfocando en los animales, sino en el fondo blanco. Esto suele ocurrir en casos donde un elemento suele encontrarse en un entorno determinado, causando que la red clasifique por el entorno y no por el elemento en sí.

La principal diferencia entre visualizar neuronas de salida o de capas ocultas es que las de salida provienen de capas fully connected, y su salida es un valor unidimensional, mientras que la otras son convolucionales y calculan un valor multidimensional (el mapa de activaciones). Debido a esto, el detectar los patrones de una imagen que causan un mayor valor de activación a una neurona es más fácil en el primer caso, ya que consisten de un único valor, mientras que en el caso de las neuronas de capas ocultas hay que utilizar alguna función de agregación (máximo, media, suma, etc) en su mapa de activaciones.

El problema de visualización se puede abordar de dos formas. Por un lado, se puede generar la imagen que represente las características que más estimulen a la neurona objetivo, y por otro lado, se puede destacar en una imagen concreta las regiones o patrones que más afectan a la decisión de dicha neurona. Esta última forma es la más común y práctica, ya que en general resulta de mayor interés entender el motivo de la predicción de una neurona que identificar su estímulo preferido.

A continuación se explican algunos métodos que funcionan de la segunda forma, destacando las secciones de una imagen que más afectan a la decisión de la red, ya que permiten explicar ese resultado y ese es el objetivo de este trabajo.

3.1 Mapeo de activación de clase (CAM)

Esta técnica, propuesta por Zhou et al. en 2015 para visualizar clases o neuronas de salida [17], genera mapas de calor que resaltan regiones de imágenes que son específicas respecto a una clase. De esta manera, proporciona una representación visual de las regiones de la imagen que más influyen a la decisión tomada por una CNN para realizar una clasificación particular, es decir, el CAM para una clase particular indica las regiones discriminativas de la imagen usadas por la CNN para identificar dicha clase.

Este método requiere el uso de una red neuronal con una característica particular, que es la presencia de una capa GAP (*Global Average Pooling*) previa a la capa *fully connected* cuya salida son las decisiones de clasificación.

En el funcionamiento normal de una red con GAP, esta capa transforma un mapa de características en un único número tomando el promedio de los valores dentro de ese mapa. Entonces si se tuvieran K=512 mapas (donde cada mapa corresponde a la salida de uno de los 512 filtros), se terminaría con K=512 números. Este proceso consiste en sumar todos los elementos de un mapa y dividir el resultado por el total de elementos en dicho mapa.

En general, luego del GAP se obtienen K números (siendo K la cantidad de filtros) los que se utilizan para tomar las decisiones de clasificación usando una única capa *fully connected*. Para esto, cada número es enviado a cada clase de salida siendo previamente multiplicado por un peso, que puede ser diferente para cada una de estas conexiones. Finalmente se suman, para cada clase de salida, los K números entrantes multiplicados por sus pesos correspondientes.

En el caso de CAM, en vez de utilizar los pesos para multiplicar cada número de K de forma individual, se utilizan para multiplicar los valores de los mapas de características completos (se multiplica cada elemento espacial del mapa por el peso conseguido). Es decir, si antes el mapa M2 generaba un número K2 el cual se multiplicaba por un peso W2-1 al conectarse al nodo que detecta la clase 1, ahora nos interesa el resultado de M2 multiplicado por W2-1 directamente, y tras realizar este procedimiento para cada mapa de activación, se suman todos los resultados para obtener la salida deseada. En la figura 11 puede observarse un diagrama de este proceso, donde se realiza la combinación lineal de los mapas de activaciones de la última capa convolucional, utilizando pesos que dependen de la importancia de cada mapa en la predicción final.

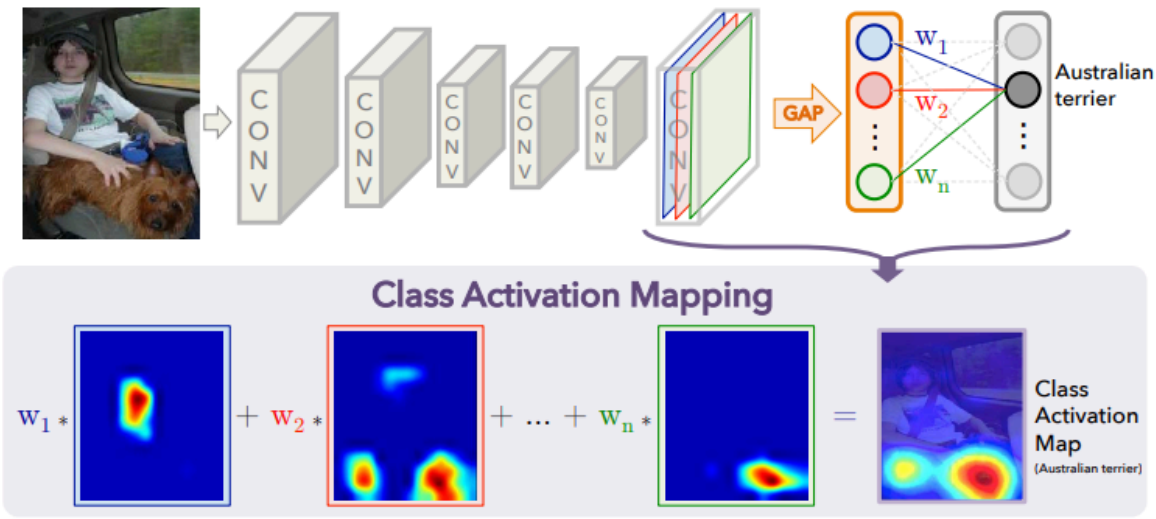


Figura 11: Diagrama del método CAM.[17]

De esta forma la salida del CAM es una grilla de números, los cuales corresponden al mapa de calor que buscamos.

Para una imagen dada, si $f_k(x,y)$ representa la activación de la unidad k en la última capa convolucional, en la ubicación espacial (x,y). Entonces para la unidad k el resultado del GAP es $F_k = \sum_{x,y} f_k(x,y)$ (ignorando la división por la cantidad de elementos del GAP, ya que no afecta esta explicación). Entonces, para una cierta clase c, la entrada a la capa softmax es $S_c = \sum_k w_k^c F_k$ donde w_k^c es el peso correspondiente a la clase c para la unidad k, básicamente indicando la importancia de F_k para la clase c.

Podemos definir M_c como el mapa de activación de clase para una clase c, donde cada elemento espacial está dado por $M_c(x,y) = \sum_k w_k^c f_k(x,y)$. De esta forma la entrada a la capa softmax es $S_c = \sum_{x,y} M_c(x,y)$, por lo que $M_c(x,y)$ indica directamente la importancia de la activación en la grilla espacial (x,y) llevando a la clasificación de una imagen hacia la clase c.

f_k puede verse como el mapa de la presencia de patrones visuales en el campo receptivo de una unidad los cuales activan dicha unidad. El mapa de activación de clase es simplemente una suma lineal ponderada (combinación lineal) de la presencia de estos patrones visuales en diferentes ubicaciones espaciales, y al hacer un *upsampling* del CAM hacia el tamaño de la imagen original de entrada, se pueden identificar las regiones de la imagen más relevantes para la categoría particular. El *upsampling* es el proceso inverso al realizado por las capas de agrupación o pooling layers. En su forma más sencilla, el proceso sería simplemente redimensionar la imagen obtenida copiando píxeles las veces que haga falta hasta obtener una imagen de las dimensiones deseadas.

3.2 Mapeo de activación de clases ponderado por gradiente (Grad-CAM)

Un inconveniente del método CAM es que requiere mapas de características ubicados justo antes de las capas softmax, lo que restringe su aplicabilidad a un grupo particular de arquitecturas de CNNs que realicen un *Global Average Pooling* (GAP) sobre los mapas convolucionales inmediatamente antes de la predicción. Estas arquitecturas pueden no cumplir con los requerimientos necesarios para nuestros objetivos, y adaptar otras arquitecturas para poder utilizar el método CAM podría ser muy costoso en términos de tiempo y recursos. Por estas razones, Selvaraju et al.[18] en 2016 proponen el método Grad-CAM que elimina estas restricciones y facilita la combinación de mapas de características sin necesidad de realizar ningún cambio a la arquitectura de una red neuronal.

Grad-CAM permite obtener los mismos resultados que el método CAM pero sin la necesidad de la existencia de una capa de GAP previa a la capa *fully connected* cuya salida son las decisiones de clasificación, ya que simula su funcionamiento de manera externa a la red en sí.

Como se mencionó previamente, en las redes neuronales convolucionales, las capas más profundas capturan elementos visuales de un nivel superior, y las capas convolucionales conservan información espacial que se pierde en capas completamente conectadas, por esto es razonable esperar que las últimas capas convolucionales tengan el mejor equilibrio entre semántica de alto nivel e información espacial detallada. Las neuronas en esas capas buscan información semántica específica para una clase en una imagen (como partes de objetos). Grad-CAM utiliza la información del gradiente que se dirige hacia la última capa convolucional de la CNN para asignar valores (pesos) de importancia a cada neurona para una decisión particular. Sin embargo, es importante señalar que Grad-CAM no se limita únicamente a estas neuronas finales, sino que nada impide utilizarlo para visualizar cualquier otra neurona de la red.

Para obtener el Grad-CAM primero se debe computar el gradiente de la salida para la clase c, y^c (antes del softmax), respecto a las activaciones del mapa de características A^k de una capa convolucional, por ejemplo $\frac{\partial y^c}{\partial A^k}$. Cada uno de estos gradientes recibe un procesamiento similar al que realiza la capa GAP sobre los mapas de activaciones en el método CAM, de esta forma se realiza la media aritmética de los gradientes de salida para la clase analizada c respecto a cada mapa de activaciones A de una capa concreta k, obteniendo como resultado los pesos de importancia correspondientes a cada neurona α_k^c.

$$\alpha_c^k = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial \mathrm{A}_{ij}^k}}_{\text{gradientes mediante backpropagation}}$$

Durante la computación de α_k^c mientras se hace el backpropagation de los gradientes respecto a las activaciones, la computación total equivale a productos matriciales sucesivos de las matrices de pesos y el gradiente con respecto a las funciones de activación hasta la última capa convolucional a la que los gradientes son propagados.

Entonces, este peso α_k^c captura la “importancia” del mapa de características k para una clase objetivo c.

Por último se hace una combinación lineal de los mapas de activación y se le aplica una función no lineal ReLU.

$$L^c_{Grad-CAM} = ReLU \underbrace{\left(\sum_k \alpha^c_k A^k \right)}_{\text{combinación lineal}}$$

Es preciso señalar que esto resulta en un mapa de calor con dimensiones iguales a la de los mapas de características convolucionales (14x14 en el caso de VGG por ejemplo).

Se aplica ReLU a la combinación lineal de los mapas porque solo importan las características que tienen una influencia positiva en la clase de interés, básicamente pixeles cuya intensidad debe ser incrementada para aumentar y^c. Los pixeles negativos tienen más probabilidades de pertenecer a otras categorías en la imagen. Sin el ReLU, los mapas de calor pueden resaltar más que la clase deseada.

3.3 Explicaciones Locales Interpretables Independientes del Modelo (LIME)

LIME, por sus siglas del inglés Local Interpretable Model-Agnostic Explanations fue propuesto en 2016 por Marco Tulio Ribeiro et al.[19] . Este algoritmo propone explicar las predicciones en cualquier clasificador o regresor, mediante aproximaciones locales con un modelo interpretable. La aproximación local implica centrarse en entender cómo funciona el modelo en una región específica del conjunto de datos en lugar de intentar entender todo el modelo en su conjunto. El objetivo de LIME es entonces explicar las predicciones de un clasificador en una manera interpretable, a partir de aprender o generar un modelo que sea interpretable en la localidad de la predicción.

Para esto, primero hay que explicar que significa “interpretable”. Se refiere a una explicación que sea entendible por humanos, sin importar las características reales usadas por el modelo. Por ejemplo, en clasificación de imágenes, una representación interpretable podría ser un vector binario que indica la presencia o ausencia de parches contiguos de píxeles similares. Estos parches se denominan super-píxeles y son regiones contiguas en una imagen que se agrupan en función de características como el color, la textura o la proximidad espacial. En lugar de considerar cada píxel de forma individual, los super-píxeles representan regiones más grandes que capturan información semántica y estructural coherente en la imagen. Por otro lado, una representación no interpretable podría ser un tensor con tres canales de color por cada pixel para representar una imagen, ya que esto es menos comprensible para los humanos.

Formalmente, se define una explicación como un modelo g ∈ G, donde G es una clase de modelos potencialmente interpretables, como árboles de decisión o modelos lineales. Estos son modelos que pueden ser presentados al usuario utilizando artefactos visuales o textuales para facilitar su comprensión.

El dominio de g es $\{0,1\}^{d'}$, es decir, funciona sobre la presencia/ausencia de componentes interpretables. Sin embargo, dado que no todo g ∈ G puede ser lo

suficientemente simple como para ser interpretable, usamos Ω(g) como medida de la complejidad de g. Esta medida puede ser, por ejemplo, la profundidad de un árbol de decisión o la cantidad de pesos distintos de cero en modelos lineales.

Si se denota el modelo con el que trabajamos como f: $\mathbb{R}^d$ $->$ $\mathbb{R}$ entonces en clasificación, f(x) es la probabilidad de que x pertenece a una clase. También usamos $\pi_x(z)$ como una medida de proximidad entre una instancia z y la instancia x, de manera de definir localidad alrededor de x. Por último $L(f,g,\pi_x)$ es una medida de que tan poco fiel es g como aproximación de f en la localidad definida por π_x.

Para asegurar que la aproximación local sea fiel y que al mismo tiempo sea interpretable, es necesario minimizar $L(f,g,\pi_x)$ mientras que se mantiene Ω(g) bajo para que sea interpretable por humanos. La explicación producida por LIME se obtiene como:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} L(f, g, \pi_x) + \Omega(g)$$

Esto es aplicable a diferentes familias de explicaciones G, funciones de fidelidad L y medidas de complejidad Ω, aunque nos centramos principalmente en modelos lineales.

Se busca minimizar la pérdida $L(f,g,\pi_x)$ la cual considera la localidad; sin hacer suposiciones sobre f, ya que se quiere que la explicación sea agnóstica respecto al modelo. Para aprender el comportamiento local de f a medida que las entradas interpretables varían, se aproxima L mediante muestras, utilizando π_x como peso. Se lleva a cabo un muestreo de instancias alrededor de x obteniendo elementos distintos de 0 de manera aleatoria y uniforme.

Dada una muestra perturbada z' ∈ $\{0,1\}^{d'}$ (que contiene una fracción de elementos de x distintos de 0) se recupera la muestra en la representación original z ∈ $\mathbb{R}^d$ y se obtiene f(z), que se usa como etiqueta para el modelo de explicación. Con este conjunto de datos Z de muestras perturbadas con sus etiquetas asociadas, se optimiza ξ(x) para obtener una explicación. La intuición primaria de LIME es muestrear instancias tanto en la vecindad de x (con un peso alto por π_x) como lejos de este (bajo peso por π_x).

Aunque el modelo original sea muy complicado como para explicarlo globalmente, LIME presenta una explicación local (en este caso lineal), manteniendo coherencia con la explicación global, donde la localidad es capturada por π_x.

En la figura 12 se observa una representación teórica del funcionamiento del método LIME. La función compleja f (desconocida para LIME) se visualiza mediante el fondo azul/rosa, y no puede ser aproximada correctamente por un modelo lineal. La cruz roja resaltada es la instancia que está siendo explicada. LIME muestrea instancias alrededor de esta, obtiene predicciones utilizando f, y las pesa por la proximidad a la instancia siendo explicada (representada acá mediante círculos y cruces de diferentes tamaños). A partir de esta información se construye la línea punteada, que es la explicación aprendida, precisa a nivel local pero no global.

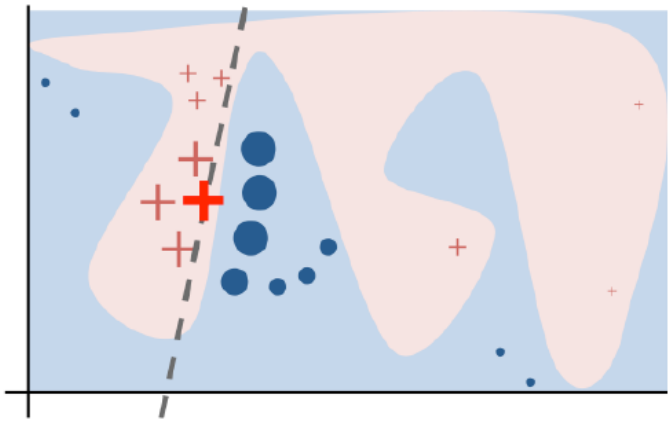


Figura 12: Representación teórica del funcionamiento del método LIME.[19]

En nuestro caso, que usamos imágenes, los componentes interpretables a utilizar son super-píxeles. Empleamos la técnica de oclusión, donde a partir de la imagen original generamos imágenes modificadas apagando (poniendo en color gris) super-píxeles específicos. Luego se aprende el nuevo modelo lineal que minimice el error existente entre sus predicciones y las del original para las imágenes perturbadas (que están en la vecindad del original, respecto al dominio de g), ponderando más los errores correspondientes a las imágenes menos alteradas (más cercanas a la original). Gracias a este enfoque, se pueden identificar los super-píxeles con mayores pesos asociados, los cuales tienen mayor influencia en la decisión final del nuevo modelo, y al combinarlos para generar una imagen, ocultando los demás, se puede obtener una representación de la parte más importante de la imagen para dicha clasificación.

En la figura 13 pueden observarse las explicaciones proporcionadas por LIME respecto a tres clasificaciones diferentes sobre una misma imagen. En orden de izquierda a derecha se muestran la imagen original, seguida de las explicaciones visuales sobre los super-pixeles de la imagen que más influyen en la decisión al clasificarla como “guitarra eléctrica”, “guitarra acústica” y “Labrador”

Figura 13: Explicación de las clasificaciones de una imagen mediante LIME. [19]

3.4 Explicaciones aditivas SHapley (SHAP)

El método SHAP “SHapley Additive exPlanations” es un enfoque de teoría de juegos para explicar la salida de cualquier modelo de aprendizaje automático. Su implementación en el ámbito del aprendizaje automático fue desarrollada en el 2017 en un paper de Scott Lundberg et al.[20]. Está basado en el concepto de valores de Shapley, los cuales fueron introducidos en 1953 por Lloyd Shapley para la teoría de juegos, utilizados por primera vez en 2010 por Strumbelj y Kononenko para explicar predicciones de aprendizaje automático.

Cuando se introduce el concepto de valores de Shapley, se hace con la idea de utilizarlos para distribuir de manera equitativa las ganancias de un juego cooperativo entre varios jugadores. En el contexto de Machine Learning, las características de entrada serían los jugadores, y la ganancia sería la diferencia entre la salida real del modelo y la salida esperada.

El método SHAP provee una forma de calcular los valores de Shapley para cada característica de entrada, lo que provee una medida de la contribución de cada característica al resultado del modelo. El valor Shapley para la característica “i”, denotado por ϕ_i, es definido como la contribución media de la característica “i” a través de todas las uniones posibles de características. Matemáticamente, el valor de Shapley puede expresarse de la siguiente forma:

$$\phi_i(f, S) = \sum_{T \subseteq S \setminus \{i\}} \frac{|T|!(|S| - |T| - 1)!}{|S|!} (f(T \cup \{i\}) - f(T))$$

donde X es el conjunto de todas las características de entrada, S es una unión de características que no incluye i, |S| es el tamaño de dicha unión, y f(S ∪ i) es la salida del modelo cuando las características en S e i están presentes. El término f(S) es la salida del modelo cuando sólo están presentes las características en S. Los valores de Shapley representan la contribución marginal promedio de la característica i sobre todas las posibles uniones.

A continuación se presenta un ejemplo del funcionamiento de SHAP, mostrando como aproxima los valores Shapley [21]:

Supongamos que hay una empresa que tiene 3 empleados, Anne, Bob y Charlie. La compañía terminó el mes con una ganancia de 100 y quiere distribuirla entre los empleados según su contribución. De esta forma tenemos la información de la ganancia de la compañía sin empleados (0) y con empleados (100)

Mediante los registros históricos, la compañía tiene determinada la ganancia cuando diferentes combinaciones de empleados trabajaron en los últimos meses:

	Empleados	Ganancia
1	Nadie	0
2	Anne	10
3	Bob	20
4	Charlie	30
5	Anne,Bob	60
6	Bob,Charlie	70
7	Anne,Charlie	90
8	Anne, Bob, Charlie	100

A primera vista parece que Bob contribuye 50 a la ganancia, ya que en la línea 2 se ve que Anne contribuye 10 y en la línea 5 Anne y Bob juntos es 60. Sin embargo, al ver la línea 4, Charlie contribuye 30, y en la 6 Bob y Charlie ganan 70, por lo que Bob contribuye 40, contradiciendo lo anterior.

En realidad, ambos resultados son correctos, ya que es un juego colaborativo y nos interesa la contribución de los empleados cuando trabajan juntos, y para entender las contribuciones individuales hay que analizar todos los caminos posibles desde “sin empleados” hasta “todos los empleados”

Camino	
1	Anne -> Anne,Bob -> Anne,Bob,Charlie
2	Anne -> Anne,Charlie -> Anne,Bob,Charlie
3	Bob -> Anne,Bob -> Anne,Bob,Charlie
4	Bob -> Bob,Charlie -> Anne,Bob,Charlie
5	Charlie -> Anne,Charlie -> Anne,Bob,Charlie
6	Charlie -> Bob,Charlie -> Anne,Bob,Charlie

Entonces se calcula la contribución de cada empleado en un camino. Por ejemplo, en el primer camino, Anne contribuye 10 (línea 1 de la tabla anterior), Bob contribuye 50 (línea 5 menos la contribución de Anne), y Charlie contribuye 40 (línea 8 menos línea 5). La contribución total tiene que dar la ganancia total: Anne = 10 + Bob = 50 + Charlie = 40 -> 100.

Si se repite el proceso, se puede calcular la contribución de cada empleado por cada camino, y al final se promedian las contribuciones, este sería el valor de Shapley para cada empleado

Camino	Combinaciones de empleados	Anne	Bob	Charlie
1	Anne -> Anne,Bob -> Anne,Bob,Charlie	10	50	40
2	Anne -> Anne,Charlie -> Anne,Bob,Charlie	10	10	80
3	Bob -> Anne,Bob -> Anne,Bob,Charlie	40	20	40
4	Bob -> Bob,Charlie -> Anne,Bob,Charlie	30	20	60
5	Charlie -> Anne,Charlie -> Anne,Bob,Charlie	30	40	30
6	Charlie -> Bob,Charlie -> Anne,Bob,Charlie	60	10	30
	Promedio (valor de Shapley)	**30**	**25**	**45**

Al pasar este proceso al ámbito de Machine Learning, los “individuos” son las características en el dataset, que pueden ser miles o millones. Por ejemplo, en clasificación de imágenes, cada píxel es una “característica”.

SHAP usa un método similar para explicar la contribución de características en la predicción de un modelo. Sin embargo, calcular la contribución de cada una no es posible en algunos casos (como en imágenes con millones de píxeles) debido a la cantidad de caminos posibles. SHAP hace simplificaciones para calcular las contribuciones de las características. Es esencial recordar que SHAP es una aproximación, no el valor de contribución real.

Al trabajar con imágenes, los valores obtenidos se representan gráficamente, indicando con color rojo los píxeles que aumentan la salida del modelo, y con azul los que la decrementan, normalmente mostrando debajo la imagen original en escala de grises y de manera casi transparente para poder ubicar visualmente donde se encuentran dichos pixeles. En la figura 14 puede observarse un ejemplo al trabajar con clasificación de imágenes, donde los píxeles en rojo influyen de manera positiva a la predicción actual, y los azules lo hacen de manera negativa.

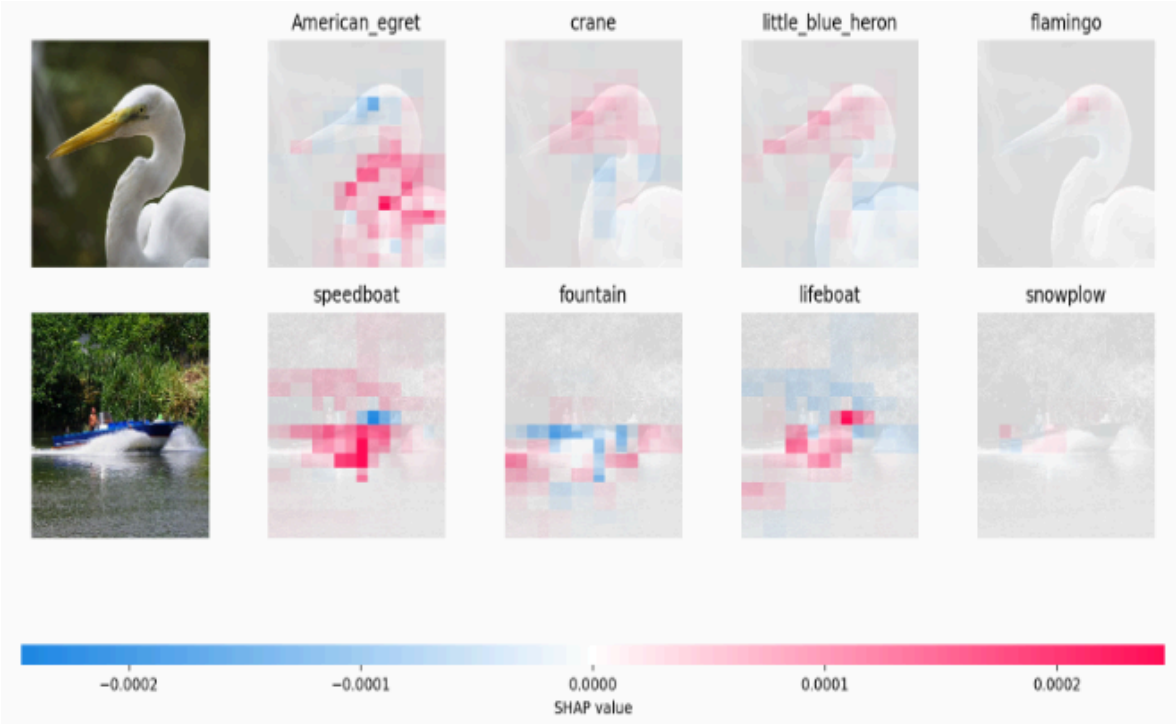


Figura 14: Ejemplo de uso de SHAP en imágenes.[36]

Puede verse que las técnicas de visualización son posibles debido a que las redes neuronales convolucionales basan sus características internas en el córtex visual humano. Esta característica hace posible obtener información con diferentes niveles de complejidad a medida que se avanza por la red en sí. En este trabajo se utilizarán estas técnicas de visualización para mostrar las secciones de una imagen que más afectan a la decisión de la red a modo de explicarla.

Respecto a los métodos vistos durante este capítulo, puede observarse que CAM y Grad-CAM generan mapas de calor. Sin embargo CAM requiere de una arquitectura específica en las redes (una capa de GAP) mientras que Grad-CAM utiliza la información del gradiente que se dirige a la última capa para obtener el mapa de calor (simula el GAP por fuera de la red). Por otro lado, LIME divide la imagen en super-píxeles y muestra únicamente los más influyentes, mientras que SHAP utiliza las características de la entrada, que en el caso de imágenes son los píxeles, para mostrar su contribución al resultado final.

3.5 Arquitecturas de CNNs consideradas

En esta sección se explica brevemente las características particulares de algunas arquitecturas de redes neuronales convolucionales que podrían ser analizadas con los métodos de visualización presentados previamente:

La mayoría de las redes presentadas en este apartado poseen implementaciones online de fácil y rápido acceso, pre-entrenadas con conjuntos de imágenes provenientes de bases de datos con cientos de miles de entradas, como por ejemplo el dataset de Imagenet[22], que consiste en más de 1.2 millones de imágenes etiquetadas manualmente

que cubren miles de categorías diferentes. Estas implementaciones pre-entrenadas permiten que los usuarios utilicen estas redes sin necesidad de entrenarlas desde cero.

LeNet-5 [23]

Es la arquitectura más antigua, creada por LeCun en 1998, recibe como entrada imágenes de 32x32 píxeles en escala de grises (1 canal), las cuales pasan por una capa de convolución con seis kernels de tamaño 5x5, resultando en un mapa de características de tamaño 28x28. Luego aplica average pooling con un filtro de 2x2 y un stride de dos. El proceso continúa de manera similar a medida que la información recorre las capas más profundas, reduciendo el tamaño de los mapas e incrementando el número de canales. Esta arquitectura posee alrededor de 60 mil parámetros.

Esta red fue creada originalmente para reconocer dígitos escritos a mano. Como puede verse en la figura 15, consiste de dos bloques de una capa convolucional y una de average pooling cada uno, y al final la información es procesada por dos capas completamente conectadas. En el caso de las versiones más nuevas de esta red, se emplea un clasificador softmax posterior a las capas completamente conectadas.

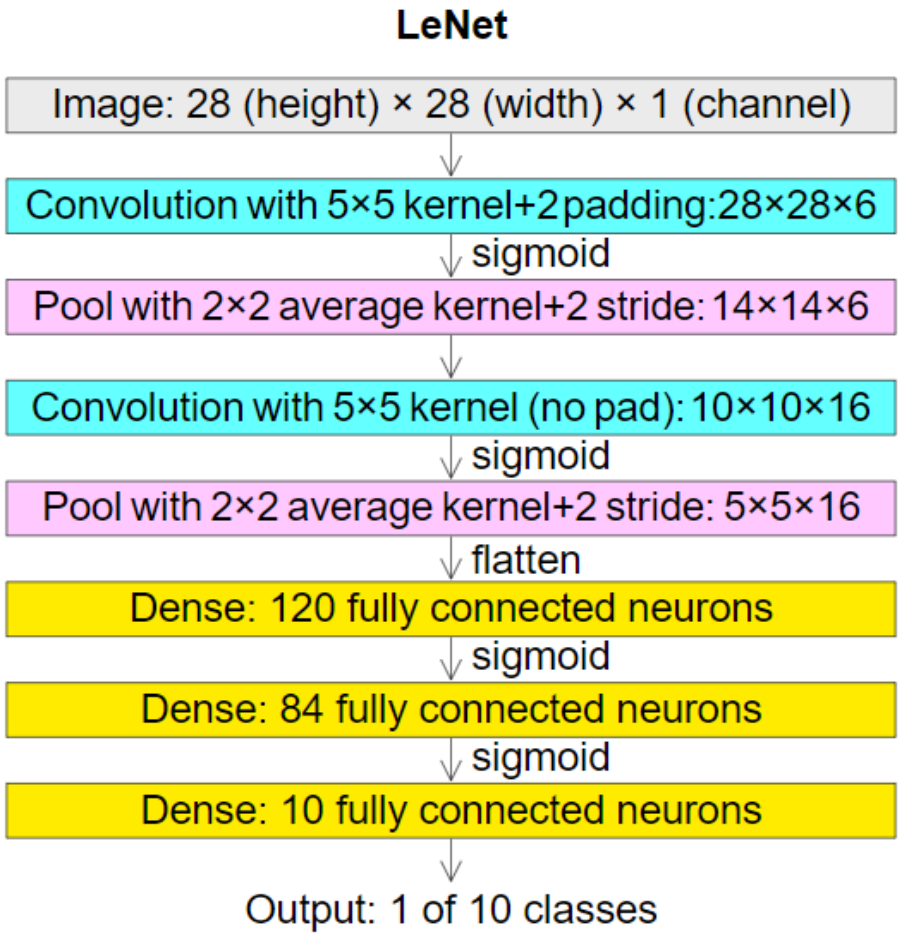


Figura 15: Diagrama de la red LeNet-5

VGG-16 [24]

Esta arquitectura introducida en el año 2014 por Karen Simonyan y Andrew Zisserman usa filtros de 3x3 y stride 1 en las capas convolucionales junto a capas de max pooling con filtros de tamaño 2x2 y stride 2, poseyendo en total 16 capas. Esta arquitectura posee un total de 138 millones de parámetros.

VGG-16 recibe como entrada imágenes de 224x224 y 3 canales (RGB), y es de las más utilizadas para extracción de características en imágenes, pudiendo clasificarlas entre 1000 tipos diferentes.

En la figura 16 puede verse un diagrama de la arquitectura.

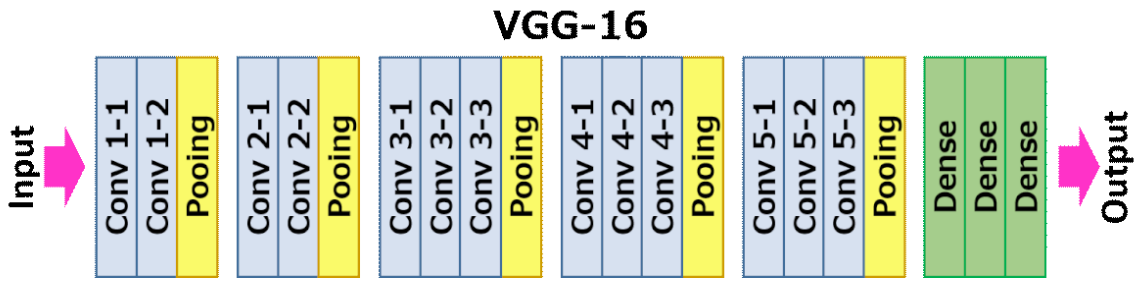


Figura 16: Diagrama de la red VGG-16 [25]

ResNets [26]

ResNet, por la abreviatura del inglés “Residual Network”, es una arquitectura introducida en el 2015 por la empresa Microsoft. La característica distintiva de ResNet es el uso de conexiones residuales, permitiendo una conexión directa entre la entrada de una capa n y una capa n + x, las cuales se activan mediante pesos de sesgo fuertemente positivo. Esto permite entrenar redes muy profundas (con decenas o cientos de capas) más fácilmente, y que estas se aproximen a una mayor precisión al profundizar

En la figura 17 puede verse el diagrama de una versión de estas redes denominado ResNet50, por las 50 capas de profundidad que posee. Esta arquitectura posee 23.5 millones de parámetros.

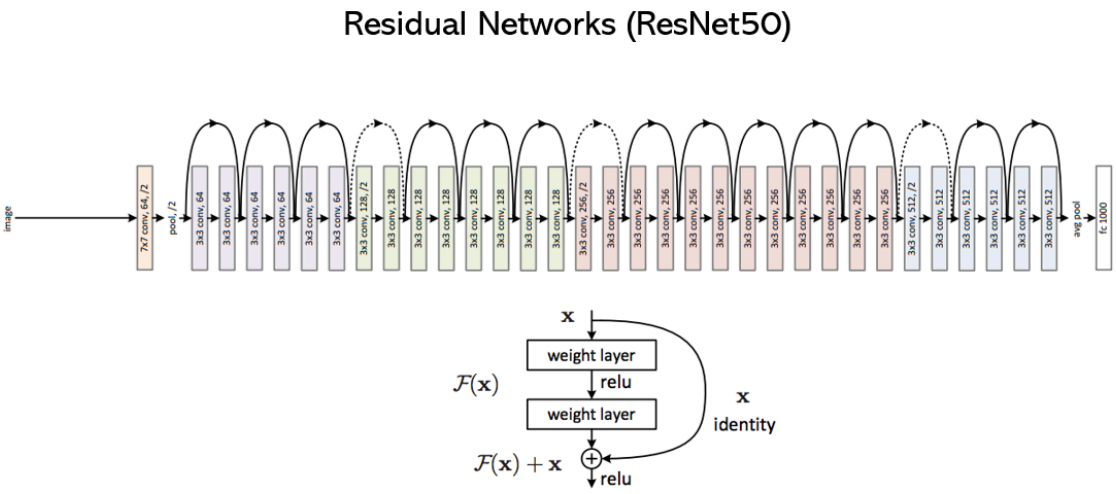


Figura 17: Diagrama de una red ResNet50

Inception [27]

Las redes Inception surgen como respuesta a dos problemas, el primero siendo que al trabajar con redes muy profundas estas pueden ser propensas a sobreajuste, y el segundo es el hecho de que diferentes imágenes pueden tener sus áreas de interés de diferentes tamaños y en diferentes regiones, lo que causa problemas al elegir el tamaño del kernel.

La solución que plantean estas redes es utilizar una arquitectura más ancha en vez de más profunda, aplicando múltiples filtros de convolución de diferentes características, además de max pooling, a una misma entrada, para luego concatenar estas salidas antes de enviarlas al siguiente módulo de la red. De esta manera el modelo extrae características generales y locales al mismo tiempo.

En la figura 18 puede observarse el diagrama de una implementación de la arquitectura inception llamada GoogLeNet, la cual posee alrededor de 6.7 millones de parametros.

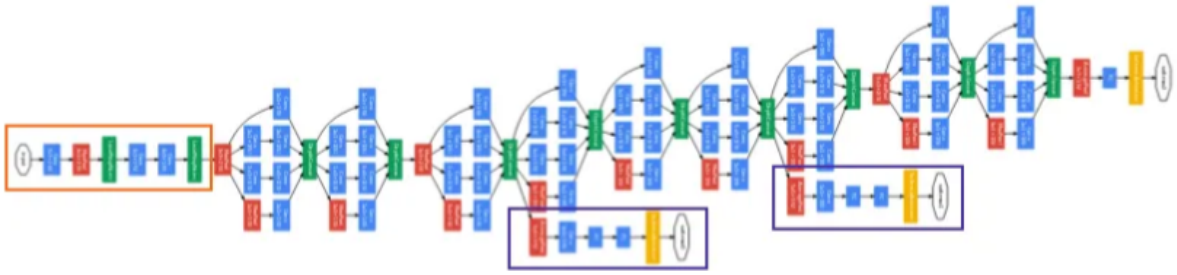


Figura 18: Diagrama de la red GoogLeNet [27 p.7]

Para este trabajo, se decide utilizar principalmente las arquitecturas ResNet50 y VGG16 debido a su simplicidad y la disponibilidad de información, además de ser arquitecturas muy utilizadas para *transfer learning* (Transferencia de aprendizaje), que consiste en la reutilización de un modelo preentrenado para resolver un nuevo problema, cómo se explica a continuación.

3.6 Transferencia de aprendizaje (*Transfer Learning*)

La transferencia de aprendizaje (*Transfer Learning*)[28][29][30] es una técnica ampliamente utilizada para la creación de CNNs, en lugar de comenzar desde cero. Esta técnica se basa en la idea de transferir el conocimiento aprendido durante la resolución de un problema sobre otro diferente pero relacionado. Para ello se utiliza un modelo pre-entrenado en una tarea similar, con una gran cantidad de imágenes, como punto de partida. Esto es especialmente útil cuando el conjunto de datos con el que se cuenta para entrenar es pequeño y no es suficiente para entrenar un modelo complejo desde cero. La transferencia de aprendizaje permite aprovechar los conocimientos adquiridos por el modelo pre-entrenado en tareas anteriores y adaptarlos al nuevo problema, lo que suele resultar en un mejor rendimiento y una convergencia más rápida durante el entrenamiento.

Se debe tener mucho cuidado al elegir el modelo pre-entrenado que se va a utilizar en cada caso. Si el problema es muy diferente, la predicción que se obtiene será muy inexacta. Existen muchas arquitecturas pre-entrenadas que están directamente disponibles para su uso en la biblioteca de Keras [31]. Algunas de ellas, se construyeron utilizando como datos de entrada el conjunto de imágenes conocido como ImageNet [22]. Este conjunto es lo suficientemente grande (1.2M de imágenes) para permitir crear un modelo generalizado. Estos modelos clasifican correctamente las imágenes en 1000 categorías de objetos que representan clases de objetos que se encuentran en la vida cotidiana, como especies de perros, gatos, diversos objetos domésticos, vehículos, etc.

Existen numerosos modelos disponibles para el transfer learning, pero dos de los más destacados son VGG16 y ResNet50. Ambas arquitecturas han sido pre-entrenadas en conjuntos de datos masivos como ImageNet, y se utilizan exitosamente como base para realizar la transferencia de aprendizaje a tareas específicas.

La transferencia de aprendizaje si bien es una técnica valiosa, es importante reconocer que puede ser aplicada incorrectamente y llevar a errores . Entre los errores que se pueden cometer se incluyen la selección inadecuada del modelo pre-entrenado base, problemas de sobreajuste debido a conjuntos de datos pequeños o no representativos, ajuste inadecuado de los hiper-parámetros del modelo, la presencia de datos de entrenamiento de baja calidad que contienen ruido o errores de etiquetado entre otros. Es fundamental disponer de alguna herramienta que permita identificar posibles errores en la detección de patrones erróneos a tiempo, con el fin de prevenir mayores daños.

Capítulo 4: Desarrollo

En este capítulo se realiza un análisis de las implementaciones de los métodos de visualización presentados previamente así como de los resultados obtenidos al aplicarlos en diferentes redes neuronales. Luego se seleccionan las redes y métodos que se van a incorporar al programa a desarrollar.

Como se mencionó anteriormente, el objetivo de este trabajo es desarrollar una herramienta que integre múltiples métodos de visualización de redes neuronales convolucionales, lo cual requiere elegir tanto las técnicas de visualización como las arquitecturas de CNNs. El proceso de selección inicia con el análisis de los métodos de visualización, ya que como pudo verse previamente (puntualmente en el caso del método CAM), estos pueden poner restricciones en las arquitecturas de Redes neuronales Convolucionales cuyas salidas pueden analizar. En esta etapa surgen los primeros inconvenientes, ya que no siempre es sencillo encontrar implementaciones online de libre acceso, escritas en el lenguaje Python para los diferentes métodos y que se encuentren en un estado completamente funcional.

Durante este trabajo se decide utilizar las arquitecturas ResNet50 y VGG16 debido a su simplicidad y la disponibilidad de información pertinente. Es importante destacar que la arquitectura ResNet50 posee las características necesarias para poder utilizar el método CAM sobre sus predicciones, lo que permite comparar dicho método con los demás. Sin embargo, estos métodos de visualización pueden ser aplicados sobre otras arquitecturas si así se desea. La única restricción existente es que CAM requiere que la arquitectura posea una capa de GAP.

4.1 Implementaciones de los métodos de visualización

En este apartado se detalla la investigación y desarrollo del código para cada método de visualización, indicando los desafíos enfrentados durante este proceso. Además, se muestra una comparativa visual de los resultados presentados por cada método. Un punto a resaltar es la dificultad de conseguir implementaciones que funcionaran adecuadamente o que siguieran lo visto de forma teórica en el capítulo 3. Algunas de las implementaciones encontradas eran de tipo "caja negra", por lo que tuvieron que ser descartadas y se dio prioridad a las implementaciones *Open Source*.

En este trabajo se realizan comparativas entre los métodos de manera experimental para comprobar la coherencia existente entre ellos. Es de esperar que si se utiliza una misma red neuronal convolucional para clasificar una misma imagen, los distintos métodos de visualización indiquen resultados similares respecto a las regiones de la imagen que afectan a dicha clasificación. Es importante señalar que si bien los resultados pueden no ser satisfactorios a los ojos de una persona, el error no estaría en el método sino en la CNN.

El desafío inicial consiste entonces, en encontrar implementaciones que sirvan como punto de partida para luego realizar las adaptaciones necesarias para que funcionen con los recursos disponibles. En esta etapa se encontraron ciertos inconvenientes. Por un lado, las implementaciones oficiales, si bien son funcionales y se encuentran bien documentadas, no

se adecuan del todo a los objetivos de la herramienta a desarrollar, por ejemplo por las diferentes librerías utilizadas, o por poseer funcionalidades inseparables que resultan inadecuadas para el objetivo de este trabajo. Por otro lado las implementaciones no oficiales poseen problemas básicos, como la falta de documentación, errores simples normalmente relacionados con un fallo de tipeo, o falta de contenido en los casos en los que se presentan únicamente retazos del código original.

Respecto a las imágenes utilizadas durante las pruebas, la mayoría son imágenes aleatorias obtenidas de internet (Google) o provienen del conocido dataset ImageNet [22]. Estas últimas son útiles para asegurar que las implementaciones encontradas y modificadas proveen los resultados esperados de los métodos. Además, la mayoría de los modelos fueron entrenados con ese conjunto de datos, lo que permite descargar los modelos pre-entrenados sin necesidad de realizar los entrenamientos desde cero.

4.1.1 Método CAM

El primer desafío respecto a la investigación de implementaciones del método CAM, se relaciona con el método Grad-CAM, el cual entorpece las búsquedas en línea al influir en los resultados proporcionados por los buscadores, complicando así el acceso a información. Sin embargo es posible encontrar una implementación online[32] en la cual se utiliza CAM para analizar la salida de una CNN con la arquitectura ResNet50.

En el trabajo [17] se presenta el método CAM junto con un enlace al github donde se encuentra la implementación desarrollada por los creadores. Sin embargo utiliza PyTorch y modelos en Caffe mientras que en este trabajo se utiliza TensorFlow y Keras ya que estos frameworks automatizan algunos procedimientos que requieren las imágenes antes de ser clasificadas por las redes. Estos procesos serán posteriormente explicados con más detalle. Además se opta por estos frameworks debido a la familiaridad previa con los mismos.

Luego de implementar un código funcional, se llevan a cabo algunas pruebas y se descubre la necesidad de preprocesar las imágenes antes de ingresarlas a la CNN para su clasificación. Esto se debe a que la librería cv2, que es la librería estándar de facto para la manipulación de imágenes, importa las imágenes en formato BGR, en lugar de utilizar RGB; y por otro lado la necesidad de cambiar el tamaño de las imágenes, ya que las diferentes redes neuronales convolucionales tiene diferentes limitaciones sobre los datos de entrada, por ejemplo en el caso de VGG16 y ResNet50, se necesitan entradas de 224 x 224 píxeles.

Sin embargo, al preprocesamiento de las imágenes mencionado previamente, se le debe agregar otro preprocesamiento específico de las librerías que implementan las CNN en sí. Esto se debe a que utilizan diferentes tipos de datos o formatos para manipular la información de las imágenes, por lo que ofrecen métodos y funciones especiales para modificar las entradas de manera adecuada.

Finalmente, una vez realizadas todas las modificaciones necesarias en la implementación, se obtiene un código capaz de aplicar el método CAM para analizar la decisión de una CNN ResNet50. En la figura 19 se muestra a la derecha el resultado de aplicar el código desarrollado a la imagen ubicada a la izquierda. Este resultado consiste en

un mapa de calor que indica con colores cálidos las regiones de la imagen que más influyeron en la decisión de clasificar la imagen como “Labrador Retriever”, mientras que los colores más fríos indican las regiones que menos influyen.

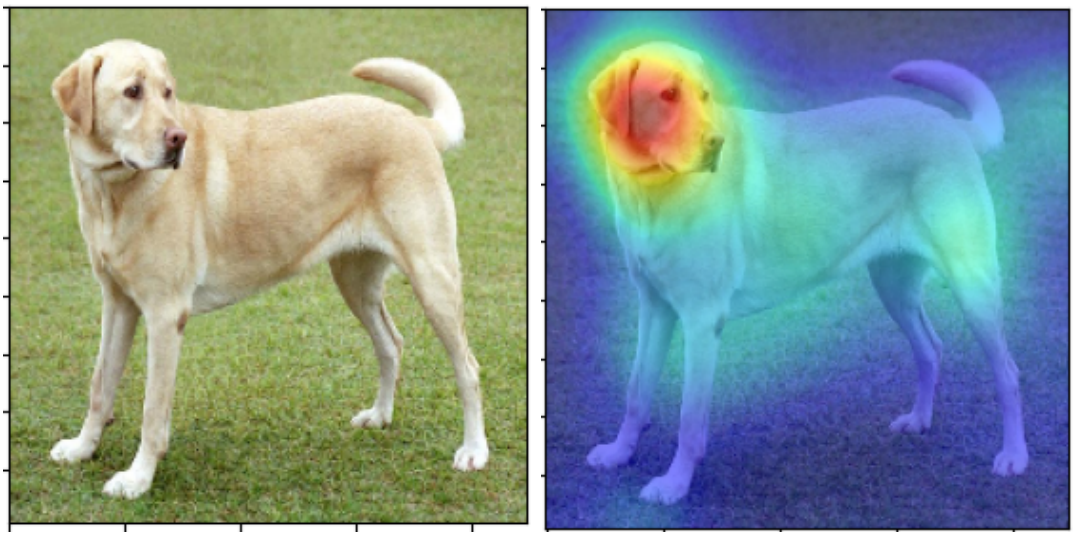


Figura 19: Resultado de la aplicación del código desarrollado para el método CAM sobre la imagen de un labrador retriever.

A pesar del éxito en el desarrollo de esta implementación, la cual puede encontrarse en Google colab[33], se ha decidido no incorporar el método CAM en el programa final debido a sus restricciones respecto a la arquitectura de las CNN que puede analizar. Esto se debe a que podría limitar la escalabilidad de la herramienta a desarrollar y generar problemas a futuro para expandir sus capacidades. Sin embargo, la información obtenida respecto a posibles problemas a solucionar en el código, y el preprocesamiento necesario para poder manipular imágenes y adecuarlas para el uso en las CNNs se utiliza en el resto de este trabajo para la correcta implementación de los demás métodos de visualización. Resaltamos que los resultados obtenidos en la aplicación del método CAM en las imágenes utilizadas se emplearán para evaluar que el resto de los métodos analizados e implementados proporcionan resultados coherentes.

4.1.2 Método Grad-CAM

Durante la investigación de este método se analiza la implementación oficial, presentada en el trabajo donde se propone el método Grad-CAM[18], la cual presenta el mismo inconveniente que la implementación oficial de CAM, utilizando diferentes librerías a las empleadas en este trabajo. Además posee una funcionalidad (y complejidad) superior a la necesaria para el programa a desarrollar que no parece ser separable del resto del código y se encuentra parcialmente escrita en el lenguaje de programación Lua. Debido a esto se decide no utilizarla.

Por otro lado, resulta sencillo encontrar una implementación en la página oficial de Keras [34] con el inconveniente que utiliza la arquitectura de red neuronal convolucional “Xception”, por lo que es necesario convertir el código para que utilice ResNet50 y VGG16. Lo que implica realizar algunas modificaciones de las cuales la más importante es agregar

todo el preprocesamiento necesario a las imágenes de entrada, como fue analizado previamente.

Respecto a esta implementación se realizaron algunas modificaciones que no afectan al funcionamiento del método en sí, pero son útiles, o al menos interesantes, durante la etapa de investigación. Una de ellas es la posibilidad de ver y analizar los mapas de calor individuales que más afectan a la generación del mapa de calor final (que se genera mediante una combinación lineal de estos individuales) como puede verse en la figura 20. Además, se agrega una sección de código que modifica la visualización de la salida obtenida para poder comparar más fácilmente los resultados con los del método CAM, generando como salida una imagen del mismo tamaño e intensidad del mapa de calor. Esto permite realizar una comparativa visual más directa, como puede observarse en la figura 21, donde a la izquierda de la imagen se ve el resultado de aplicar Grad-CAM y a la derecha el resultado de aplicar CAM.

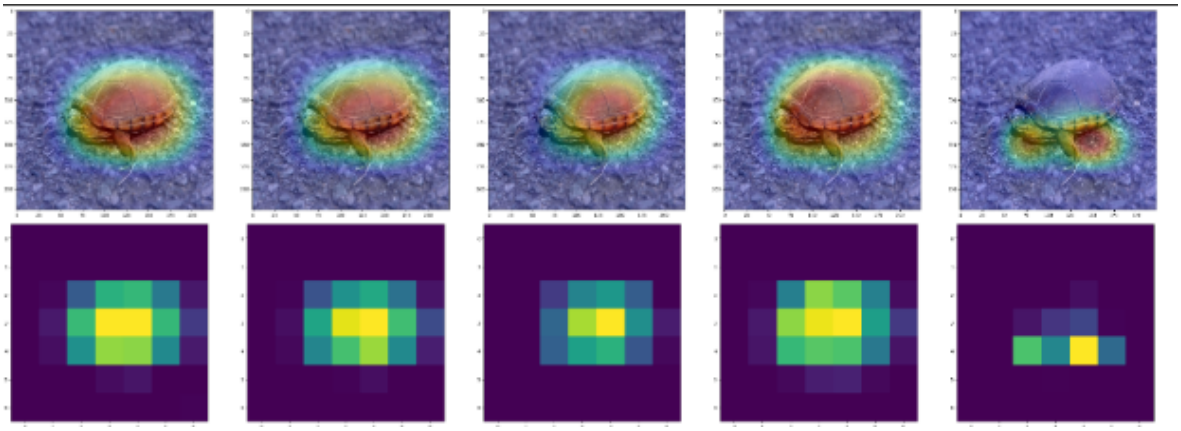


Figura 20: Pueden verse los mapas de calor correspondientes a los cinco mapas de activación con más peso en la decisión final

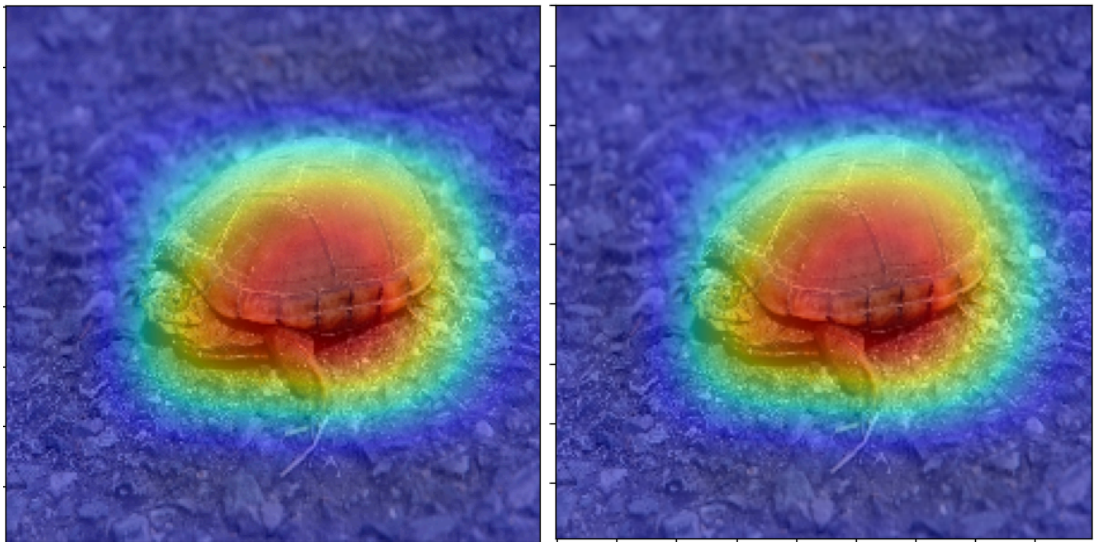


Figura 21: Resultados de aplicar Grad-CAM (izquierda) y CAM (Derecha) a la clasificación mediante ResNet50 de la misma imagen.

En la figura 22 puede verse el mapa de calor resultante de aplicar Grad-CAM sobre la salida de una red ResNet50 sin las modificaciones para la comparación con CAM. Esta red identifica correctamente a la imagen como una representación de la categoría "*mud turtle*"

Figura 22: Aplicación de Grad-CAM sobre ResNet50, identificando "mud turtle"

Durante esta etapa se hicieron pruebas tanto con la arquitectura ResNet50 como con VGG16 y un dato a considerar para la correcta representación de los resultados, es que los mapas de activación en base a los cuales se generan los mapas de calor poseen diferentes dimensiones según la arquitectura, por lo que se debe modificar la escala con la cual se presentan los resultados para que se correspondan efectivamente con la posición y dimensiones de la imagen de entrada.

Por otro lado, debido al funcionamiento del método Grad-CAM, es necesario tener conocimiento sobre la organización interna de la red a analizar, ya que se debe indicar la última capa convolucional, de la cual se obtiene información del gradiente que se dirige hacia esta. Dicha información es necesaria para replicar el funcionamiento de una capa GAP de manera externa a la red en sí, lo cual es un proceso fundamental en el funcionamiento del método. Es necesario entonces llevar a cabo un análisis de las capas internas para identificar la última capa convolucional, aunque cabe destacar que esto se puede lograr de manera programática.

La implementación para este método de visualización puede encontrarse en Google colab[35]

4.1.3 Método LIME

La implementación oficial de este método[19], si bien es funcional, tiene una complejidad muy superior a la necesaria para este trabajo, lo que a su vez dificulta su estudio y análisis. Para comprender mejor el método, se decide buscar información y desarrollar una implementación de más bajo nivel, con el objetivo de entender cada paso del proceso.

Debido a lo mencionado previamente se decide buscar una implementación de más bajo nivel como la desarrollada por el Dr. Cristian Arteaga[37], de la Universidad de Nevada. En ese trabajo se presenta información detallada respecto al método y su funcionamiento, descomponiendo los diferentes segmentos del procesamiento y es la utilizada como base para este trabajo. La implementación encontrada utiliza la arquitectura de CNN InceptionV3, pero es necesario flexibilizar esta limitación desacoplando el método y la arquitectura de la red, ya que el objetivo de esta herramienta es que pueda ser empleada con diferentes arquitecturas de CNN. Además esta implementación ofrece la posibilidad de alterar las perturbaciones, es decir la cantidad, tamaño y forma de los super-pixeles en los que será dividida la imagen que se está analizando.

Como conclusión se desarrolla una implementación de LIME independiente de la arquitectura de la red, es decir que se puede utilizar con diferentes arquitecturas de redes sin necesidad de realizar modificaciones internas, con la posibilidad de configurar los super-pixeles en los que se divide la imagen para su procesamiento, como puede observarse en la figura 23.

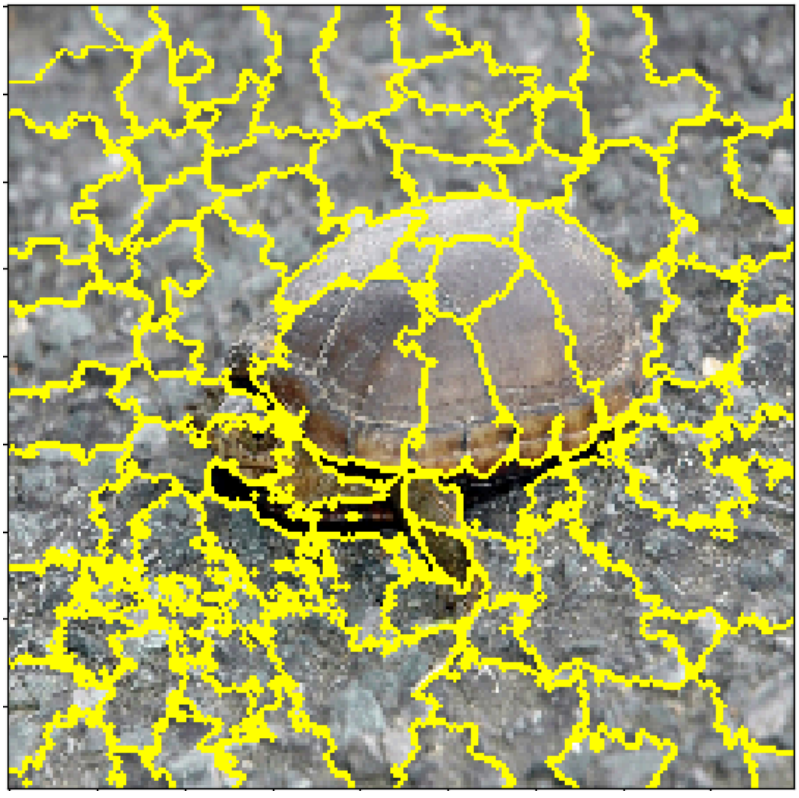


Figura 23: Perturbaciones o super-pixeles en los que será dividida la imagen para realizar posteriormente el procesamiento propio de LIME.

La información provista por LIME permite una comparación visual con los resultados de los métodos analizados previamente. Es evidente que en los casos planteados hay similitudes entre las conclusiones obtenidas mediante ambos métodos. En la figura 24 se puede observar en la imagen a la izquierda el resultado de aplicar el método LIME y a la derecha el resultado de aplicar grad-CAM sobre la misma imagen, pudiendo notarse que en ambos casos la decisión de la CNN fue influida en gran parte por la misma región de la imagen.

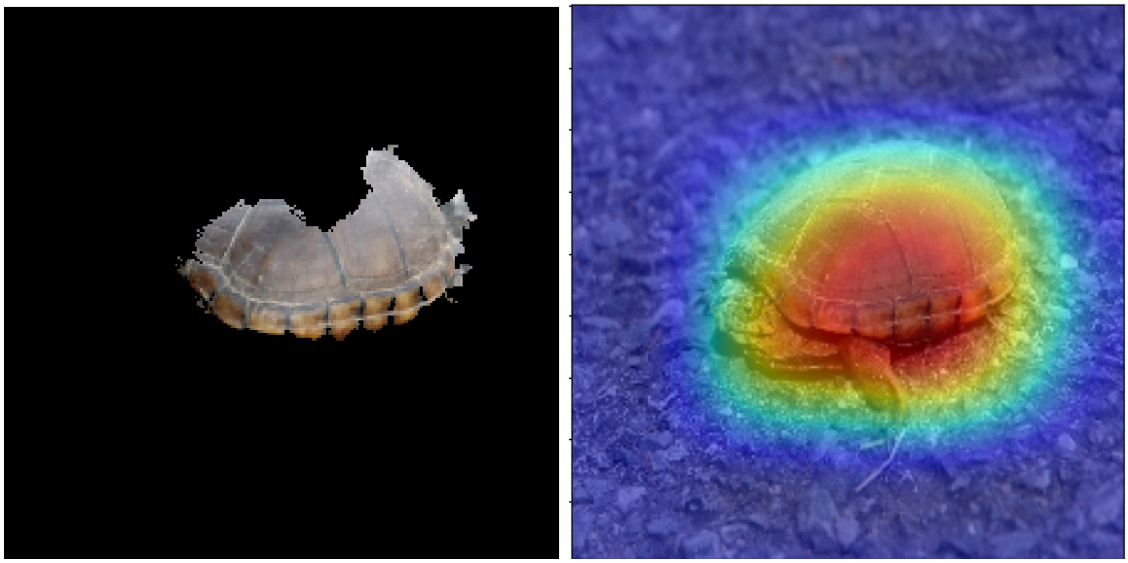


Figura 24: Se muestran a la izquierda los super-pixeles resultantes de LIME y a la derecha el mapa de calor resultado de Grad-CAM.

Un detalle a considerar de LIME es que los super-pixeles que se muestran como los más influyentes para una decisión pueden variar para una misma imagen y una misma red neuronal convolucional debido a dos motivos. Por un lado los parámetros utilizados para realizar las perturbaciones, ya que estos pueden cambiar la forma y cantidad de super-pixeles en los que la imagen será dividida, y por otro lado la cantidad de super-pixeles a mostrar en el resultado final, ya que si se divide una imagen en cien super-pixeles y se decide mostrar los noventa que más afectan a la decisión, se estaría mostrando prácticamente toda la imagen, incluyendo regiones que no afectan realmente a la decisión.

Figura 25: Resultado de aplicar LIME a la misma imagen cambiando la cantidad de super-pixeles a mostrar, siendo 4 en la versión de la izquierda y 9 en la imagen de la derecha.

Esta implementación de LIME se encuentra en Google colab[38] para realizar pruebas, y si bien es funcional con imágenes a color, no es capaz de procesar imágenes en escala de grises (ni imágenes .png que posean un canal alfa) en su versión actual debido a las características del proceso para generar las perturbaciones.

4.1.4 Método SHAP

Esta implementación pudo obtenerse directamente de la documentación oficial de SHAP[36], una librería que permite utilizar el método para visualizar información acerca de las decisiones tomadas por CNNs. Este código utiliza imágenes previamente procesadas para la red a analizar. Este preprocesamiento no es transparente ni fácil de replicar, ya que no es posible encontrar información al respecto. Por lo tanto, es necesario realizar modificaciones que permitan aplicar a las imágenes el preprocesamiento adecuado según la red a utilizar.

En la figura 26 puede observarse el resultado de aplicar SHAP a la clasificación realizada mediante ResNet50 a la misma imagen analizada previamente, donde la información obtenida es presentada en un formato que presenta a la izquierda la imagen original, y a la derecha el análisis realizado por el método SHAP. En dicha figura puede verse a simple vista que la presentación de la imagen original se encuentra extremadamente distorsionada en sus colores, mientras que a la derecha, superpuestos sobre la imagen, se presentan los píxeles que más afectan a la decisión. Esto se condice con lo analizado previamente mediante los otros métodos, mostrando que esa parte del procesamiento funciona correctamente. Es importante tener en cuenta que a mayor cantidad de evaluaciones en la ejecución del método, se obtiene mayor detalle y precisión, es decir, la información obtenida será dividida en una mayor cantidad de píxeles.

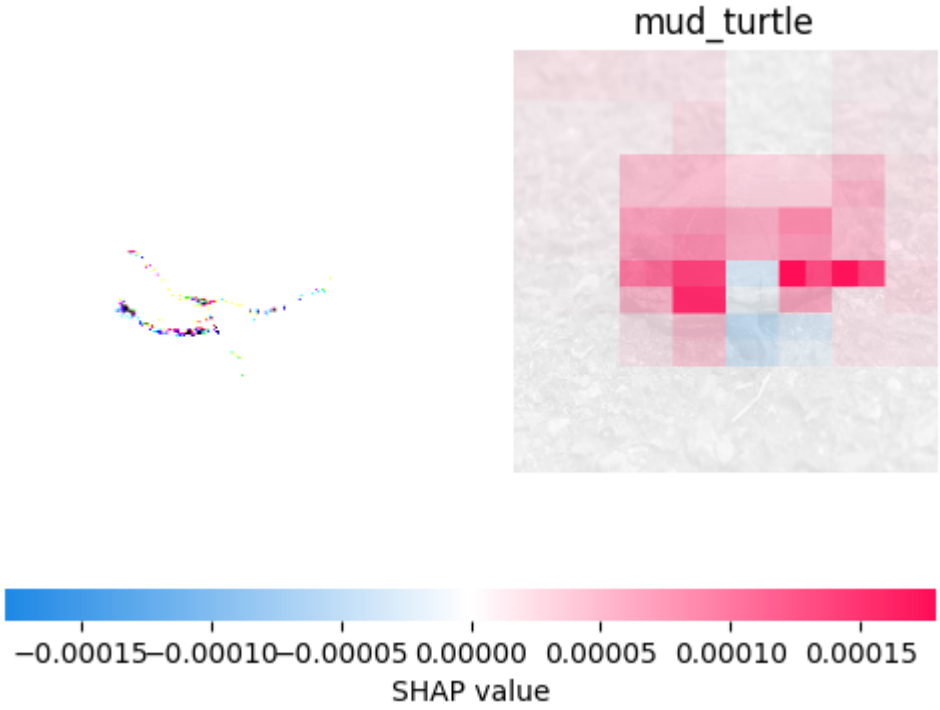


Figura 26: Resultado de aplicar SHAP a la misma imágen analizada con técnicas previas.

Otro detalle a tener en cuenta es que en la implementación se definen “enmascaradores” (Maskers) utilizados para entrenar a los “explicadores” (Explainers), los cuales, como indica su nombre, se encargan de realizar las explicaciones respecto al modelo a analizar y sus decisiones. SHAP provee tres alternativas para Maskers “blur(kernel_xsize, kernel_xsize)”, “inpaint_telea”, o “inpaint_ns”. Tras realizar pruebas no se notan diferencias para los casos analizados en este trabajo, por lo tanto se decide usar “blur” de manera arbitraria.

Por último, tras realizar todas las modificaciones necesarias, se consigue un código funcional con resultados similares a lo mostrado en la documentación de la librería. En la figura 27 se presenta el resultado de analizar la clasificación realizada por ResNet50 sobre la imagen utilizada previamente, en este caso utilizando una mayor cantidad de píxeles que los utilizados en la figura 26. Puede notarse que no todo el caparazón de la tortuga afecta positivamente a la decisión, ya que existe una región interna, marcada en azul, que lo hace de forma negativa.

La implementación para este método de visualización puede encontrarse en Google colab[39].

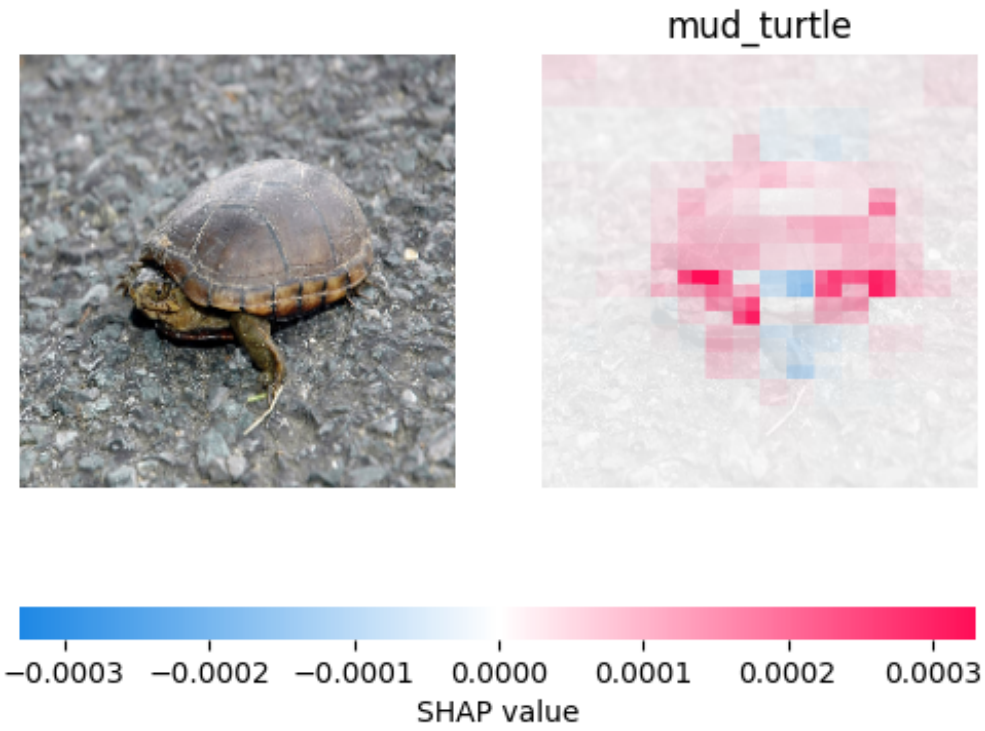


Figura 27: Resultado de aplicar SHAP a la misma imágen analizada previamente, en este caso tras solucionar el problema de distorsión sobre la imagen original.

4.2 Comparativa entre los métodos vistos para diferentes arquitecturas

A continuación se presentan los resultados obtenidos de los diferentes métodos de visualización al explicar las decisiones de las redes VGG16 y ResNet50 sobre la clasificación realizada de algunas imágenes.

Respecto a las configuraciones de los métodos LIME y SHAP, en ambos se realizan mil iteraciones. En el caso de LIME, la imagen se divide en una cantidad cercana a ochenta super-pixeles, ya que no es un valor que se pueda ingresar manualmente, sino que depende de otros parámetros al momento de realizar las subdivisiones de la imagen. Posteriormente se presentan la mayor cantidad entre ocho y el diez por ciento de los super-pixeles totales más influyentes luego de analizar mil perturbaciones, es decir, mil combinaciones de super-pixeles visibles y ocultos de manera aleatoria.

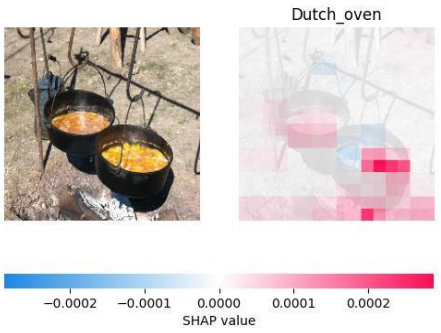
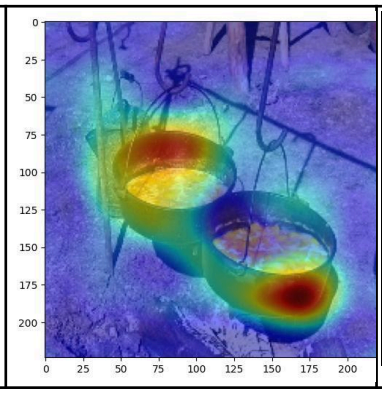
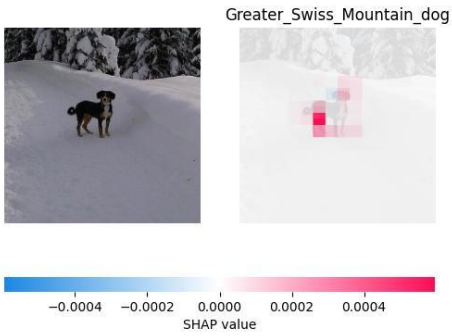
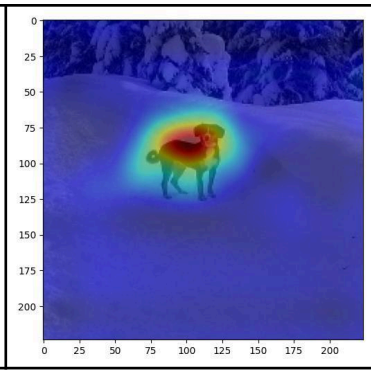
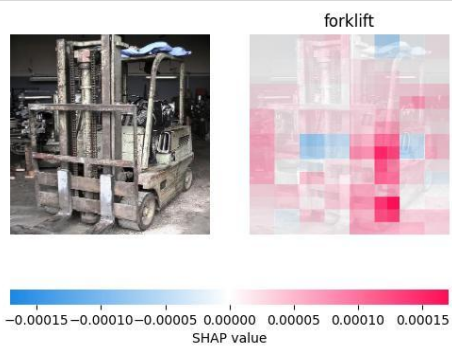
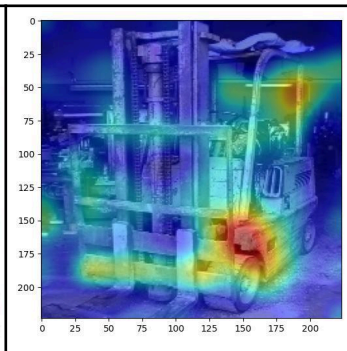
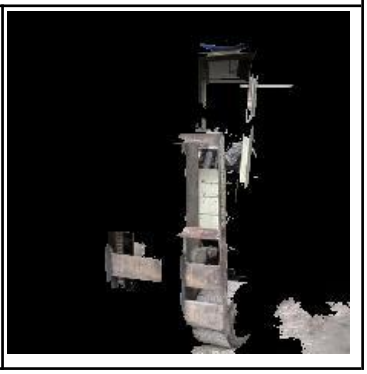
Debido a las limitaciones del método CAM, sus resultados se muestran únicamente en el caso de utilizar ResNet50, aunque se espera que los resultados sean similares sino idénticos a los presentados por Grad-CAM.

4.2.1 Visualización sobre los resultados de VGG16

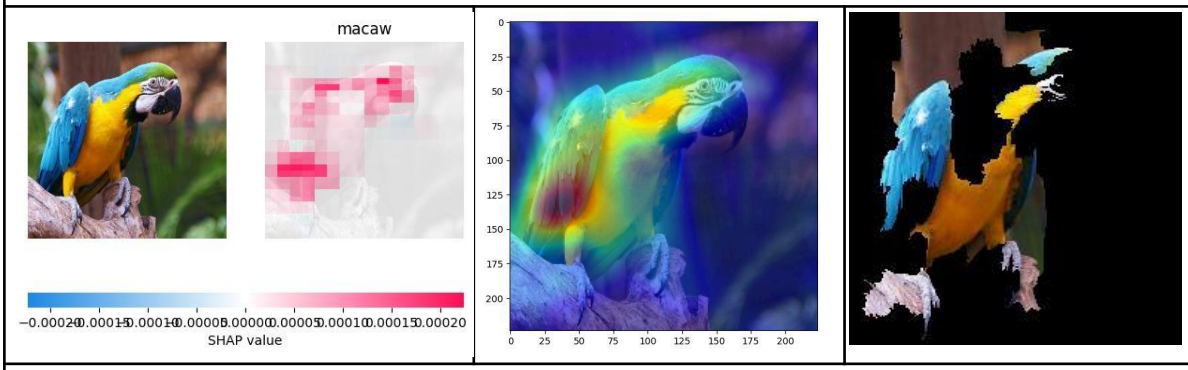
En la tabla 1 se muestran visualmente las explicaciones de los resultados luego de clasificar once imágenes. Las primeras seis son obtenidas del repositorio ImageNet [22] y las siguientes cuatro de lugares aleatorios de Internet. La última muestra el resultado de

aplicar el “sticker” mencionado anteriormente, que al estar presente, engaña a la red haciéndola clasificar la imagen como una tostadora [2]. Se muestra de manera intercalada la clase detectada por la red neuronal convolucional y debajo las aplicaciones de los métodos de visualización. En la celda de la izquierda se observa la imagen original junto al resultado de aplicar SHAP, en la del centro el resultado de Grad-CAM y a la derecha el resultado de aplicar LIME. En las primeras seis imágenes se indica también la etiqueta correspondiente según ImageNet.

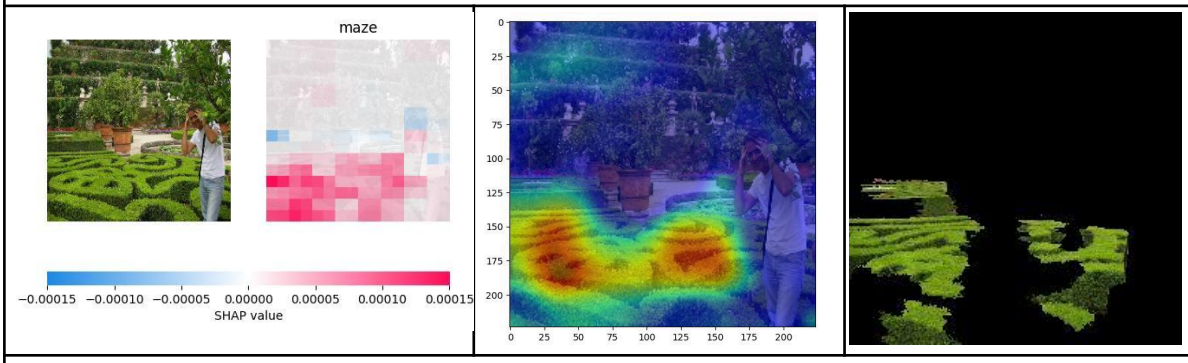
Tabla 1: Imágenes clasificadas por VGG16 junto con las visualizaciones de las regiones destacadas por SHAP, Grad-CAM y LIME

Clase: Dutch Oven **Clase detectada**: Dutch Oven			
			
Clase: Entlebucher **Clase detectada:** 1-Greater Swiss Mountain Dog 2-Appenzeller 3-EntleBucher			
			
Clase: Forklift **Clase detectada**: Forklift			
			

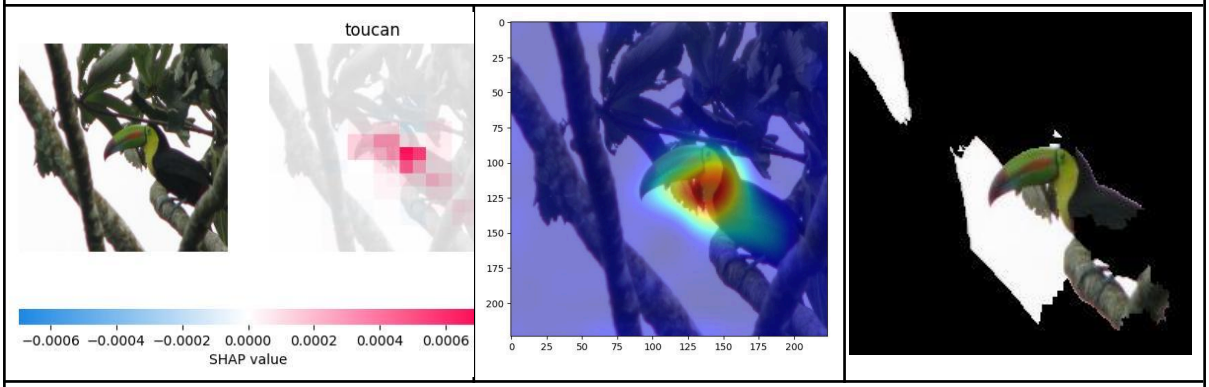
Clase: Macaw
Clase detectada: Macaw



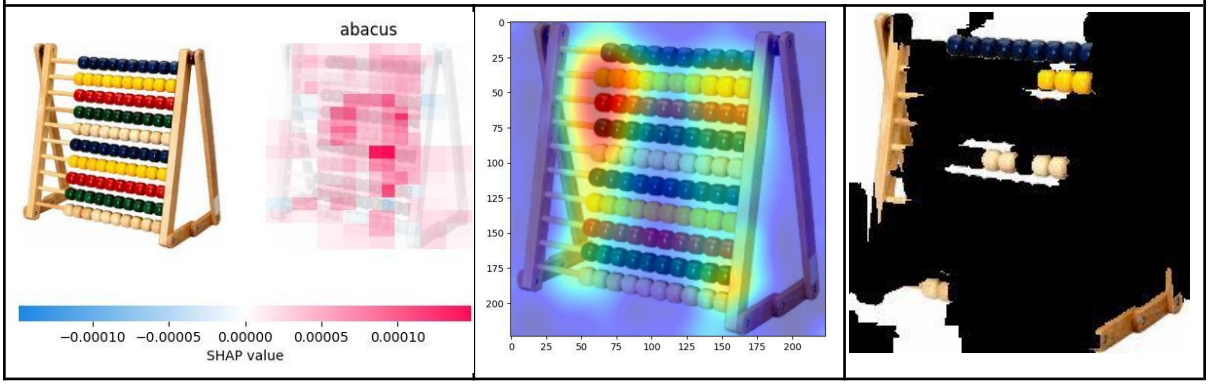
Clase: Maze
Clase detectada: Maze



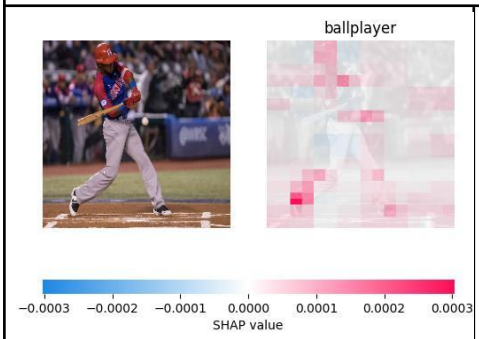
Clase: Toucan
Clase detectada: Toucan

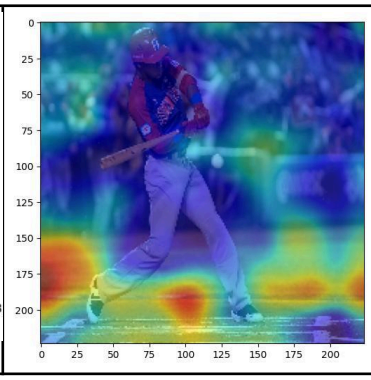


Clase detectada: Abacus

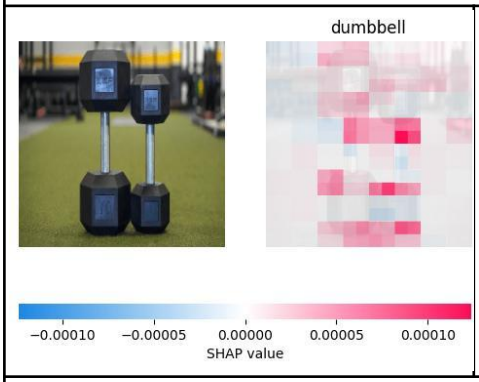


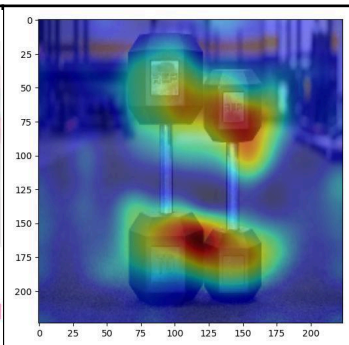
Clase detectada: Ballplayer

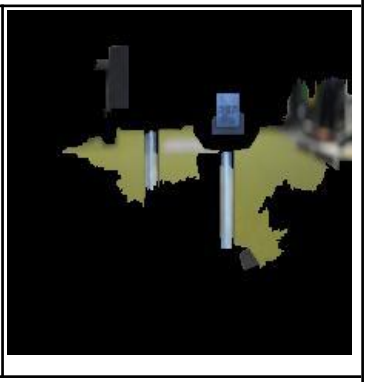




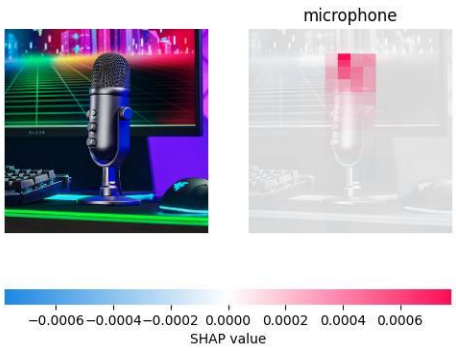
Clase detectada: Dumbbell

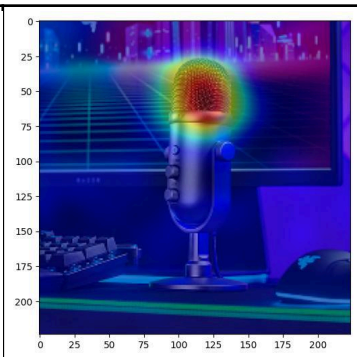




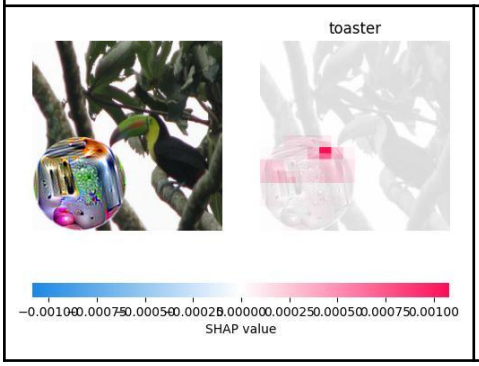


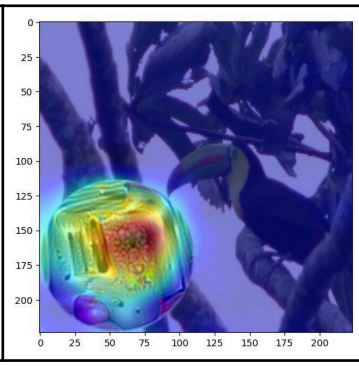
Clase detectada: Microphone

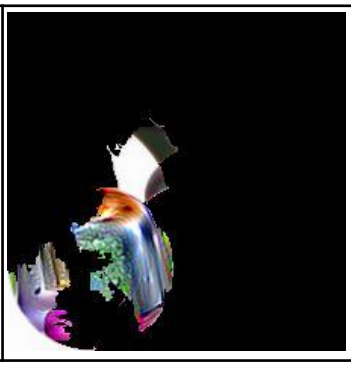




Clase detectada: Toaster







Puede verse que, en términos generales, los diferentes métodos proveen explicaciones similares en los casos en que información importante para la clasificación se

encuentra en un segmento reducido de la imagen, como pueden ser los casos de “Maze” o “Toucan”. Por el contrario, en los casos como “Abacus” en los que la mayoría de la imagen posee elementos visualmente distintivos de la clase, los diferentes métodos proveen explicaciones diferentes entre ellos.

Otro detalle a tener en cuenta es que la explicación proporcionada por LIME depende de la división de la imagen en super-pixeles. Por ejemplo, si hay dos regiones que influyen mucho en la clasificación, pero una se encuentra dividida en múltiples super-píxeles, es posible que cada uno de de estos no influya lo suficiente de manera individual como para ser presentado en la explicación. En este caso solamente se mostraría la segunda región, que no se encuentra dividida, y tiene comparativamente más peso. Además del hecho de que a mayor cantidad de super-pixeles, se debería analizar una mayor cantidad de permutaciones para obtener un resultado más preciso, lo que implica un mayor tiempo de procesamiento.

En relación a esto último, y como se mencionó previamente de manera breve, también hay que tener en cuenta la cantidad de super-pixeles a presentar en la explicación final, ya que si se muestran demasiados, se llega a un punto en el cual pueden verse elementos de la imagen que no tuvieron una influencia real en la decisión de la red neuronal. Por ejemplo, si hay cuatro super-pixeles que influyen en un noventa por ciento de la decisión final y se decide mostrar cinco, la explicación incluye una región de la imagen que realmente solo influye en una fracción del diez por ciento restante.

Resulta interesante analizar el caso de la imagen “Ballplayer” en la que puede observarse que Grad-CAM informa que la decisión de la red se basa principalmente en el suelo en el cual se encuentra el jugador. SHAP y LIME también parecieran hacer énfasis principalmente en una combinación del suelo y las piernas del jugador junto a una muy ligera importancia en el casco del mismo. Estos resultados sugieren que la red podría estar clasificando la imagen por razones equivocadas, dando importancia al entorno en lugar de a la persona que se encuentra jugando. Este ejemplo destaca la importancia de utilizar múltiples métodos de visualización para contrastar resultados, ya que plantea la posibilidad de que la CNN esté clasificando una imagen según el entorno en vez del sujeto primario de análisis.

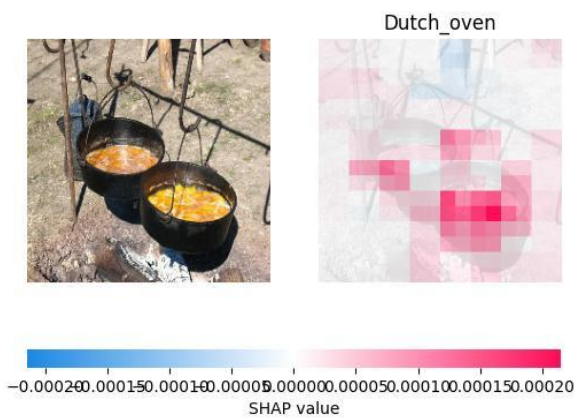
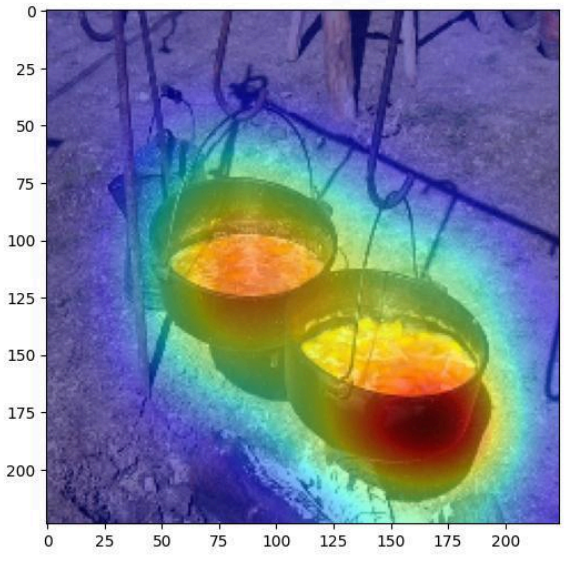
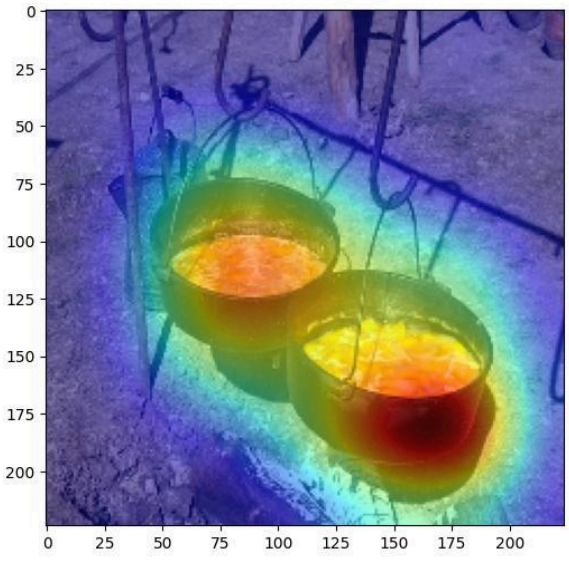
También es importante destacar el último caso analizado, donde se utiliza un “sticker” para engañar a la CNN respecto a la clasificación de la imagen. Este ejemplo ilustra de forma muy clara el tipo de comportamiento problemático que puede ocurrir en este ámbito y cómo estos métodos de visualización ofrecen la capacidad de detectarlos y proveer información visual de gran utilidad.

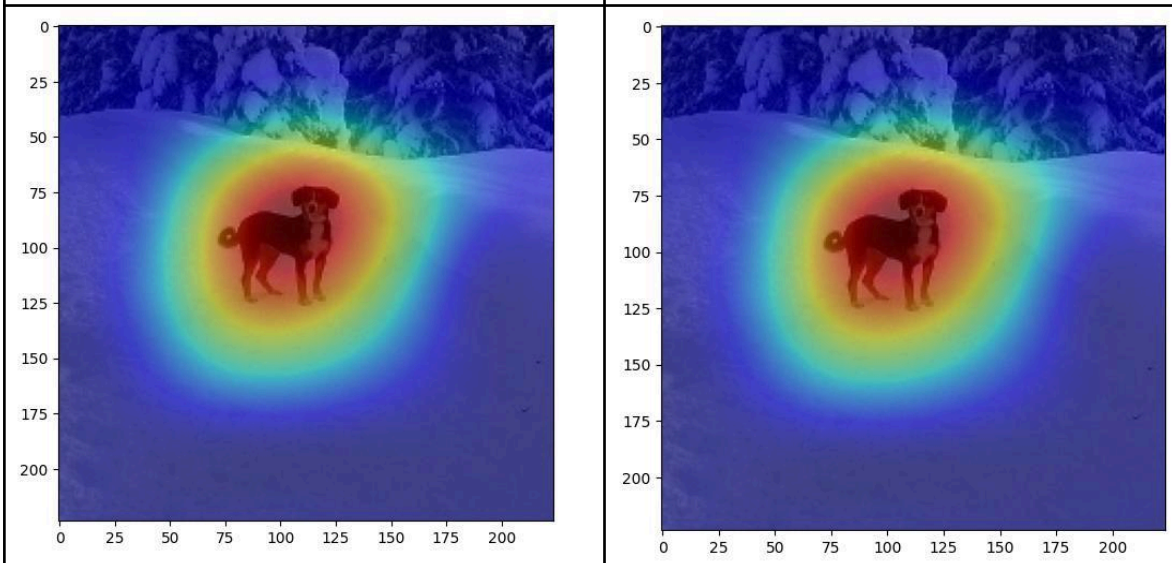
4.2.2 Visualización sobre los resultados de ResNet50

En la tabla 2 se muestran los resultados de clasificar, utilizando ResNet50, las mismas once imágenes analizadas con VGG16 en el apartado previo. De manera similar a la tabla 1, se presentan de manera intercalada la clase detectada por la red neuronal convolucional y debajo las aplicaciones de los métodos de visualización. En este caso se muestran cuatro imágenes, en la celda de arriba a la izquierda se encuentra la explicación

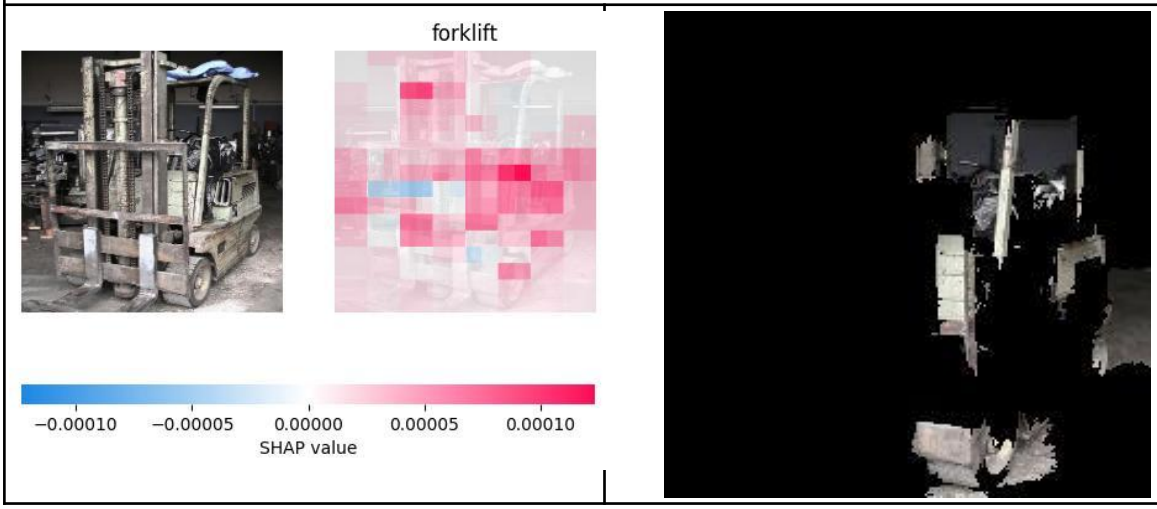
de SHAP y a su derecha la de LIME, abajo de estas pueden verse a la izquierda el resultado de CAM y a la derecha el de Grad-CAM. En las primeras seis imágenes se indica también la etiqueta correspondiente según ImageNet.

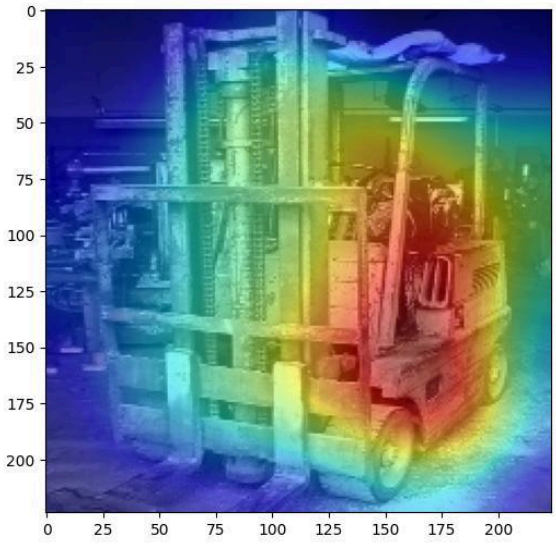
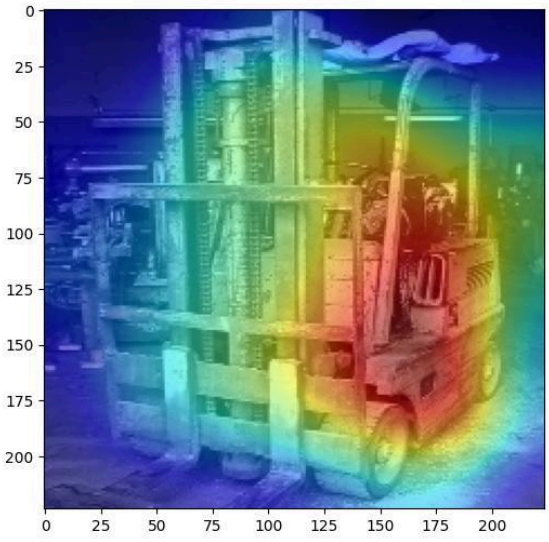
Tabla 2: Imágenes clasificadas por ResNet50 junto con las visualizaciones de las regiones destacadas por SHAP, CAM, Grad-CAM y LIME

Clase: Dutch Oven **Clase detectada**: Dutch Oven
 
 
Clase: Entlebucher **Clase detectada**: 1-Appenzeller 2-EntleBucher 3-Greater Swiss Mountain Dog

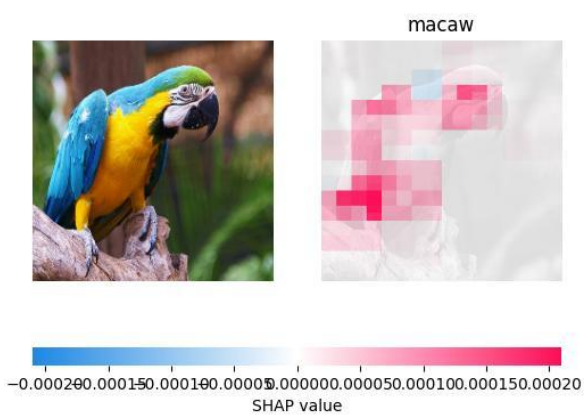


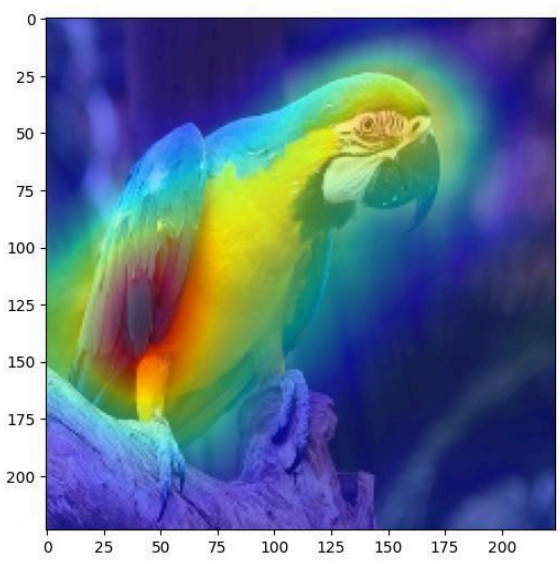
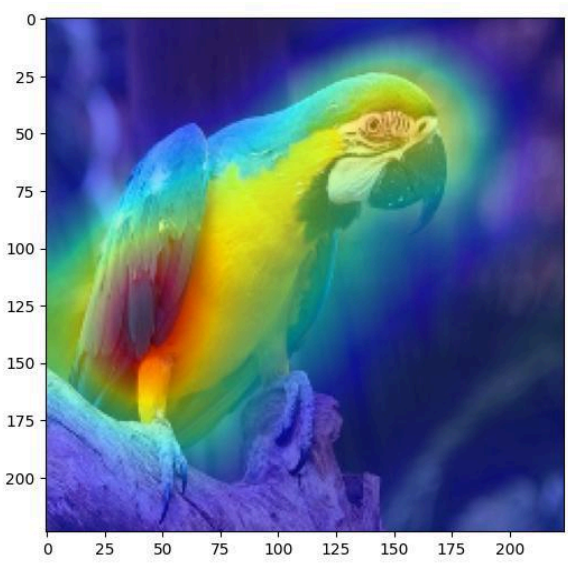
Clase: Forklift
Clase detectada: Forklift



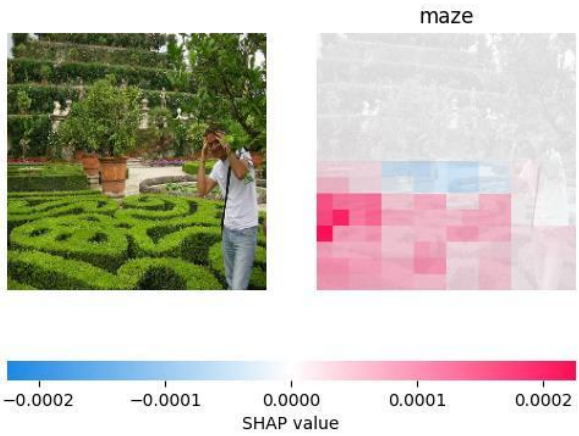



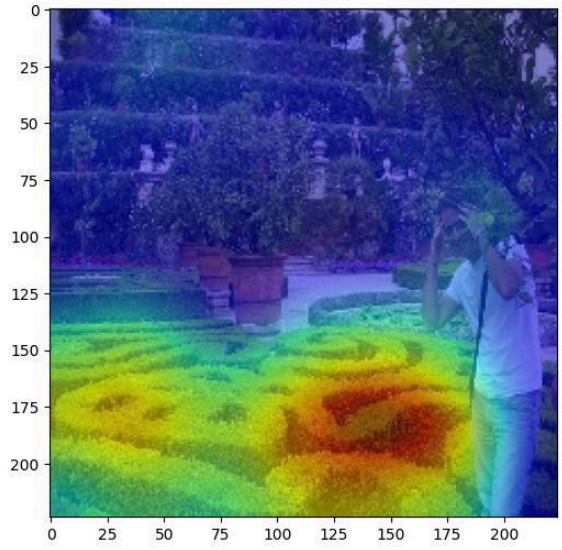
Clase: Macaw
Clase detectada: Macaw

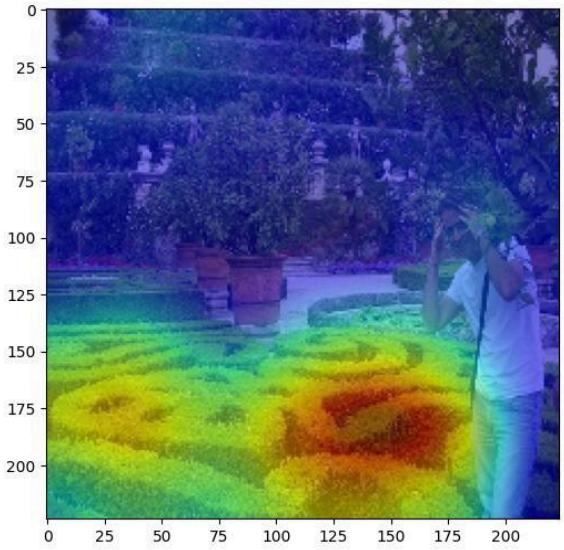


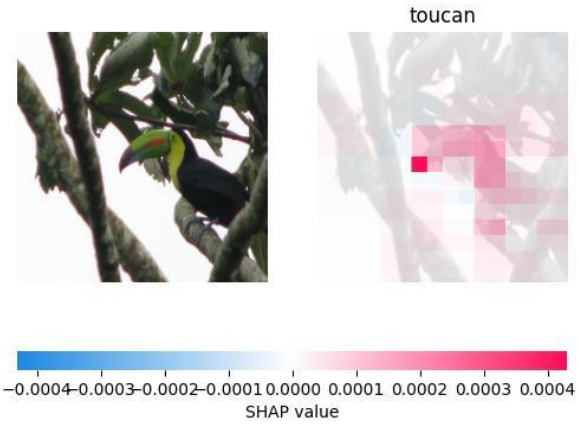
Clase: Maze
Clase detectada: Maze

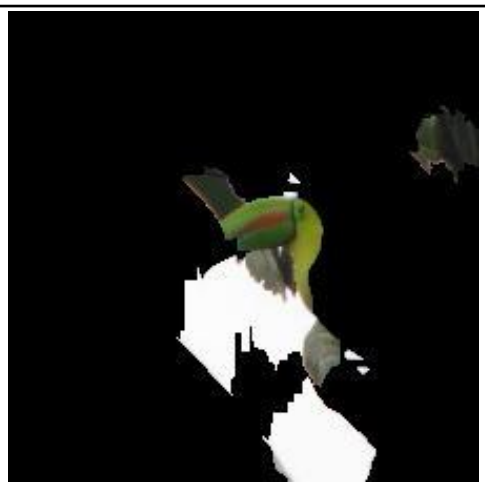


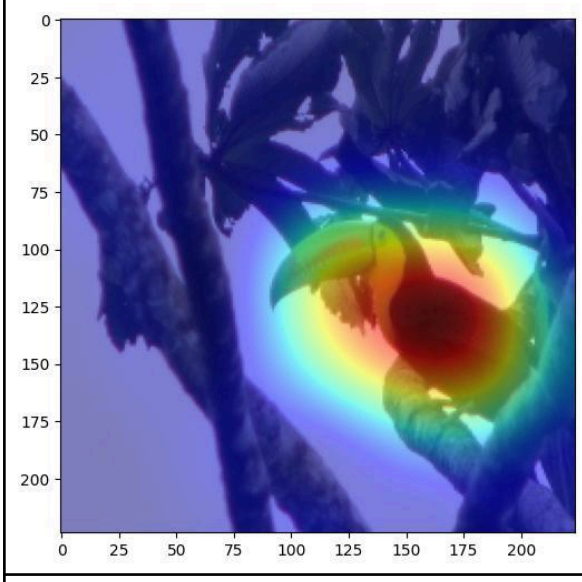
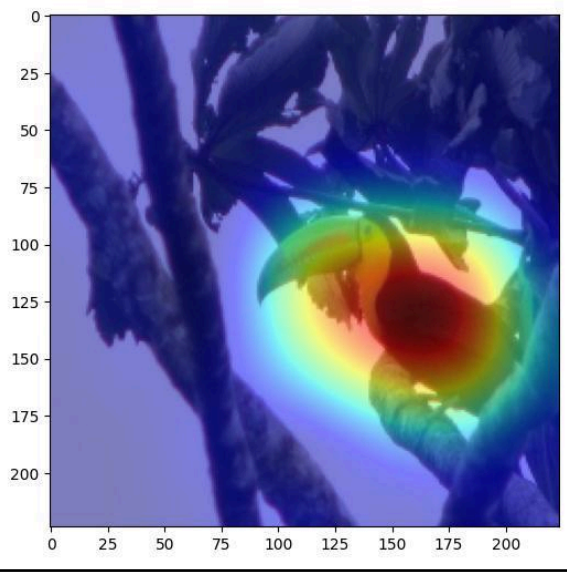




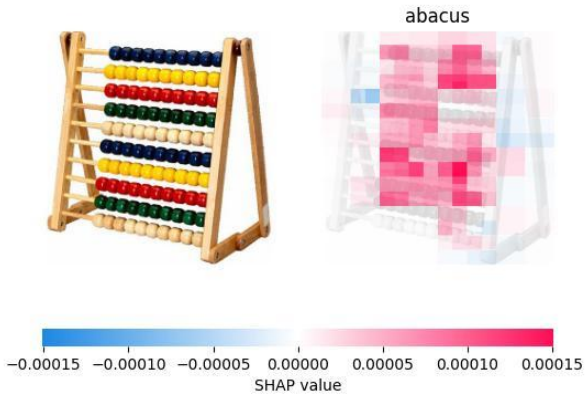
Clase: Toucan
Clase detectada: Toucan

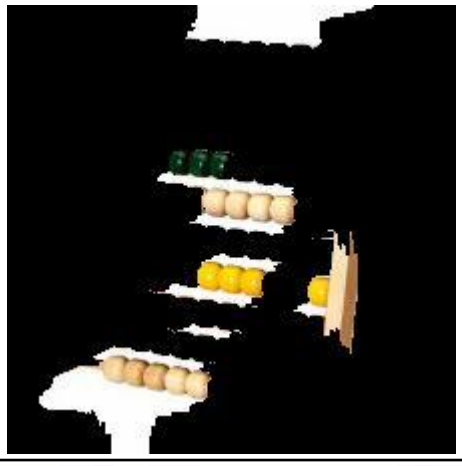
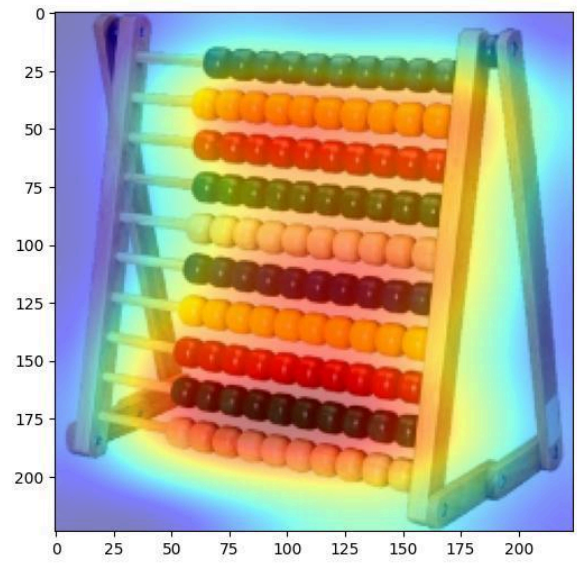
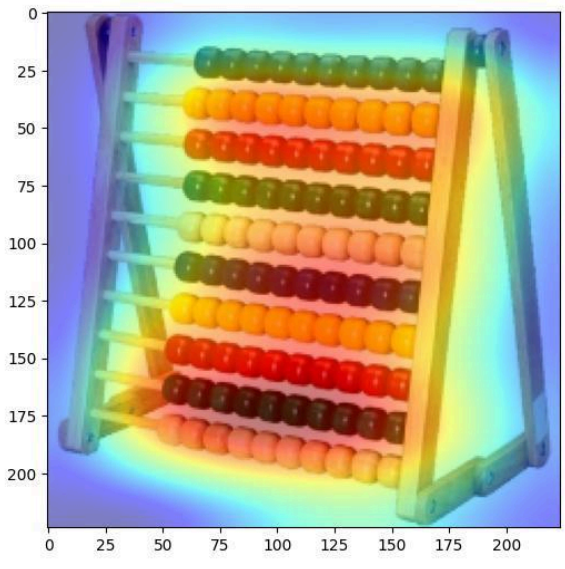




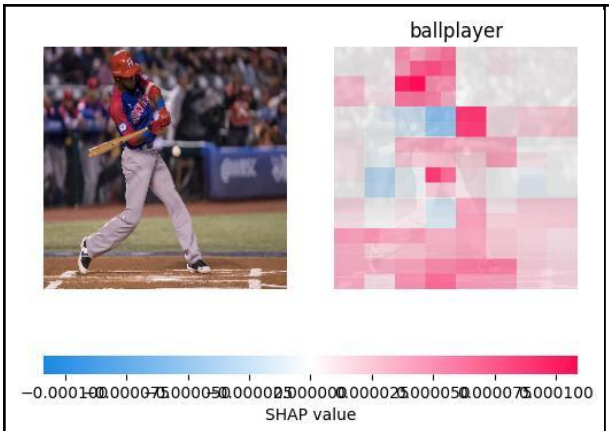



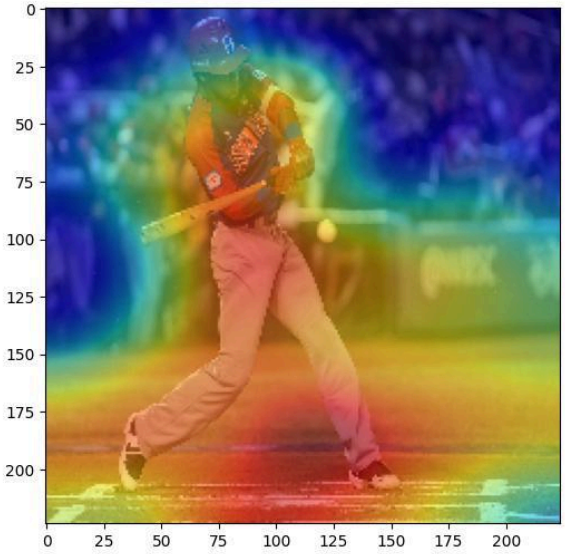
Clase detectada: Abacus

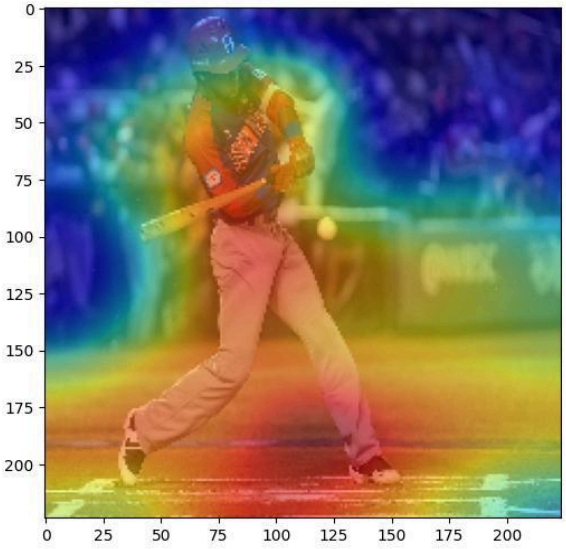


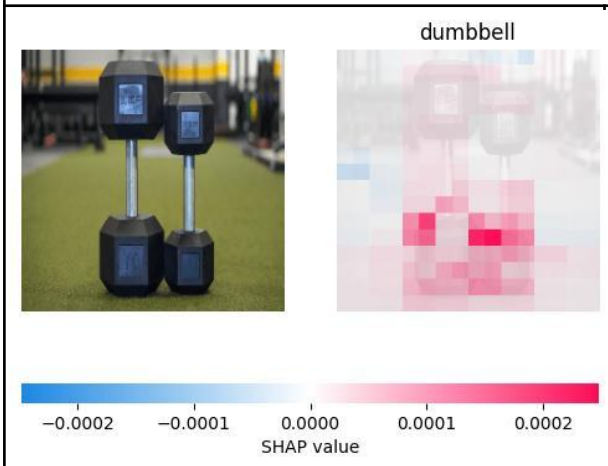
Clase detectada: Ballplayer

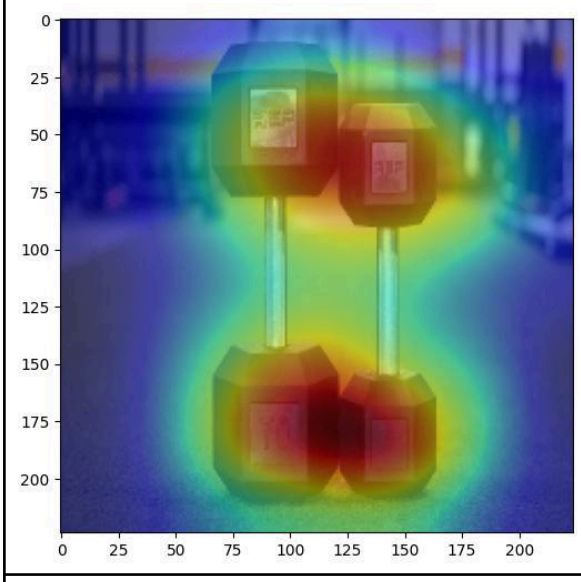


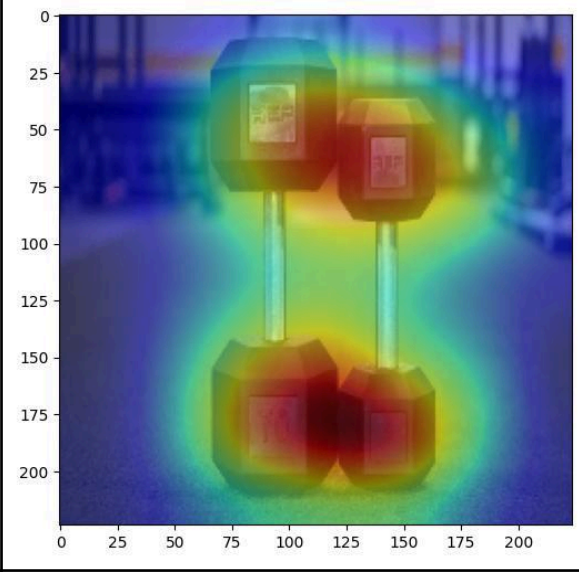




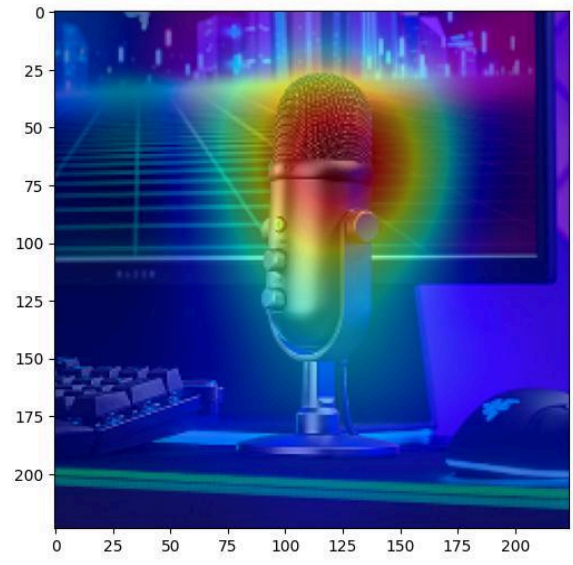
Clase detectada: Dumbbell

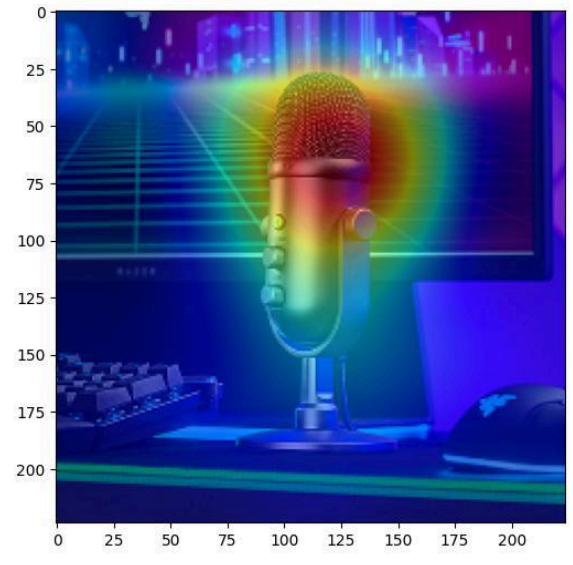




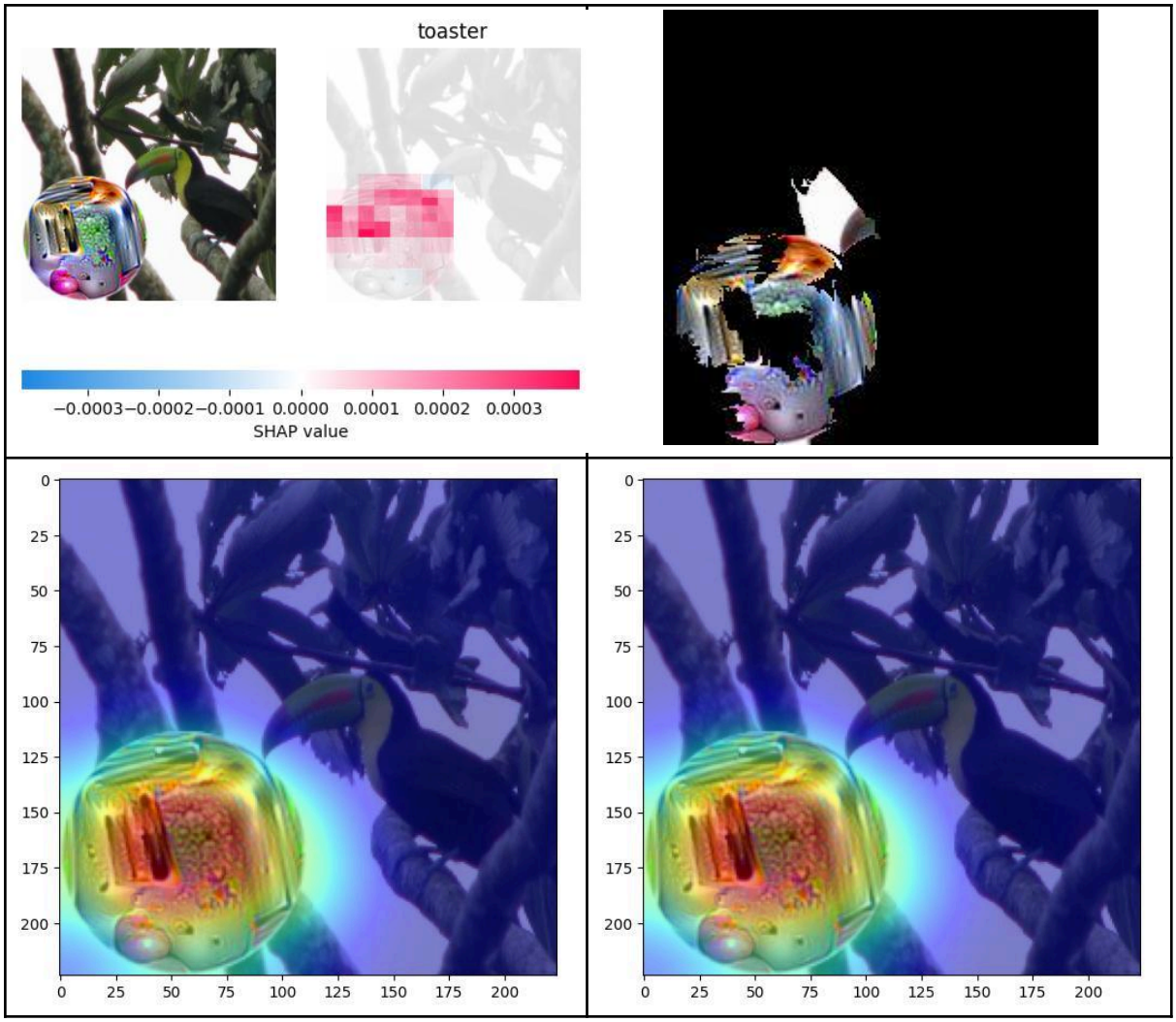


Clase detectada: Microphone





Clase detectada: Toaster



En general se observa que tanto las clasificaciones como las regiones más influyentes para cada clasificación realizadas por ResNet50, coinciden en su mayoría con los resultados obtenidos mediante el uso de VGG16.

El caso de “ballplayer” es diferente al observado previamente, ya que puede observarse que esta CNN la clasifica poniendo más énfasis en el jugador en sí, aunque se sigue viendo que el suelo donde se encuentra el jugador también afecta a dicha decisión.

En general se observa que tanto las clasificaciones como las regiones más influyentes para cada clasificación realizadas por ResNet50, coinciden en su mayoría con los resultados obtenidos mediante el uso de VGG16. Sin embargo, ambas presentan algunas diferencias. Respecto a las clasificaciones, en el caso de la segunda imagen “Entlebucher” ambas redes la clasifican erróneamente, indicando en el caso de VGG16 que es un Greater Swiss mountain dog, y en el caso de ResNet50, que es un Appenzeller, aunque en ambos casos se destaca la región correcta de la imagen.

Respecto a diferencias en las regiones más influyentes en las clasificaciones, en el caso de “ballplayer” puede observarse que esta CNN pone más énfasis en el jugador en sí, aunque puede verse que el suelo donde se encuentra el jugador también afecta a dicha decisión. Y en el caso de “abacus”, todos los métodos de visualización parecen indicar la

región más central de la imagen a diferencia de lo que ocurría en VGG16 donde cada método destacaba una región diferente.

4.3 Conclusiones e información general

Los cuatro métodos analizados parecen ser coherentes entre sí en la mayoría de los casos respecto a la información que presentan. Sin embargo es importante considerar un detalle sobre los métodos CAM y grad-CAM, ya que al presentar los resultados en forma de un “mapa de calor” tienden a perder precisión y detalle en la información detectada. Dicho de otro modo, LIME y SHAP permiten manipular las subdivisiones de la imagen a procesar, por lo que tienen la capacidad de analizar regiones más pequeñas de manera independiente. Esto tiene como resultado un análisis más detallado y preciso. Por ejemplo en la figura 24 puede verse que SHAP detecta que el caparazón contiene una región que afecta negativamente a la decisión de la red, mientras que en la figura 19 puede verse que CAM y grad-CAM detectan que el caparazón en su totalidad afecta positivamente.

Esa mayor precisión alcanzada por LIME y SHAP se debe a la capacidad de ajustar la cantidad de subdivisiones de la imagen a utilizar durante el análisis. En cambio, en CAM y Grad-CAM no existen parámetros modificables que permitan cambiar la información obtenida entre ejecuciones. Sin embargo, este análisis más detallado de parte de los primeros dos métodos requiere de un mayor tiempo de ejecución, ya que se requieren más iteraciones para procesar la mayor cantidad de subdivisiones. Por otro lado, CAM y Grad-CAM son extremadamente rápidos en comparación, requiriendo una fracción del tiempo para presentar un resultado al usuario. Además, la información que proveen puede resultar suficiente como una explicación útil.

Otra cuestión importante a tener en cuenta es la necesidad de procesar las imágenes de manera diferente para cada arquitectura a utilizar, lo cual implica realizar un preprocesamiento antes de realizar la elección del método de visualización en sí. Hasta ahora solo se realizaron modificaciones en las entradas debido a que se trabaja con ResNet50 y VGG16. Sin embargo, al momento de agregar nuevas redes a la herramienta será necesario investigar el preprocesamiento requerido por las entradas de cada una de ellas, aunque ésta es una preocupación que se debe abordar en el futuro.

Unos detalles para mencionar son, por un lado que en el caso de ResNet50 pueden verse explicaciones que resultan más coherentes con lo que un ojo humano espera a simple vista, y por otro lado, al utilizar SHAP y LIME para explicar clasificaciones de VGG16, suele requerirse aproximadamente el doble de tiempo. Esto se debe a la diferencia en la cantidad de parámetros de VGG, que con 138 millones es entre cinco y seis veces superior a los 23.5 millones de parámetros que posee ResNet50.

En este apartado se presentaron el proceso y los problemas ocurridos durante el desarrollo del código para cada método de visualización, junto a una muestra de las explicaciones de cada método para clasificaciones mediante las redes VGG16 y ResNet50. Además se resaltó la importancia de disponer de varios métodos de visualización para contrastar resultados y aumentar la confianza (o desconfianza) en la clasificación.

En el siguiente capítulo se presenta la herramienta propuesta que incorpora las múltiples implementaciones desarrolladas hasta el momento para los métodos de visualización analizados en este trabajo permitiendo analizar la clasificación obtenida para una imagen mediante CNNs.

Capítulo 5: Herramienta propuesta

En este capítulo se realiza un análisis sobre los frameworks y herramientas utilizados para la creación del programa, junto a los motivos de su uso. Posteriormente se mencionan decisiones tomadas respecto a la construcción de la herramienta en base a sus características, como el tipo y cantidad de usuarios, estructura interna y presentación visual. Por último se muestra brevemente el funcionamiento de la aplicación en sí.

La arquitectura pensada para la herramienta es la de una aplicación web, teniendo un frontend local accesible desde el navegador donde se ingresan los datos necesarios, los cuales son procesados y luego enviados a una función que realiza la clasificación de imágenes mediante las redes neuronales, y el análisis mediante métodos de visualización, haciendo uso de la GPU del usuario en caso de ser posible.

Si bien es posible obtener esta funcionalidad utilizando únicamente las librerías mencionadas previamente y las herramientas provistas por Python de manera nativa, se decide emplear frameworks y herramientas adicionales que faciliten la creación, implementación y distribución de la aplicación a realizar.

5.1 Herramientas y frameworks adicionales

En esta sección se provee información sobre las librerías y frameworks adicionales utilizados en el desarrollo del programa, junto a los motivos de su utilización.

Este apartado considera tres aspectos que influyen en el desarrollo de la herramienta. El primero es la aplicación web en si, para lo cual se utiliza el framework Flask; El segundo es la necesidad de utilizar una GPU para la clasificación de imágenes y el posterior análisis de la clasificación, (no es obligatorio realmente pero si se utiliza la CPU en su lugar aumenta significativamente el tiempo de procesamiento). Para ello, se requiere instalar el toolkit de CUDA (y cuDNN) para que TensorFlow pueda hacer uso de la GPU (éstas son librerías desarrolladas por Nvidia, por lo que están diseñadas para trabajar con GPUs de Nvidia). Por último, con el objetivo de garantizar la portabilidad y la facilidad de instalación en diferentes entornos, se ofrece la posibilidad de que sea descargada desde github y utilizada en diferentes computadoras sin necesidad de realizar muchos cambios manualmente utilizando el gestor de entornos Miniconda.

- ### Framework para la aplicación web: Flask [40]

Flask es un framework web ligero (un microframework) que provee un conjunto de herramientas y características que facilitan la creación de aplicaciones web en Python. Junto a otras librerías como “WTForms”[41] y “flask-WTF”[42] permite crear rápidamente una aplicación web. En el caso de este trabajo se trata de un formulario en HTML mediante el cual se ingresa la información necesaria, que luego es procesada y enviada a las funciones encargadas de realizar las clasificaciones y visualizaciones con la configuración proporcionada.

- Gestor de entornos - Miniconda [43]

Se trata de un sistema de gestión de paquetes y entorno open source, el cual permite especificar paquetes, librerías y sus versiones para su instalación en un entorno controlado, sin afectar la configuración global del sistema. De esta manera es posible tener un control de las versiones de librerías utilizadas, garantizando su funcionamiento y compatibilidad entre sí. La utilización de un gestor de paquetes y entorno fue necesaria para conseguir que TensorFlow utilice de manera nativa la GPU de un sistema con Windows. La última versión que soporta esta forma de uso es la 2.10, que es previa a la actual y tiene como requisito utilizar las versiones de CUDAtoolkit 11.2 y cuDNN 8.1.0. Estas dos últimas librerías son las causantes de abandonar el uso de virtualenv [44] (otro gestor de entornos) y cambiar a miniconda, ya que parte del proceso de instalar cuDNN requiere mover archivos a una carpeta de CUDA (posterior a la instalación de este último), procedimiento que es automatizado por miniconda pero no por virtualenv. Dado que el objetivo es crear una herramienta relativamente simple de usar, se busca minimizar la necesidad realizar modificaciones de manera manual por parte del usuario, especialmente durante la instalación.

- Librerías para uso de GPU - CUDA [45] y cuDNN[46]

El conjunto de herramientas CUDA, provee un entorno de desarrollo para crear aplicaciones aceleradas por GPU de altas prestaciones, mientras que cuDNN (NVIDIA CUDA Deep Neural Network) es una librería acelerada por GPU de primitivas para redes neuronales profundas (Deep Neural Networks), por lo que provee implementaciones para múltiples rutinas estándar, como convoluciones, pooling o normalización entre otras. Ambas librerías son necesarias para que TensorFlow pueda aprovechar la presencia de una GPU en el sistema [47], lo cual acelera en gran medida el procesamiento de imágenes, reduciendo el tiempo necesario para realizar una clasificación de quince minutos a solo uno.

En resumen, Flask agiliza la creación de aplicaciones web, Miniconda facilita la gestión y distribución de la aplicación, permitiendo crear un entorno que posee todo lo necesario para ejecutar el programa sin afectar el resto del sistema, y las librerías CUDA y cuDNN permiten aprovechar la presencia de una GPU en el sistema para reducir el tiempo necesario durante la ejecución.

5.2 Decisiones durante el desarrollo

En este apartado se explican múltiples características sobre la organización y desarrollo de la aplicación así como las razones que justifican las decisiones tomadas.

Debido a que se utiliza un frontend mediante un navegador web es necesario cargar las imágenes a una carpeta predefinida para de esta forma poder utilizar sus direcciones relativas, en vez de utilizar direcciones absolutas, ya que esto es inviable por razones de seguridad.

Por otro lado, al tratarse de una aplicación completamente local, hay elementos de seguridad que pueden ser ignorados (aunque las librerías utilizadas automatizan algunas de

estas cuestiones), ya que no hay peligro real de que un tercero cause problemas en el sistema. En este aspecto también se realizan validaciones (como que se completen los campos necesarios en el formulario) únicamente en el frontend.

Respecto a la aplicación en sí, desde un punto de vista más externo se tomaron consideraciones en relación a la usabilidad y claridad. Por ejemplo, se evita la aparición y desaparición repentina de menús en caso de ser opcionales, sino que siempre son visibles, pero en caso de no ser pertinentes se ven deshabilitados.

Mientras que desde un punto de vista más interno, el núcleo de la aplicación manipula y procesa la información de una manera en la que resulte sencillo a futuro agregar nuevos elementos como redes neuronales o métodos de visualización, ya que el código encargado de esto se encuentra organizado en secciones claramente definidas e identificables.

En general se adoptaron las medidas de seguridad mínimas, ya que la aplicación no está pensada para utilizarse como servidor remoto. Además se organiza la información y el código con el objetivo de facilitar su escalabilidad a futuro.

5.3 Funcionamiento de la aplicación

En esta sección se muestra brevemente el funcionamiento del programa.

Actualmente la aplicación [48] se encuentra en GitHub, y una vez descargada y realizados los pasos indicados en el archivo “README”, esta debería encontrarse funcional y accesible desde “http://127.0.0.1:5000” en un navegador web.

Cuando el usuario accede a través del navegador, se le presenta el formulario mostrado en la figura 28. En este formulario se debe seleccionar la configuración a utilizar, indicando la imagen que será clasificada y analizada con un método de visualización, lo que a su vez implica seleccionar una o más redes neuronales con las cuales clasificar dicha imagen, y uno o más métodos de visualización para analizar dicha clasificación. En caso de no haber seleccionado al menos una red y un método, el botón de “submit” se mantendrá deshabilitado para impedir el ingreso de información incorrecta.

Además, los campos correspondientes a los métodos de visualización se encuentran deshabilitados por defecto, evitando la transferencia de información adicional innecesaria, al mismo tiempo que permite ver los campos que se deberían rellenar en caso de seleccionar dicho método aun antes de hacerlo. Esto último es para evitar confusión debido a la posible sorpresa que habría en el caso de que los campos aparecieran y desaparecieran según las selecciones realizadas.

En el caso de querer utilizar una red propia, se deben respetar ciertas restricciones, ya que actualmente el sistema permite utilizar redes creadas mediante *transfer learning* en base a la arquitectura VGG16 y analizar sus decisiones mediante SHAP y LIME. Para esto la red debe encontrarse como un archivo de formato hdf5 (archivos con extensión .h5) en la carpeta “/static/files/models”. Además, se deben ingresar las categorías por las cuales la red clasifica imágenes separadas por comas.

Debido a las capacidades actuales de la implementación, no es posible utilizar el método Grad-CAM para analizar estas redes propias del usuario, pero la integración de esta capacidad queda como trabajo futuro.

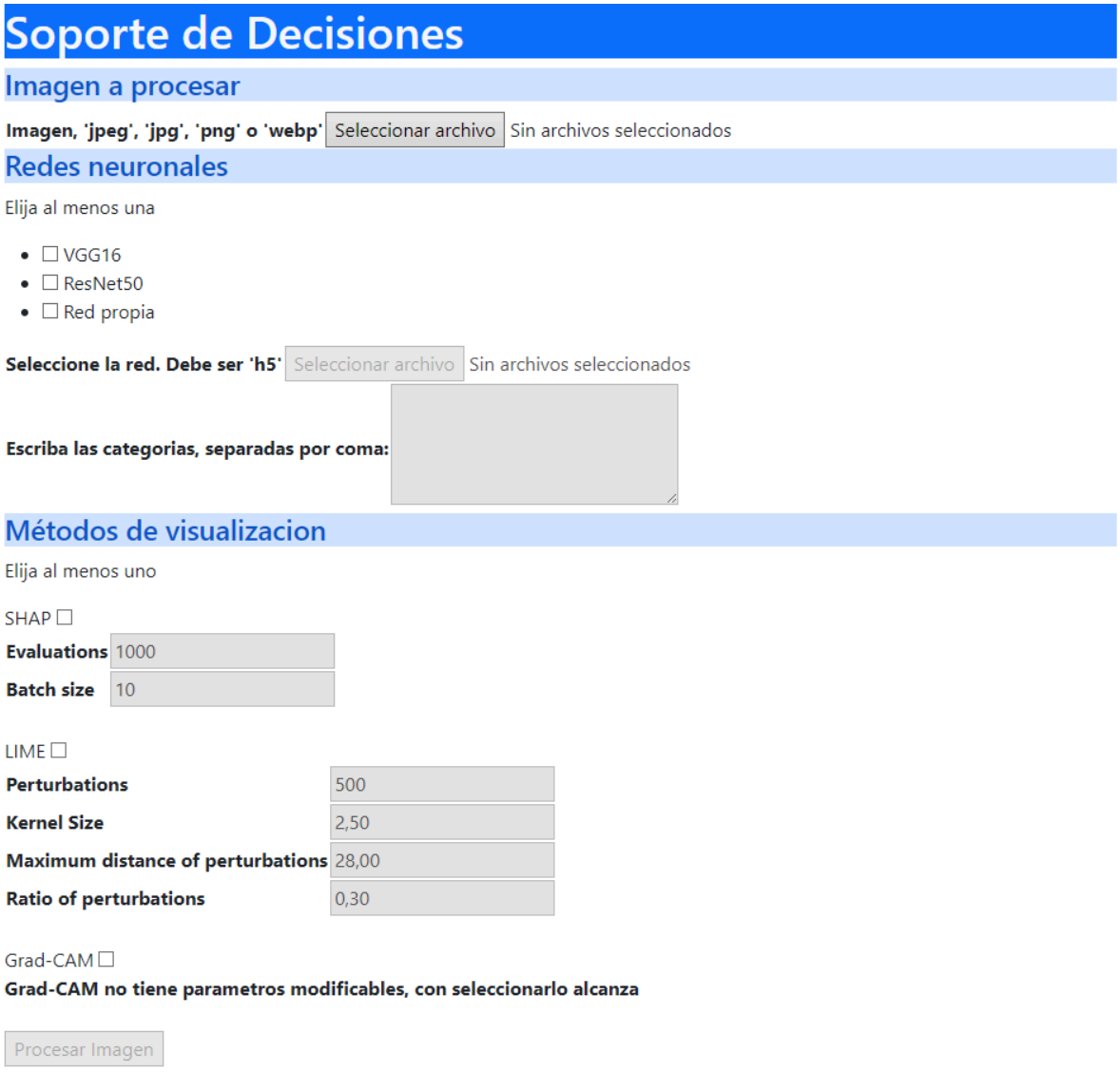


Figura 28: Formulario de la aplicación, accesible mediante un navegador web.

5.3.1 Configuración del método SHAP

Para optimizar la aplicación y brindar al usuario la libertad de priorizar el tiempo requerido o el nivel de detalle deseado, se ofrece la posibilidad de configurar algunos parámetros del método SHAP, como la cantidad de evaluaciones y el tamaño del lote.

El campo de "*Evaluations*" es el que permite indicar la cantidad de evaluaciones, lo que se traduce en la cantidad de permutaciones de contribuciones a analizar, por lo que mientras mayor sea este numero, mayor cantidad de análisis realizados, mayor

granularidad, pero también un mayor tiempo necesario para el procesamiento de la información

El campo de *Batch Size* es el que representa el tamaño de lote, es decir, la cantidad de muestras utilizadas en cada pasada, a mayor batch size, mayor velocidad, pero también hace falta utilizar más memoria, y suele implicar una disminución en la precisión.

Los valores recomendados son cercanos a 1000 para las evaluaciones y 10 para el tamaño de lote, ya que esto permite un buen balance entre detalle y tiempo requerido para su procesamiento. Es importante considerar que el tamaño de lote se encuentra limitado por la memoria disponible en la GPU, por lo tanto si se supera ese valor se pierden los beneficios de utilizarlo. Respecto a las evaluaciones, con valores muy pequeños la información obtenida no es de utilidad ya que no se pueden distinguir adecuadamente las regiones que más afectan a la clasificación. En el caso de valores mucho mayores, llega un punto en que aún aumentando la cantidad no provee nueva información, pero se incrementa el tiempo de ejecución. Esos valores recomendados fueron implementados como valores por defecto, como se muestra en el formulario presentado en la figura 29 para SHAP.

5.3.2 Configuración del método LIME

Con los mismos objetivos que los planteados con el método SHAP, se ofrece la posibilidad de configurar algunos parámetros del método LIME, como la cantidad de perturbaciones a analizar y tres parámetros que afectan a la división de la imagen en super-pixeles, los cuales son tamaño de kernel, distancia máxima de las perturbaciones y proporción de las perturbaciones.

El parámetro *Perturbations* representa la cantidad de permutaciones a analizar. Son las combinaciones posibles de super-pixeles habilitados y deshabilitados, por lo que mientras mayor sea el número, el tiempo de procesamiento se incrementa. Sin embargo, se obtiene una mayor certeza en que los super-pixeles presentados son los que más afectaron en la clasificación de la imágen.

El parámetro *Kernel size* afecta a la generación de super-pixeles en sí. La función que los genera utiliza una distribución gaussiana, y el kernel size se corresponde con la desviación estándar, por lo que un mayor número resulta en un mayor tamaño de los super-pixeles pero en menor cantidad.

Maximum distance of perturbation también está relacionado con la generación de super-pixeles, indicando un punto de corte para las distancias de los datos, por lo que un valor mayor implica un tamaño mayor de los super-pixeles y una menor cantidad.

Ratio of perturbations equilibra la proximidad del espacio de color y la proximidad del espacio de imagen. Los valores más altos dan más peso al espacio de color. Es decir, altera la forma de los super-pixeles generados, causando que estos se guien mas por las diferencias de color en la imagen.

Los valores recomendados para estos parámetros fueron obtenidos empíricamente y representan un punto medio entre detalle y tiempo de procesamiento, por lo que se implementaron como valores por defecto, como se muestra en la figura 29 en la sección correspondiente a LIME.

5.3.3 Configuración del método Grad-CAM

Este formulario no requiere ninguna configuración debido al funcionamiento del método Grad-CAM en sí, como este realiza la media aritmética de los gradientes de salida para una clase con respecto a cada mapa de activaciones de una capa concreta, el procedimiento siempre es el mismo.

5.4 Procesamiento de la imagen

Una vez realizada la configuración, se comienza a procesar la imagen. En la figura 29 puede verse primero el resultado del modelo seleccionado mostrando las cinco clasificaciones con mayor probabilidad y sus porcentajes de certeza. Luego comienzan a ejecutarse los métodos de visualización seleccionados en el formulario. En la parte inferior se muestra el tiempo restante estimado para terminar.

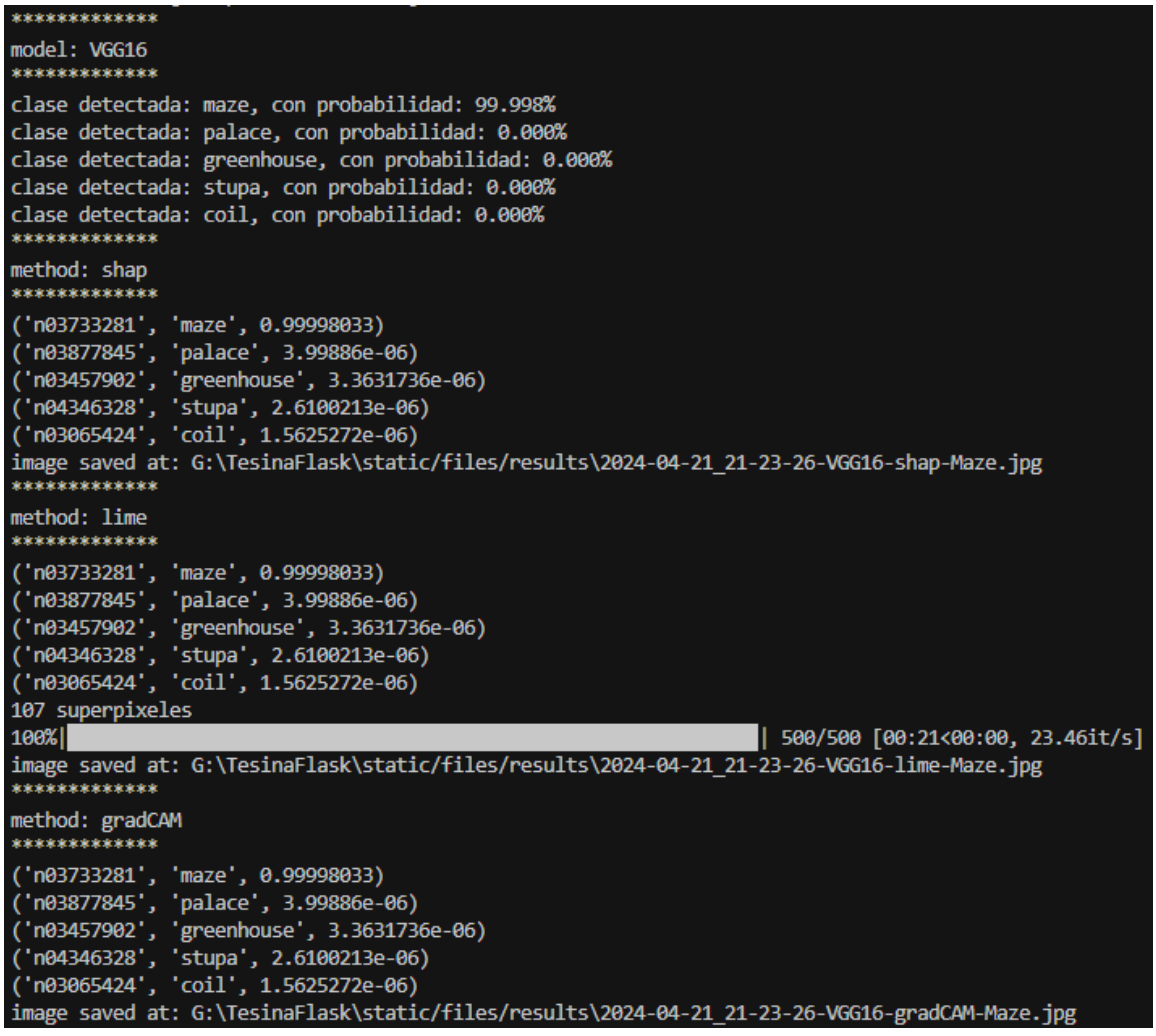


Figura 29: Consola durante la ejecución de la aplicación.

Cuando se termina de procesar la imagen, cada combinación de red y método es presentada en pantalla como puede observarse en la figura 30. Además los resultados son almacenados en la carpeta de la aplicación “static/files/results” bajo un nombre formado por la fecha y hora del procesamiento, la red, el método y el nombre original de la imagen.

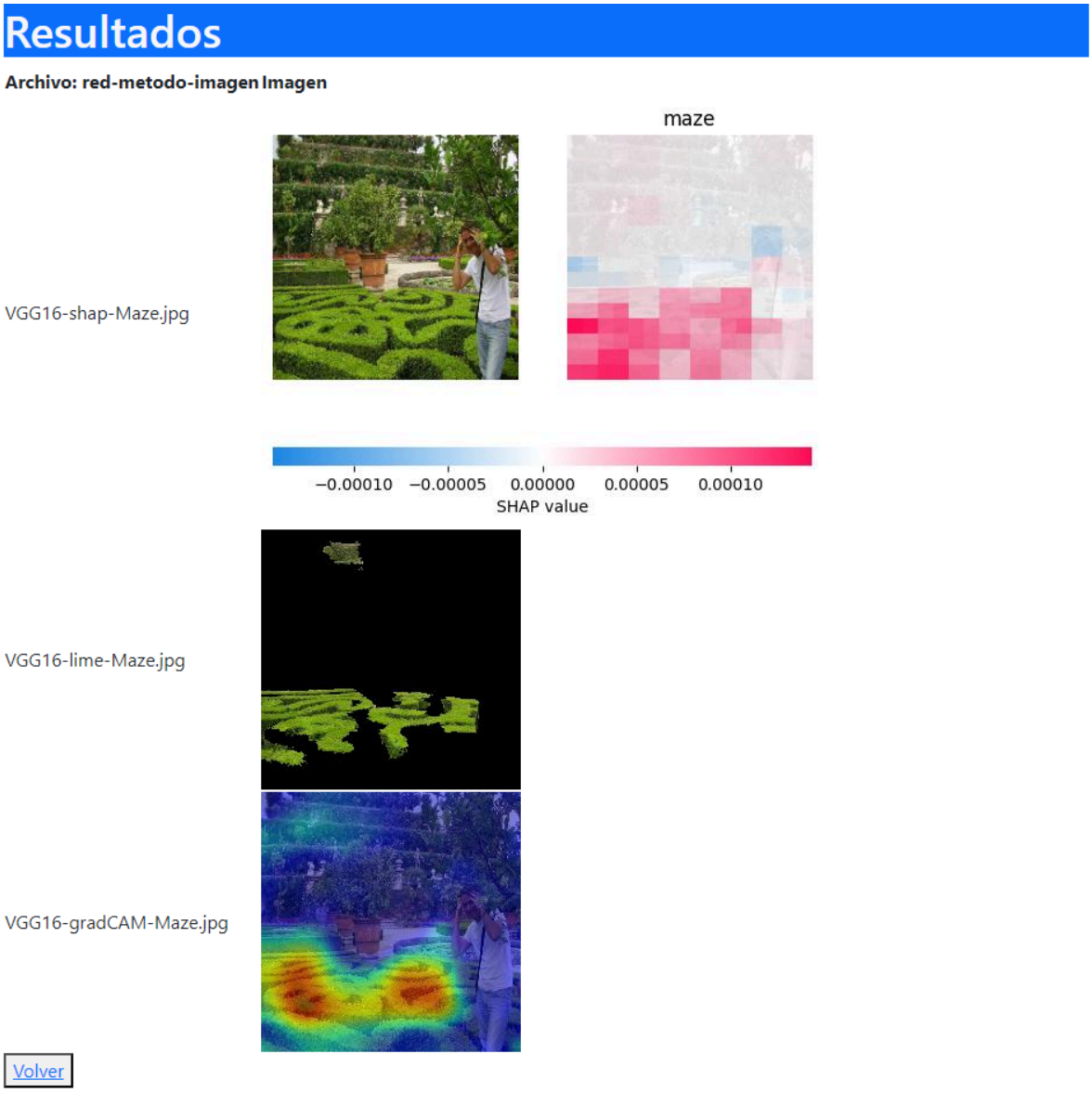


Figura 30: Resultado final tras la ejecución de la aplicación.

Un detalle para considerar es que si se utilizan placas de video más modernas que las soportadas por la versión de TensorFlow que se utiliza, es necesario tener los drivers de estas actualizados, ya que estos pasan a encargarse del manejo de la GPU en sí.

La aplicación es desarrollada con la intención de que tanto su instalación como su uso resulten sencillos al usuario. Luego de instalar miniconda es posible crear el entorno en unas pocas líneas en la consola, y tras iniciar el programa se debe completar un formulario con varios valores predefinidos y esperar unos momentos para obtener los resultados.

Capítulo 6: Conclusiones y trabajos futuros

El objetivo de este trabajo es la realización de un programa en el cual se incorporan múltiples métodos de visualización para justificar la toma de decisiones en redes neuronales destinadas a la clasificación de imágenes. De esta forma se consiguen representaciones visuales, en base a las cuales poder contrastar los resultados obtenidos, en una forma simplificada y más accesible, lo que implica que el funcionamiento de dichas redes sea más comprensible, obteniendo de esta forma, para un caso particular, una explicación del funcionamiento interno de estas redes comúnmente conocidas como “cajas negras”.

Como se indica en apartados previos de este trabajo, el objetivo pareciera cumplido durante el desarrollo de esta tesina, ya que efectivamente se consigue crear el programa. Además de esto, en la realización de este trabajo se analizan los conceptos y características de las redes neuronales convolucionales y métodos de visualización, centrándose en cuatro de estos últimos, CAM, Grad-CAM, LIME y SHAP.

Uno de los factores que más afecta a la incorporación de múltiples redes neuronales convolucionales es la necesidad de pre procesar la información a ser ingresada (en este caso, las imágenes). Esto se debe a que el simple hecho de tener que alterar la imagen a clasificar en algunos casos, como tener que comprimir el tamaño a un máximo de 224 x 224 píxeles al trabajar con ResNet 50 o VGG-16, puede causar distorsiones que afecten al resultado. Además del hecho de que cada red necesita un pre procesamiento diferente, ya que en el caso de inception v3 por ejemplo, aceptan entradas hasta de tamaño 299 x 299 x 3 píxeles.

Relacionado a esto último se encuentra el hecho de que los métodos de visualización actualmente realizan el pre procesamiento de dichas imágenes de manera interna en las funciones donde se emplea la técnica de visualización. Siendo el caso más concreto el código para el método LIME en el cual se recibe la ubicación de la imagen en el sistema de archivos, y esta es cargada por dos librerías de manipulación de imágenes, siendo una la encargada del pre procesamiento para la posterior clasificación mediante la CNN, y la otra encargada de la generación de los super-pixeles y permutaciones para la aplicación del método LIME.

Estas características mencionadas no implican que sea imposible el incrementar las capacidades de la aplicación desarrollada, sino que son aspectos importantes a tener en cuenta a futuro al ampliar sus prestaciones. Además de ser una muestra de los elementos a tener en cuenta al desarrollar un programa de este estilo.

Entre los posibles cambios a realizar sobre este programa en trabajos futuros, hay una gran cantidad y variedad de opciones ya que es simplemente una primera versión. Esto se debe a que por ejemplo hay múltiples redes neuronales convolucionales que pueden ser agregadas, y no solamente las redes más genéricas, sino que debería ser posible incorporar implementaciones de redes especializadas y diseñadas para realizar trabajos más específicos. Aunque en estos casos hay que considerar los métodos de visualización utilizados actualmente, ya que algunos tienen limitaciones en relación al tipo de CNN que

pueden analizar debido a sus características teóricas, mientras que otros métodos al basarse en librerías externas también pueden verse afectados por el mismo tipo de limitaciones.

Esto último lleva a otras posibilidades de enriquecer al programa, ya que por un lado existe la posibilidad de agregar nuevos métodos de visualización, lo que ampliaría las herramientas disponibles para el análisis de redes, y por otro lado, la posibilidad (y hasta cierto punto necesidad) de modificar los métodos incorporados para mantenerlos actualizados y optimizados y de esa forma facilitar la futura incorporación de nuevas arquitecturas de redes para que puedan ser analizadas.

Referencias

[1] Besse, Philippe & Castets-Renard, Céline & Garivier, Aurélien & Loubes, Jean-Michel. (2018). Can Everyday AI be Ethical? Machine Learning Algorithm Fairness (english version). 10.13140/RG.2.2.22973.31207.

[2] Brown, Tom & Mané, Dandelion & Roy, Aurko & Abadi, Martín & Gilmer, Justin. (2017). Adversarial Patch.

[3] Esteva, A., Kuprel, B., Novoa, R. *et al.* Dermatologist-level classification of skin cancer with deep neural networks. *Nature* **542**, 115–118 (2017). https://doi.org/10.1038/nature21056

[4] Reddysetty S. Traditional Programming vs Machine Learning - Sravya Reddysetty - Medium. Medium [Internet]. 2022. Available from: https://sravya-tech-usage.medium.com/traditional-programming-vs-machine-learning-e9bbed5e491c, last accessed 04/2024.

[5] "Visualizando neuronas en redes neuronales convolucionales" David Erroz Arroyo, 2019, Academica-e (https://academica-e.unavarra.es/xmlui/handle/2454/33694)

[6] Brownlee J. Why Initialize a Neural Network with Random Weights? MachineLearningMastery.com [Internet]. 2022. Available from: https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/, last accessed 04/2024.

[7] How do you balance the trade-off between regularization and complexity in CNNs? www.linkedin.com [Internet]. 2024. Available from: https://www.linkedin.com/advice/1/how-do-you-balance-trade-off-between-regularization?locale=en, last accessed 04/2024.

[8] RMSProp | Interactive Chaos [Internet]. [date unknown]. Available from: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/rmsprop#:~:text=RMSProp%20o%20Root%20Mean%20Square,exponencial%20ponderada%20para%20suavizar%20los, last accessed 04/2024.

[9] Ioffe, Sergey & Szegedy, Christian. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[10] Navarro S. ¿Qué es Batch-Normalization para red convolucional? KeepCoding Bootcamps [Internet]. 2024. Available from: https://keepcoding.io/blog/batch-normalization-red-convolucional/, last accessed 04/2024.

[11] Sarkar, D., Bali, R., & Sharma, T. (2017, December 20). Practical Machine Learning with Python. Apress. http://books.google.ie/books?id=9ClEDwAAQBAJ&printsec=frontcover&dq=978-1-4842-3207-1&hl=&cd=1&source=gbs_api

[12] “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position” Fukushima Kunihiko, 1980, https://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf

[13] Aplicación de Deep Learning en Robótica Móvil para Exploración y Reconocimiento de Objetos basados en Imágenes - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Red-neuronal-convolucional-4_fig7_308783857, last accessed 04/2024

[14] Navarro S. Tipos de capas de red neuronal convolucional. KeepCoding Bootcamps [Internet]. 2024. Available from: https://keepcoding.io/blog/tipos-capas-red-neuronal-convolucional/, last accessed 04/2024.

[15] Vignesh S. The world through the eyes of CNN. - Analytics Vidhya - Medium. Medium [Internet]. 2021. Available from: https://medium.com/analytics-vidhya/the-world-through-the-eyes-of-cnn-5a52c034dbeb, last accessed 04/2024.

[16] Grill-Spector, Kalanit & Malach, Rafael. (2004). The human visual cortex. Annual review of neuroscience. 27. 649-77. 10.1146/annurev.neuro.27.070203.144220.

[17] Zhou, Bolei & Khosla, Aditya & Lapedriza, Àgata & Oliva, Aude & Torralba, Antonio. (2016). Learning Deep Features for Discriminative Localization. 10.1109/CVPR.2016.319.

[18] Rs, Ramprasaath & Cogswell, Michael & Das, Abhishek & Vedantam, Ramakrishna & Parikh, Devi & Batra, Dhruv. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. 618-626. 10.1109/ICCV.2017.74.

[19] Ribeiro, Marco & Singh, Sameer & Guestrin, Carlos. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. 1135-1144. 10.1145/2939672.2939778.

[20] Lundberg, Scott & Lee, Su-In. (2017). A Unified Approach to Interpreting Model Predictions.

[21] Garbin C. Machine learning interpretability with feature attribution. Christian Garbin’s personal blog [Internet]. 2022. Available from: https://cgarbin.github.io/machine-learning-interpretability-feature-attribution/#shapley-values, last accessed 04/2024.

[22] Imagenet Homepage. [Online]. Available: http://www.image-net.org, last accessed 04/2024.

[23] Lecun, Yann & Bottou, Leon & Bengio, Y. & Haffner, Patrick. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE. 86. 2278 - 2324. 10.1109/5.726791.

[24] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

[25] Hassan MU. VGG16 – Convolutional Network for Classification and Detection. Neurohive / Neural networks [Internet]. 2023. Available from: https://neurohive.io/en/popular-networks/vgg16/, last accessed 04/2024.

[26] He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

[27] Szegedy, Christian & Liu, Wei & Jia, Yangqing & Sermanet, Pierre & Reed, Scott & Anguelov, Dragomir & Erhan, Dumitru & Vanhoucke, Vincent & Rabinovich, Andrew. (2014). Going Deeper with Convolutions.

[28] Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. How transferable are features in deep neural networks?. Adv. Neural Inf. Process. Syst. 20, 3320–3328 (2014).

[29] Weiss, K., Khoshgoftaar, T. M. & Wang, D. A survey of transfer learning. J. Big Data 3, 9. https://doi.org/10.1186/s40537-016-0043-6 (2016).

[30] Tan, C. et al. A survey on deep transfer learning. In International Conference on Artificial Neural Networks, 270–279. https://doi.org/10.1007/978-3-030-01424-7_27 (2018).

[31] “Keras” https://keras.io/about/, last accessed 04/2024.

[32] Seachaos. Get Heatmap from CNN (Convolution Neural Network), AKA CAM. Medium [Internet]. 2022. Available from: https://tree.rocks/get-heatmap-from-cnn-convolution-neural-network-aka-grad-cam-222e08f57a34, last accessed 04/2024.

[33] Google Colaboratory [Internet]. Implementación CAM. Available from: https://colab.research.google.com/drive/1vydSKv-oh7BDfPTz-Z6MZdIKVz2t8T_d?usp=sharing, last accessed 04/2024.

[34] Team K. Keras documentation: Grad-CAM class activation visualization [Internet]. [date unknown]. Available from: https://keras.io/examples/vision/grad_cam/, last accessed 04/2024.

[35] Google Colaboratory [Internet]. Implementación Grad-CAM. Available from: https://colab.research.google.com/drive/1IO1qwcDBKQrwCAdJ_XdhT_cfGzKF8IcJ?usp=sharing, last accessed 04/2024.

[36] Explain ResNet50 ImageNet classification using Partition explainer — SHAP latest documentation [Internet]. [date unknown]. Available from: https://shap.readthedocs.io/en/latest/example_notebooks/image_examples/image_classification/Image%20Multi%20Class.html, last accessed 04/2024.

[37] “Interpretable Machine Learning with LIME for Image Classification” [Internet]. [date unknown], Cristian Arteaga https://nbviewer.org/url/arteagac.github.io/blog/lime_image.ipynb, last accessed 04/2024.

[38] Google Colaboratory [Internet]. Implementación LIME. Available from: https://colab.research.google.com/drive/1ttAy3KkniwSIu9NmoVwCYSO_KOWCtWbD?usp=sharing, last accessed 04/2024.

[39] Google Colaboratory [Internet]. Implementación SHAP. Available from: https://colab.research.google.com/drive/1XU2BCtwRUrt3AyHTLXOgqztHrLgEoR3V?usp=sharing, last accessed 04/2024.

[40] Welcome to Flask — Flask Documentation (3.0.x) [Internet]. [date unknown]. Available from: https://flask.palletsprojects.com/en/3.0.x/, last accessed 04/2024.

[41] WTForms — WTForms Documentation (3.1.x) [Internet]. [date unknown]. Available from: https://wtforms.readthedocs.io/en/3.1.x/, last accessed 04/2024.

[42] Flask-WTF — Flask-WTF Documentation (1.2.x) [Internet]. [date unknown]. Available from: https://flask-wtf.readthedocs.io/en/1.2.x/, last accessed 04/2024.

[43] Miniconda — Anaconda documentation [Internet]. [date unknown]. Available from: https://docs.conda.io/projects/miniconda/en/latest/, last accessed 04/2024.

[44] virtualenv [Internet]. [date unknown]. Available from: https://virtualenv.pypa.io/en/latest/, last accessed 04/2024.

[45] NVIDIA CUDA Toolkit - Free Tools and Training. NVIDIA Developer [Internet]. [date unknown]. Available from: https://developer.nvidia.com/cuda-toolkit, last accessed 04/2024.

[46] NVIDIA CUDA Deep Neural Network (cuDNN). NVIDIA Developer [Internet]. [date unknown]. Available from: https://developer.nvidia.com/cudnn, last accessed 04/2024.

[47] Install TensorFlow with pip. TensorFlow [Internet]. [date unknown]. Available from: https://www.tensorflow.org/install/pip, last accessed 04/2024.

[48] KaitoKurogami. GitHub - KaitoKurogami/FlaskTesina-Temp-Name-. GitHub [Internet]. [date unknown]. Available from: https://github.com/KaitoKurogami/FlaskTesina-Temp-Name-, last accessed 04/2024.

[49] TensorFlow. TensorFlow [Internet]. [date unknown]. Available from: https://www.tensorflow.org, last accessed 04/2024.

Apéndice 1 Herramientas/Frameworks en las implementaciones de las redes convolucionales

En este apartado se mencionan algunas librerías y demás herramientas computacionales utilizadas durante las primeras etapas de este trabajo, especialmente, aquellas inmediatamente relacionadas a la utilización de redes neuronales convolucionales.

Python - Numpy:

Python es un lenguaje de programación interpretado y multiparadigma, soportando programación orientada a objetos, imperativa y funcional, además de ser multiplataforma.

Es uno de los lenguajes más utilizados en Machine Learning debido a su simplicidad y la gran cantidad de librerías disponibles y de fácil acceso. Viéndose ayudado por el hecho de que es código abierto y su gran capacidad de manejo de datos.

Una de las extensiones de Python más populares es NumPy, que agrega mayor soporte para vectores y matrices, constituyendo una biblioteca matemática de alto nivel para operar con esos vectores, resultando muy útil cuando se trabaja con Redes Neuronales, ya que operan con gran cantidad de matrices y datos.

También hay que destacar la librería MatPlotLib, que genera gráficos a partir de datos en arreglos, por lo que su uso también está ligado a NumPy.

Keras [31]:

Esta biblioteca de Redes Neuronales de código abierto escrita en Python, está especialmente diseñada para facilitar una rápida experimentación en deep learning. permite crear prototipos rápidos de redes profundas y llevar a cabo su entrenamiento de forma cómoda y rápida.

Es un framework de alto nivel y está diseñado para correr sobre otros frameworks de más bajo nivel como TensorFlow o PyTorch.

TensorFlow [49]:

Es una biblioteca de código abierto desarrollada por Google para aprendizaje automático. Esta librería de computación matemática ejecuta de forma rápida y eficiente gráficos de flujo, los cuales están formados por operaciones matemáticas representadas sobre nudos y cuyas entradas y salidas son vectores multidimensionales de datos (tensores).

PyTorch:

Es una biblioteca de código abierto desarrollada por FAIR (Facebook’s AI Research) que se puede considerar una plataforma donde uno puede trabajar con tensores para computar modelos de deep learning con aceleración de GPU.

Es básicamente la competencia de TensorFlow, y actualmente lo supera en el ámbito de data science e investigación.

Google Colab:

Colaboratory de Google es un entorno gratuito que no requiere configuración y se ejecuta completamente en la nube. Este entorno interactivo permite escribir y ejecutar código desde el navegador, pudiendo combinarlo con texto, imágenes, etc.

La ventaja de trabajar en este entorno es la posibilidad de ejecutar código en GPUs potentes que ofrece Google. Esto se traduce en un ahorro más que considerable de tiempo (en Deep Learning se trabaja con grandes matrices que requieren de potentes GPUs para procesarlas rápido, por lo que si no se dispone de dichas GPUs, Google Colab es una buena alternativa). [5 p.23]