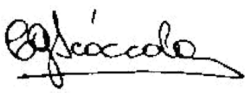
Universidad Nacional de La Plata
Facultad de Ciencias Astronómicas y Geofísicas

Tesis para obtener el grado académico de
Licenciada en Astronomía

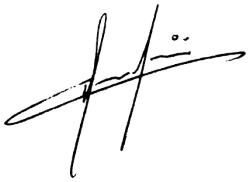
Wiener Filter para mapas del Fondo Cósmico de Radiación utilizando redes neuronales

María Belén Costanza


Claudia Scóccola
Directora


María Pía Piccirilli
Jurado

Directora: Dra. Claudia G. Scóccola
Co-director: Prof. Dr. Matías Zaldarriaga


Mauro Mariani
Jurado

La Plata, Argentina
- Marzo de 2022 -

A mis abuelas,
Valentina y Kelly

Resumen

El Fondo Cósmico de Radiación es la radiación más antigua del Universo, propagándose a través del Universo desde la recombinación, 380.000 años después del Big Bang. Por ende, constituye una de las herramientas más importantes para conocer sobre el origen y evolución del Universo. El estudio de las anisotropías de la radiación, correspondientes a fluctuaciones en la temperatura y polarización, permite determinar con gran precisión los parámetros cosmológicos fundamentales del modelo estándar ΛCDM. Ese nivel de precisión requiere que el análisis de los datos sea estadísticamente óptimo para la explotación de los mismos.

En este trabajo estudiamos un mecanismo para el procesamiento de señales llamado filtro de Wiener, a partir del cual se reduce el ruido presente en la señal recibida e intenta reconstruir la señal esperada sin ruido. Estudiamos su implementación actual a través de un método iterativo que utiliza el gradiente conjugado a mapas de temperatura del fondo cósmico de radiación pero que resulta ser lento y costoso computacionalmente. Siendo el filtro de Wiener un procedimiento importante para la optimización del análisis de los datos, estudiamos la aplicación de técnicas de aprendizaje automático, específicamente de redes neuronales, para la realización de dicho filtro.

El objetivo de esta Tesis es contribuir al entendimiento y desarrollo de redes neuronales para la resolución de problemas que hoy en día afronta el análisis de datos cosmológicos.

Abstract

The Cosmic Microwave background is the oldest radiation in the Universe, propagating across the Universe since recombination, 380,000 years after the Big Bang. It is one of the most important tool to learn about the origin and evolution of the Universe. The study of the anisotropies in the radiation, corresponding to fluctuations in its temperature and polarization, allows to determine the fundamental parameters of cosmology of the ΛCDM standard model with high accuracy. This level of precision requires the statistical optimization of the data analysis methods, in order to fully exploit the data.

In this work, we study the Wiener filter mechanism which reduces the noise present in the received signal and attempt to rebuild the expected signal without noise. We study the current implementation to Cosmic Microwave Background maps with an iterative method that uses conjugate gradient, but that is slow and has an enormous computational cost. Being the Wiener filter an important procedure for the optimization of the data analysis, we study the application of machine learning techniques, specifically neural networks, for the implementation of that filter.

The aim of this Thesis is to contribute to the knowledge and development of neural networks for solving problems that cosmological data analysis faces today.

Agradecimientos

Considero importante tomarse el tiempo para agradecer a las personas que fueron parte de un proceso que conllevó tantos años y esfuerzo. Por eso, aquí van algunos agradecimientos:

A Claudia Scóccola y Matías Zaldarriaga por el acompañamiento durante el proceso de elaboración de esta Tesis, ya sea con reuniones o respondiendo dudas. Por sus ideas, paciencia y enseñanzas; fue un honor aprender de ustedes.

A Pía y Mauro por sus correcciones para mejorar la tesis.

A la Universidad Nacional de La Plata, y especialmente a la Facultad de Ciencias Astronómicas y Geofísicas, la casa de estudios que elegí para formarme como profesional. Agradezco haber tenido la posibilidad de estudiar lo que me gusta en una Universidad pública y gratuita. También agradezco a los profesores que me formaron a lo largo de la carrera.

A el Instituto de Estudios Avanzados de la Universidad de Princeton por la utilización de sus computadoras para realizar los cálculos presentados en esta Tesis.

A mis padres Cecilia y Gabriel. Desde que les dije que quería estudiar astronomía a 70km de casa y aunque eso implicara un gran viaje por día, me apoyaron y confiaron en mí desde el día cero hasta el final, ya sea desde el soporte económico hasta el gran apoyo emocional en los distintos tramos de la carrera. Gracias a ellos entendí que la familia es un refugio y una casa que acompaña.

A mis hermanos, Francisco y Lucas, mis personas favoritas en el mundo.

A mis abuelas, a quienes dedico esta Tesis, Valentina y Kelly que, aunque con pesar una de ellas no pueda estar presente en este momento, me han acompañado en cada momento de la vida y pasé largas horas estudiando en sus casas.

A mi mejor amiga, Barbi, con quien desde los 8 años compartimos la vida, festejando los logros y sosteniéndonos mutuamente cuando es necesario.

A mis amigas del secundario: Tefi, Maro, Sofi P, Sofi S, Luli, Anto, Flor, Ailu, Romi y Vico, que desde que les conté que quería estudiar Astronomía me alentaron siempre a más y apoyaron en cada paso que daba. A mis amigos Ema y Mati, quiénes me han escuchado y alentado en este proceso.

A mis amigos y compañeros de la facultad, porque estudiar es hermoso pero mejor si se hace en buena compañía. Por las largas horas de estudio en la biblioteca tomando mate, compartiendo congresos y soñando con ser investigadores.

Índice general

Acrónimos

Lista de acrónimos utilizados en esta tesis:

- FCR: Fondo Cósmico de Radiación (*Cosmic Microwave Background*).
- ML: Aprendizaje automático (*machine learning*).
- GRF: Campo gaussiano aleatorio (*gaussian random field*).
- CNN: Redes neuronales convolucionales (*convolutional neural network*).
- WF: Filtro de Wiener (*Wiener Filter*).
- CG: Gradiente conjugado (*conjugate gradient*).

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La presente Tesis se enmarca dentro de un proyecto de investigación que apunta a estudiar la polarización del Fondo Cósmico de Radiación (FCR), con el objetivo general de medir y caracterizar los modos B primordiales cuyo patrón de polarización constituiría la evidencia observacional indirecta más fuerte a favor de los modelos inflacionarios.

El objetivo general de esta Tesis consiste en utilizar técnicas de aprendizaje automático (*machine learning*, ML), en particular, redes neuronales, para la aplicación de Filtros de Wiener (*Wiener filter*, WF) a mapas del FCR. Dichos filtros tienen como propósito reducir el ruido presente en la señal de modo tal que la salida del filtro se aproxime lo más posible a la señal sin ruido.

Nos enfocaremos particularmente en los mapas de temperatura, pero en el futuro, la metodología puede generalizarse para aplicarla también a mapas de polarización del FCR.

1.1. Contexto

Luego del Big Bang, cuando la temperatura del Universo disminuyó al orden de $10^4 K$, ocurrió la recombinación a un corrimiento al rojo (*redshift*) de $z \simeq 1100$, donde los electrones libres y los protones se combinaron para formar hidrógeno neutro. Los fotones, que antes estaban fuertemente acoplados a los electrones por la dispersión de Thomson, se desacoplaron del plasma primordial y viajaron libremente a través del espacio. El Universo se hace transparente a la radiación, y estos fotones liberados en la superficie de última dispersión componen el Fondo Cósmico de Radiación (FCR) que podemos detectar hoy (Dodelson, 2003).

De esta manera, el FCR constituye la radiación reliquia del big bang cuando el Universo tenía 380.000 años. La distribución espectral de energía de los fotones es la de un cuerpo negro con una temperatura de $T \sim 2.725 K$. Presenta anisotropías correspondientes a fluctuaciones en la temperatura del orden de 10^{-5} debidas a pequeñas perturbaciones en el plasma cósmico, las cuales fueron descubiertas por el satélite COBE (*Cosmic Brackground Explorer*)[(i)] al hacer mediciones en distintas escalas angulares. También presenta anisotropías en la polarización, de intensidad aún más débil.

Por lo tanto, el estudio de las anisotropías en la temperatura y polarización del FCR es una de las herramientas más importantes para conocer sobre el origen y la evolución del Universo.

El FCR, desde su descubrimiento en el año 1965 por Penzias y Wilson (Penzias & Wilson, 1965), ha sido estudiado en gran detalle por experimentos en Tierra y por satélites, tales como

[(i)] https://science.nasa.gov/missions/cobe/

COBE, WMAP (*Wilkinson Microwave Anisotropy Probe*)[ii] y Planck[iii], cuyos resultados han sido los mapas de temperatura y polarización en todo el cielo. La figura 1.1 es un mapa de anisotropías del FCR en todo el cielo observado por la colaboración Planck, los puntos azules corresponden a direcciones en el cielo donde la temperatura es 10^{-5} menor que la media, mientras que las regiones amarillas y naranajas corresponden a regiones calientes.

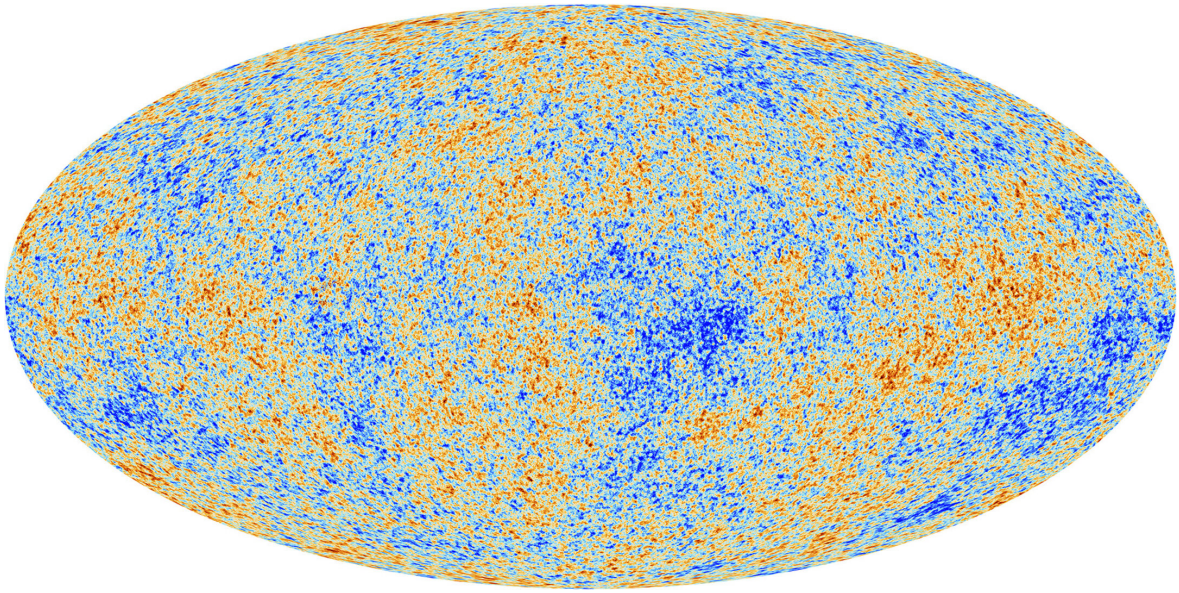


Figura 1.1. Mapa de fluctuaciones en la temperatura del Fondo Cósmico de Radiación observado por la colaboración Planck. Figura tomada de https://sci.esa.int/web/planck/.

Nuestro Universo se encuentra modelado por el modelo de Cosmología estándar ΛCDM, el cual dice que el Universo es euclídeo (plano, de curvatura nula) y está gobernado por la métrica de Friedmann Lemaitre Robertson Walker (métrica del espacio tiempo que describe un Universo en expansión, homogéneo e isótropo). Está dominado por la materia oscura fría y una constante cosmológica, cuyas perturbaciones iniciales fueron generadas por la inflación en el Universo temprano. Así, las perturbaciones primordiales en la densidad quedaron impresas en las anisotropías del FCR.

Las perturbaciones en la temperatura respecto de la media $\Delta T(\vec{n})$ van a depender de la dirección del momento lineal de los fotones, donde $\vec{n}$ denota las direcciones en el cielo. Al ser una función en la esfera, expandimos las perturbaciones en la temperatura en la base de armónicos esféricos:

$$\Theta(\vec{n}) \equiv \frac{\Delta T(\vec{n})}{T_0} = \sum_{lm} a_{lm} Y_{lm}(\vec{n}), \tag{1.1}$$

donde

$$a_{lm} = \int d\Omega Y^*_{lm}(\vec{n}) \Theta(\vec{n}). \tag{1.2}$$

Y_{lm} son los armónicos esféricos en la 2-esfera con $l = 0$, $l = 1$ y $l = 2$ correspondientes al monopolo, dipolo y cuadrupolo respectivamente, y $m = -l, ..., l$ el momento magnético dipolar. Los coeficientes del desarrollo a_{lm} son el observable que contiene la información del campo de temperatura cuya muestra para un determinado valor de l sigue una distribución

[ii] https://wmap.gsfc.nasa.gov/mission/
[iii] https://sci.esa.int/web/planck/

gaussiana con media cero y varianza dada por:

$$C_l^{TT} = \frac{1}{2l+1}\sum_m \langle a_{lm}^* a_{lm}\rangle \tag{1.3}$$

o

$$\langle a_{lm}^* a_{l'm'}\rangle = C_l^{TT}\delta_{ll'}\delta_{mm'}. \tag{1.4}$$

Los C_l constituyen el espectro angular de anisotropías del FCR, como puede verse en la figura 1.2. Su estudio requiere de métodos de análisis de datos sofisticados, ya que a través del mismo se puede hacer estimaciones de los parámetros cosmológicos fundamentales del modelo ΛCDM con gran precisión.

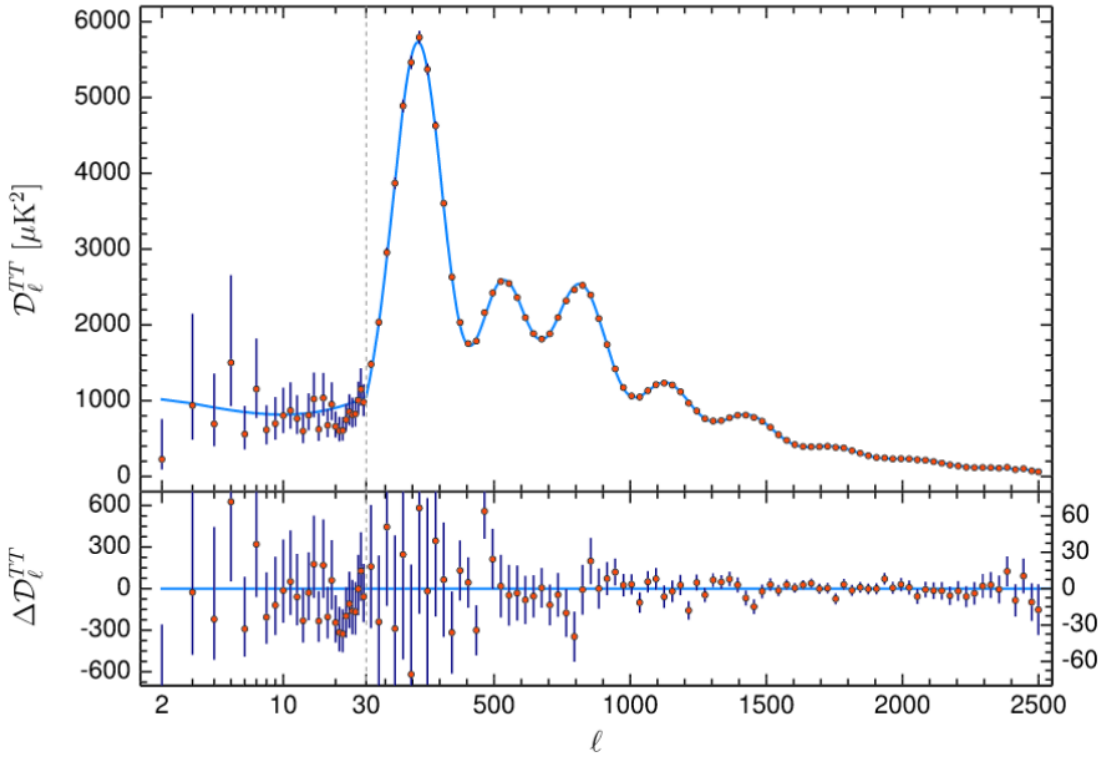


Figura 1.2. Espectro angular de anisotropías del Fondo Cósmico de radiación donde los puntos rojos son las mediciones hechas por el satélite Planck y la linea gruesa es la predicción teórica del modelo de cosmología con los parámetros cosmologicos fiduciales basados en las condiciones iniciales de inflación. En el eje y se encuentra graficado $D_l = \frac{l(l+1)C_l}{2\pi}$. Figura tomada de (Dodelson, 2003).

El filtrado de Wiener (WF) sobre mapas simulados o reales del FCR es un procedimiento básico para que el análisis de datos sea estadísticamente óptimo. Sin embargo, éste constituye un cuello de botella en términos de costos computacionales provocando que su realización sea difícil de implementar. Aplicar el WF a datos del FCR (Smith et al., 2007) requiere la inversión de matrices de señal y ruido, por lo que el método usual utilizado para tal fin es el método iterativo que utiliza el gradiente conjugado (*conjugate gradient*, CG) (Press et al., 2007).

Por lo tanto, se ha desarrollado una red neuronal llamada WienerNet (Münchmeyer & Smith, 2019) cuyo entrenamiento reproduce el WF con gran aproximación en mucho menos tiempo de cómputo. La misma constituye otro método diferente al CG para el filtrado de los mapas y puede ser de utilidad para la optimización del análisis de datos del FCR. Como se

verá en secciones siguientes, lo innovador de esta red neuronal es que su entrenamiento no requiere de mapas filtrados con WF, además de que el mapa de salida de la red es lineal con el mapa de entrada.

1.2. Objetivos

El objetivo de este trabajo de Tesis es incorporar el WF a nivel de mapas de temperatura para reducir el nivel de ruido a través de la implementación de redes neuronales. El objetivo general es aplicar específicamente la red neuronal WienerNet y estudiar si el entrenamiento de la misma predice los resultados esperados del WF, así como su eficiencia en términos del tiempo de cómputo, en contraposición con el método tradicional iterativo CG.

De esta manera, primero se estudiará los conceptos básicos de aprendizaje automático, luego cómo se construyen redes neuronales sencillas, la propagación de la información a través de las mismas y el mecanismo por el cual aprenden. Se espera contribuir así al entendimiento de técnicas de ML útiles para ser aplicadas en las próximas generaciones de instrumentos o distintos escenarios cosmológicos.

Luego se estudiarán estructuras más complejas de redes neuronales como las redes convolucionales *autoencoders*.

Se diseñarán algoritmos de redes neuronales convolucionales para la aplicación de filtros de Wiener a mapas pequeños de 28x28 con aproximación plana considerando ruido blanco y homogéneo. Luego se analizarán las predicciones para estos mapas junto con los resultados del método tradicional que utiliza el CG. Se estudiará, a su vez, teóricamente cómo se realiza el WF.

Se aumentará el tamaño de los mapas para analizar casos más realistas, hasta 512x512 píxeles y se añadirá una máscara donde el ruido es infinito. Esto requerirá el diseño de estructuras más complejas de la red neuronal.

Se estudiará como escalea con el tiempo la realización del WF con ambos métodos y se analizará la eficiencia de la red neuronal.

1.3. Metodología

La metodología de trabajo consistió en la generación de mapas simulados del FCR para distintos número de píxeles con las librerias CAMB y Healpy, la programación de redes neuronales con paquetes diseñados para tal fin como Tensorflow y Keras en lenguaje *python*, y el estudio del filtro de Wiener para su posterior comparación con la utilización del código público Nifty.

Los cálculos más demandantes para la generación de grandes conjuntos de datos y el entrenamiento de las redes neuronales fueron realizados con las supercomputadoras del Instituto de Estudios Avanzados de la Universidad de Princeton. También fue utilizado el cluster Helios, constituido por 64 nodos de 128GB de memoria RAM cada uno, y los nodos Apollo con 8 GPUs (*graphics processing unit*) NVIDIA A100, ambos del mismo Instituto.

La Tesis está organizada de la siguiente manera:

- En el capítulo 2 se presenta teóricamente qué es el filtro de Wiener y por qué su realización es óptima para el análisis de datos.

- En el capítulo 3 se introducen definiciones básicas sobre aprendizaje automático con el fin de presentar conceptos necesarios para la construcción de redes neuronales sencillas.
- En el capítulo 4 detallamos un tipo específico de red neuronal que será de utilidad en esta Tesis llamado "red neuronal convolucional". Luego se presentarán estructuras más complejas compuestas de *Autoencoders* y se explicará el funcionamiento de la red neuronal WienerNet, la cual luego será programada para nuestro problema en cuestión.
- En el capítulo 5 presentamos los resultados de la aplicación de la red neuronal WienerNet a mapas de diferentes tamaños para explorar los distintos grados de complejidad. Se muestra el análisis de datos sobre los mismos y un estudio comparativo de este modelo respecto del método tradicional para el filtrado de Wiener.
- En el capítulo 6 presentamos las conclusiones del trabajo realizado durante esta Tesis y las perspectivas de trabajo futuro que pueden plantearse dentro de este tema de investigación.

Capítulo 2

Filtros de Wiener

En el área de procesamiento de señales, el Filtro de Wiener (WF) es muy utilizado para reducir el ruido de una señal. Es un procedimiento que utiliza métodos estadísticos para reducir el ruido presente en la señal de entrada, de manera tal que la señal de salida del filtro se aproxime lo más posible a una señal deseada (sin ruido).

El formalismo de los filtros de Wiener es ampliamente utilizado en Cosmología. Se usa tanto en la reconstrucción del espectro de potencias de la materia, para estudiar la estructura a gran escala del Universo (Seljak, 1998), como en el espectro angular de potencias de las anisotropías del FCR (Elsner & Wandelt, 2012). Es particularmente útil cuando se cuenta con un conjuntos de datos incompletos, escasos y con ruido, como es el caso de los relevamientos de galaxias (por ejemplo, SDSS (Ahn et al., 2012), 2dFGRS (Colless et al., 2001)) y mapas del FCR (por ejemplo, los obtenidos con los satélites COBE (Smoot et al., 1992), WMAP (Bennett et al., 2003), y Planck (Planck Collaboration et al., 2016)). Desarrollar y entender este mecanismo de filtrado de las observaciones es importante ya que la precisión con la que se puedan testear modelos cosmológicos depende de la cantidad de información útil que podamos extraer de los mismos.

En esta Tesis nos interesa reconstruir la distribución de temperatura del FCR dado un conjunto de datos simulados con ruido homogéneo. Esto requiere el procesamiento de imágenes de mapas de temperatura para la reconstrucción de la señal subyacente.

El método de WF está basado en la minimización de la varianza entre el estimador óptimo y cualquier otra realización del campo subyacente que se quiera reconstruir. En el caso de que ese campo sea un campo gaussiano aleatorio (*Gaussian random field*; GRF), el WF adquiere su máximo potencial ya que el estimador de Wiener coincide con el estimador Bayesiano diseñado para maximizar la probabilidad a posteriori (Zaroubi et al., 1995). Por esta razón, es el método aplicado para la reconstrucción de la señal del FCR, ya que las desviaciones en las fluctuaciones de temperatura respecto de una distribución gaussiana son despreciables. También es utilizado para la reconstrucción de la estructura a gran escala del Universo ya que las perturbaciones primordiales en densidad predichas por el modelo cosmológico estándar tienen una distribución muy cercana a la gaussiana.

2.1. El problema de inversión

Consideremos un conjunto de puntos $\mathbf{d} = \{d_i\}$ $(i = 1, ..., M)$, que se corresponden con medidas sobre un campo discreto $\mathbf{s}$ que queremos estimar, $\mathbf{s} = \{s_\alpha\}$ $(\alpha = 1, ..., N)$. Estas cantidades se modelan matemáticamente como una combinación lineal del campo subyacente.

$$\mathbf{d} = \mathbf{R}\mathbf{s} + \epsilon, \tag{2.1}$$

donde $\mathbf{R}$ es una matriz de respuesta MxN del procedimiento de medición del campo y $\epsilon = \{\epsilon_i\}$ es la incerteza estadística asociada a los datos, la cual puede provenir de la respuesta del instrumento o incertezas intrínsecas del campo. Asumimos una modelización discreta del campo en alguna representación (espacio de Fourier o espacio de configuración).

Matemáticamente, para reconstuir el campo tendríamos que invertir directamente la ecuación 2.1:

$$\mathbf{s} = \mathbf{R}^{-1}\mathbf{d}. \tag{2.2}$$

Sin embargo, la matriz no es invertible ya que el número de datos independientes puede ser mucho menor a los grados de libertad del campo (N $\gg$ M), o por el contrario, tener más mediciones no independientes, y que el determinante de la matriz R sea nulo. De todos modos, aún en el caso de ser posible la inversión directa, ésta podría amplificar el ruido estadístico ϵ, conduciendo a una inversión inestable, incluso aunque la matriz sea cuadrada. En síntesis, el método de WF pretende obtener una estimación de $\mathbf{s}$ dado el conjunto de datos $\mathbf{d}$, minimizando la varianza de los residuos de la estimación.

2.2. Reconstrucción del Filtro de Wiener

El campo subyacente está parametrizado por la matriz de varianza-covarianza $\mathbf{S} = \langle ss^\dagger \rangle$, en la cual $s^\dagger$ denota el complejo conjugado del vector transpuesto de $\mathbf{s}$ y $\langle ... \rangle$ denota el promedio en el ensamble.

Definimos el estimador de WF del campo subyacente, s^{MV}, como la combinación lineal de los datos, $\mathbf{d}$, que minimiza la varianza de la discrepancia entre el estimador y las posibles realizaciones del campo. El WF es esencialmente un filtro que, en el espacio de configuración, se implementa como una convolución.

$$s^{MV} = \mathbf{F}\mathbf{d}, \tag{2.3}$$

donde $\mathbf{F}$ es una matriz NxM que minimiza la varianza del residuo $\mathbf{r}$ definido como:

$$\langle \mathbf{r}\mathbf{r}^\dagger \rangle = \langle (s - s^{MV})(s^\dagger - s^{MV\dagger}) \rangle. \tag{2.4}$$

Minimizamos la ecuación 2.4 respecto de $\mathbf{F}$ reemplazando 2.3 y obtenemos la matriz de WF:

$$\mathbf{F} = \langle \mathbf{s}\mathbf{d}^\dagger \rangle \langle \mathbf{d}\mathbf{d}^\dagger \rangle^{-1}. \tag{2.5}$$

Asumiendo que el término de incerteza ϵ es estadísticamente independiente del campo $\mathbf{s}$, $\langle \epsilon s^\dagger \rangle = 0$, las matrices de correlación que aparecen en la ecuación 2.5 utilizando la ecuación 2.1 son:

$$\langle \mathbf{s}\mathbf{d}^\dagger \rangle = \langle \mathbf{s}\mathbf{s}^\dagger \rangle \mathbf{R}^\dagger \equiv \mathbf{S}\mathbf{R}^\dagger \tag{2.6}$$

y

$$\mathbf{D} \equiv \langle \mathbf{d}\mathbf{d}^\dagger \rangle = \mathbf{R}\mathbf{S}\mathbf{R}^\dagger + \langle \epsilon\epsilon^\dagger \rangle. \tag{2.7}$$

Consideramos la contribución del ruido como $\epsilon = \mathbf{R}\sigma$, donde σ es el *shot noise* del campo subyacente que mediríamos si $\mathbf{R} = \mathbf{I}$, por lo que la matriz de correlación del ruido es:

$$\mathbf{N}_\epsilon \equiv \langle \epsilon\epsilon^\dagger \rangle = \mathbf{R}\langle \sigma\sigma^\dagger \rangle \mathbf{R}^\dagger \equiv \mathbf{R}\mathbf{N}_\sigma\mathbf{R}^\dagger. \tag{2.8}$$

Luego, la matriz de WF es:

$$\mathbf{F} = \mathbf{S}\mathbf{R}^\dagger(\mathbf{R}\mathbf{S}\mathbf{R}^\dagger + \mathbf{N}_\epsilon)^{-1} \tag{2.9}$$

o

$$\mathbf{F} = \mathbf{S}(\mathbf{S} + \mathbf{N}_\sigma)^{-1}\mathbf{R}^{-1}. \tag{2.10}$$

La ecuación 2.10 presenta explícitamente las operaciones fundamentales de WF: la inversión de la función de respuesta y la supresión del ruido dada por la razón $\frac{señal}{señal+ruido}$.

Reemplazamos la ecuación 2.10 en 2.4 y obtenemos la varianza del residuo:

$$\langle \mathbf{r}\mathbf{r}^\dagger \rangle = \mathbf{S}(\mathbf{S} + \mathbf{N}_\sigma)^{-1}\mathbf{N}_\sigma. \tag{2.11}$$

Para el caso que el campo sea GRF, el estimador de WF coincide con el estimador del campo que maximiza la probabilidad condicional $P(s|d)$, y por esa razón es óptimo.

Sea la función densidad de probabilidad del campo $\mathbf{s}$ una distribución gaussiana multivariable determinada por la matriz de covarianza $\mathbf{S}$:

$$P(\mathbf{s}) = \frac{1}{[(2\pi)^N det(\mathbf{S})]^{1/2}} exp(-\frac{1}{2}\mathbf{s}^\dagger \mathbf{S}^{-1}\mathbf{s}). \tag{2.12}$$

Si el ruido es un GRF independiente, la probabilidad conjunta de la señal y los datos es:

$$P(\mathbf{s}, \mathbf{d}) = P(\mathbf{s}, \epsilon) = P(\mathbf{s})P(\epsilon) \propto exp(-\frac{1}{2}(\mathbf{s}^\dagger \mathbf{S}^{-1}\mathbf{s} + \epsilon^\dagger \mathbf{N}^{-1}\epsilon)), \tag{2.13}$$

mientras que la probabilidad condicional de la señal, dados los datos, es:

$$P(\mathbf{s}|\mathbf{d}) = \frac{P(\mathbf{s}, \mathbf{d})}{P(\mathbf{d})} \propto P(\mathbf{s})P(\epsilon) \propto exp[-\frac{1}{2}(\mathbf{s}^\dagger \mathbf{S}^{-1}\mathbf{s} + (\mathbf{d} - \mathbf{R}\mathbf{s})^\dagger \mathbf{N}^{-1}(\mathbf{d} - \mathbf{R}\mathbf{s}))]. \tag{2.14}$$

Según el teorma de Bayes, la probabilidad a posteriori del modelo dados los datos se escribe como: $P(model|data) \propto P(data|model)P(model)$. El estimador del campo más probable es aquel que maximiza $P(model|data)$, definido como el estimador bayesiano. La densidad de probabilidad a posteriori en este caso es:

$$P(\mathbf{s}|\mathbf{d}) \propto P(\mathbf{s})P(\mathbf{d}|\mathbf{s}), \tag{2.15}$$

donde la ecuación 2.14 nos muestra que:

$$P(\mathbf{s}|\mathbf{d}) \propto exp[-\frac{1}{2}(\mathbf{s}^\dagger \mathbf{S}^{-1}\mathbf{s} + (\mathbf{d} - \mathbf{R}\mathbf{s})^\dagger \mathbf{N}^{-1}(\mathbf{d} - \mathbf{R}\mathbf{s}))]. \tag{2.16}$$

El estimador bayesiano, correspondiente a la configuración más probable del campo subyacente dados los datos, coincide con $\mathbf{s}^{MV}$, de modo que el estimador de WF es el estimador óptimo del campo subyacente.

Capítulo 3

Elementos de Aprendizaje Automático

El método empleado para realizar el WF es un método iterativo que resulta ser un cuello de botella para el análisis de datos cosmológicos. Por esta razón, en esta Tesis exploramos técnicas de aprendizaje automático útiles para reproducir el WF mediante redes neuronales.

Ya se ha estudiado en los últimos años la aplicación de redes neuronales en otros contextos cosmológicos, como por ejemplo, la reconstrucción del potencial de lente gravitacional (*lensing potential*) del FCR (Caldeira et al., 2019), la generación de mapas del FCR (Mishra et al., 2019), y la estimación de parámetros cosmológicos a través de la distribución de materia oscura (Ravanbakhsh et al., 2017).

En esta sección presentamos los conceptos básicos de aprendizaje automático (*machine learning*, ML), específicamente aprendizaje automático **supervisado** que son los que luego emplearemos a lo largo de la Tesis. En este caso, los sistemas de ML aprenden cómo combinar entradas para producir predicciones útiles sobre datos nunca antes vistos.

El aprendizaje automático consiste en un nuevo paradigma de programación diferente a la programación clásica. En este último, los humanos introducimos reglas a un programa, los datos son procesados acorde a esas reglas y obtenemos respuestas. Por el contrario, con ML los humanos introducimos datos así como las respuestas que esperamos de esos datos, y obtenemos las reglas. Esas reglas son aplicadas a nuevos datos nunca antes vistos para predecir respuestas, como se muestra en la figura 3.1 (Chollet, 2017).

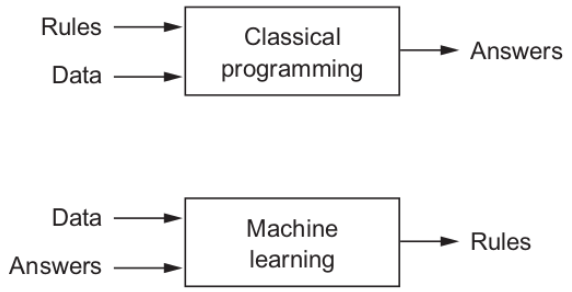


Figura 3.1. Aprendizaje automático: Un nuevo paradigma de programación. Figura tomada de Chollet (2017)

Algunas definiciones importantes son:

- La **etiqueta** es el valor que estamos prediciendo (por ejemplo, la variable y de una regresión lineal simple).

- Un **atributo** es una variable de entrada (la variable x en un modelo de regresión lineal simple).
- Un **ejemplo** es una instancia de datos en particular $\vec{x}$ que se dividen en dos categorías: ejemplos etiquetados y ejemplos sin etiquetas.

Un **ejemplo etiquetado** incluye tanto atributos como la etiqueta correspondiente (x,y). Los ejemplos etiquetados son los utilizados para entrenar un modelo de aprendizaje automático supervisado. Luego ese modelo se usa para predecir la etiqueta de ejemplos sin etiquetas.

Un modelo define la relación entre los atributos y la etiqueta. Contiene dos fases:

- **Entrenamiento**: mostrar ejemplos etiquetados al modelo y permitir que éste aprenda gradualmente las relaciones entre los atributos y la etiqueta.
- **Inferencia**: aplicar el modelo entrenado a ejemplos sin etiqueta. Es decir, utilizar el modelo para realizar predicciones útiles.

El método más sencillo es el **modelo de regresión lineal simple**, que consiste en encontrar la línea recta que mejor se adapta a un conjunto de datos. A partir de este modelo simple podemos entender conceptos elementales. La ecuación para un modelo de este tipo es:

$$y' = b + w_1 x_1, \tag{3.1}$$

donde y' es la etiqueta (resultado predicho), w_1 la ponderación/peso del atributo 1, x_1 es un atributo (una entrada conocida), y b es la ordenada al origen. Un modelo más sofisticado, que se base en tres atributos, usaría la siguiente ecuación:

$$y' = b + w_1 x_1 + w_2 x_2 + w_3 x_3. \tag{3.2}$$

Entrenar un modelo significa aprender (determinar) valores correctos para todos los pesos y ordenada al origen de los ejemplos etiquetados. En aprendizaje automático supervisado el algoritmo se construye al examinar varios ejemplos e intentar encontrar un modelo que minimice la pérdida (que definiremos a continuación). Por lo tanto, *aprender* significa encontrar el conjunto de pesos que permita que el modelo relacione correctamente los atributos con las etiquetas.

La **pérdida** es un número que indica que tan incorrecta fue la predicción del modelo en un solo ejemplo. Si la predicción del modelo es perfecta, la pérdida es cero, de lo contrario la pérdida es mayor. El objetivo de entrenar un modelo es encontrar un conjunto de pesos y ordenada al origen que, en promedio, tengan pérdidas bajas en todos los ejemplos.

Este es el objetivo de la función de pérdida, la cual toma las predicciones del modelo de ML y las etiquetas verdaderas (que es lo que deseamos que el modelo reproduzca) y calcula el apartamiento entre ambas. El uso de esta función nos ayuda a ajustar el valor de los pesos en la dirección que minimice la pérdida. Este ajuste es realizado por el **optimizador**, mecanismo por el cual el modelo se ajusta basado en los datos y la pérdida.

Un ejemplo de esto es la pérdida **error cuadrático medio (ECM)**, la cual es el promedio de la pérdida al cuadrado de cada ejemplo.

$$ECM = C = \frac{1}{N} \sum_{(x,y)\in D} (y - prediction(x))^2, \tag{3.3}$$

donde D es el conjunto de ejemplos etiquetados, que son los pares (x, y), N es la cantidad de ejemplos y *prediction(x)* es la salida del modelo.

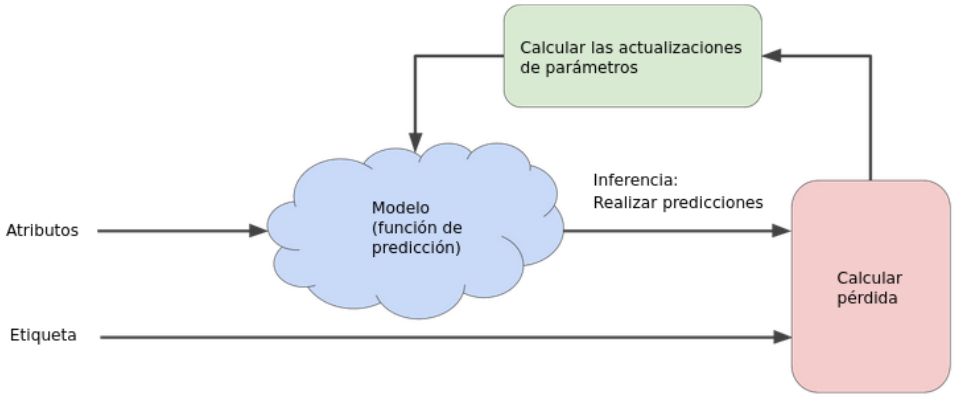


Figura 3.2. Un enfoque iterativo para entrenar un modelo.

La figura 3.2 contiene un diagrama del mecanismo de aprendizaje de ML. Se puede observar que el modelo toma uno o más atributos como entrada y devuelve una predicción y' como resultado. Por ejemplo, en un modelo de regresión simple, con un solo atributo, debemos tomar valores iniciales para b y w_1 elegidos al azar. La parte "Calcular pérdida" del diagrama es la función de pérdida que usará el modelo. La función de pérdida incorpora dos valores de entrada, la predicción y' para los atributos x y la etiqueta correcta y correspondiente a los atributos x. Finalmente llegamos a la parte de "Actualizar parámetros" del diagrama, donde el sistema de aprendizaje automático examina el valor de la función de pérdida y genera valores nuevos para b y w_1.

El aprendizaje continúa iterando hasta que el algoritmo descubre los parámetros del modelo con la pérdida más baja posible. En general, itera hasta que la pérdida general deja de cambiar o cambia muy lentamente. Cuando eso ocurre decimos que el modelo ha convergido. En la sección de "actualizar parámetros" de la figura 3.2 es donde entra el juego el optimizador elegido. Un optimizador básico es el mecanismo iterativo llamado "**descenso de gradientes**".

En los problemas de regresión, la pérdida EMC con respecto a la ponderación w_1 es una función convexa, por lo tanto posee un solo mínimo, donde la pendiente es nula. Ese mínimo es donde converge la función de pérdida.

La primera etapa en el descenso de gradientes es elegir un valor inicial para el peso w_1, el cual será al azar. Luego, el algoritmo calcula el gradiente de la curva de pérdida en el punto de partida, que para el caso de un solo peso es equivalente a la derivada.

El algoritmo toma un paso en la dirección del gradiente negativo para reducir la pérdida lo más rápido posible. Para determinar el siguiente punto a lo largo de la curva de la función de pérdida, el algoritmo agrega una fracción de la magnitud del gradiente al punto de partida η. Se repite este proceso y se acerca cada vez más al mínimo, como puede notarse en la figura 3.3.

Los algoritmos de descenso de gradientes multiplican el gradiente por un escalar, conocido como **tasa de aprendizaje** (*learning rate*), para determinar el siguiente punto.

Un hiperparámetro es todo aquello que podemos ajustar, sintonizar o modificar para alterar el desempeño del modelo. Un ejemplo de ellos es la tasa de aprendizaje, la cual debe ser elegida de modo tal que no sea ni muy pequeña (el aprendizaje llevaría mucho tiempo) ni muy grande (el siguiente punto rebotará del otro lado del mínimo y puede diverger). En la figura 3.4 puede observarse una correcta elección de la tasa de aprendizaje para el descenso de gradientes.

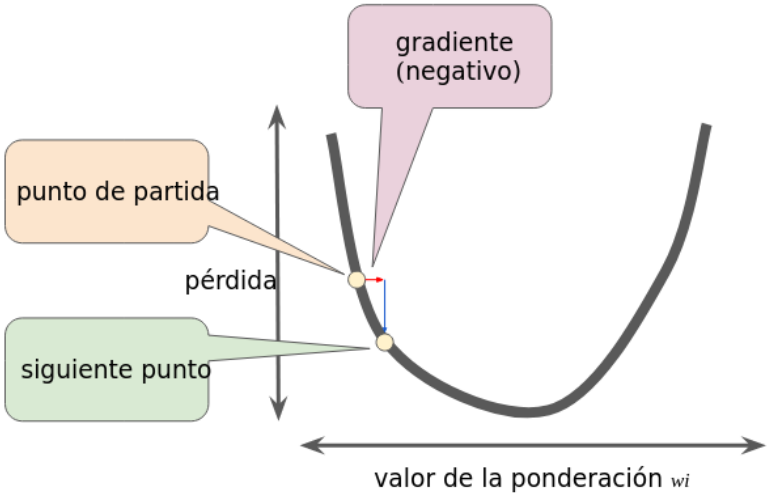


Figura 3.3. Un paso del gradiente nos mueve al siguiente punto en la curva de pérdida.

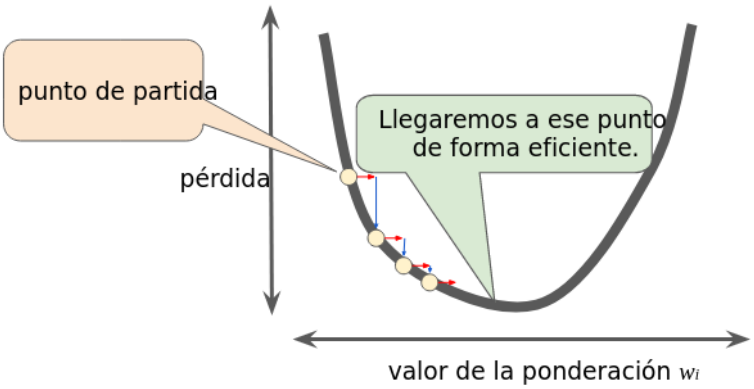


Figura 3.4. La tasa de aprendizaje es la correcta.

Los pesos se actualizan:

$$\begin{aligned} w_i' &= w_i - \eta \frac{\partial C}{\partial w_i}, \\ b_l' &= b_l - \eta \frac{\partial C}{\partial b_l}. \end{aligned} \tag{3.4}$$

En el descenso de gradientes un **lote** es la cantidad total de ejemplos que se usan para calcular el gradiente en una sola iteración, donde hasta ahora supusimos que un lote era el conjunto de datos completo. Sin embargo, los conjuntos de datos pueden tener miles de millones, o incluso cientos de millones, de ejemplos. En consecuencia, un lote puede ser enorme lo que puede causar que incluso una sola iteración tome un tiempo muy prolongado para calcularse.

Como se ha indicado, la función de pérdida es la suma de las funciones de pérdida en cada ejemplo:

$$C = \frac{1}{N} \sum_x C_x. \tag{3.5}$$

Supongamos un caso de la función de pérdida tridimensional. Para computar el gradiente de

la pérdida, realizamos el gradiente en cada uno de los ejemplos y promediamos:

$$\nabla C = \frac{1}{N} \sum_x \nabla C_x. \tag{3.6}$$

El **descenso de gradientes estocástico** es un método para obtener el gradiente en mucho menos tiempo de cómputo. La idea es elegir ejemplos al azar de nuestro conjunto de datos **minilote** (*batch size*) para estimar un promedio grande de otro mucho más pequeño. El término estocástico indica que los ejemplos que componen cada minilote son elegidos al azar.

Supongamos un minilote compuesto de m ejemplos al azar $X_1, X_2, ..., X_m$. Se aproxima el gradiente de la función de pérdida por el gradiente en el minilote lo que ayuda a que el entrenamiento requiera menos tiempo de cómputo

$$\frac{\sum_{j=1}^{m} \nabla C_{x_j}}{m} \approx \frac{\sum_x \nabla C_x}{N} = \nabla C. \tag{3.7}$$

Dado un minilote al azar, se actualiza los pesos:

$$\begin{aligned} w_i' &= w_i - \frac{\eta}{m} \sum_j \frac{\partial C_{x_j}}{\partial w_i}, \\ b_l' &= b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{x_j}}{\partial b_l}, \end{aligned} \tag{3.8}$$

donde las sumas son realizadas sobre los ejemplos aleatorios que se encuentran en el minilote.

Luego se elige otro conjunto de ejemplos aleatorios que conformen el minilote y se entrena con ellos. Finalmente, cuando el algoritmo ya examinó todos los datos de entrenamiento del conjunto de datos completo (compuesto por minilotes) decimos que se cumple una **época**. El número de épocas del modelo es otro de los hiperparámetros a ajustar del modelo.

La **generalización** hace referencia a la capacidad del modelo para adaptarse de manera adecuada a datos nuevos nunca antes vistos. Sin embargo, un modelo puede sobreajustarse a los datos de entrenamiento y no ajustarse a los datos nuevos, es decir, el modelo obtiene una pérdida baja durante el entrenamiento pero no se desempeña bien al predecir datos nuevos.

Por lo tanto, es conveniente dividir el conjunto de datos en dos subconjuntos: **conjunto de entrenamiento** y **conjunto de prueba**. Así un buen rendimiento en el conjunto de prueba es un indicador útil de un buen rendimiento en los datos nuevos.

Se puede reducir aún más la posibilidad de sobreajuste al particionar el conjunto de datos en 3 subconjuntos. Se añade el **conjunto de validación**, con el cual se evalúa los resultados del conjunto de entrenamiento en cada época. Finalizado el entrenamiento, se usa el conjunto de prueba para verificar la evaluación después de que el modelo haya pasado el conjunto de validación.

En este flujo de trabajo se selecciona el modelo que mejor se desempeñe con el conjunto de validación y se verifica el modelo con respecto al conjunto de prueba.

3.1. Redes Neuronales

Hasta ahora comentamos sobre problemas lineales donde el modelo puede representarse mediante una regresión lineal de atributos. Sin embargo, los modelos de aprendizaje automático pueden componerse de estructuras más complejas y combinaciones no lineales de los atributos, dependiendo del problema a tratar. Un modelo lineal es una combinación lineal de las entradas, como se ilustra en la figura 3.5. Cada círculo azul representa un atributo

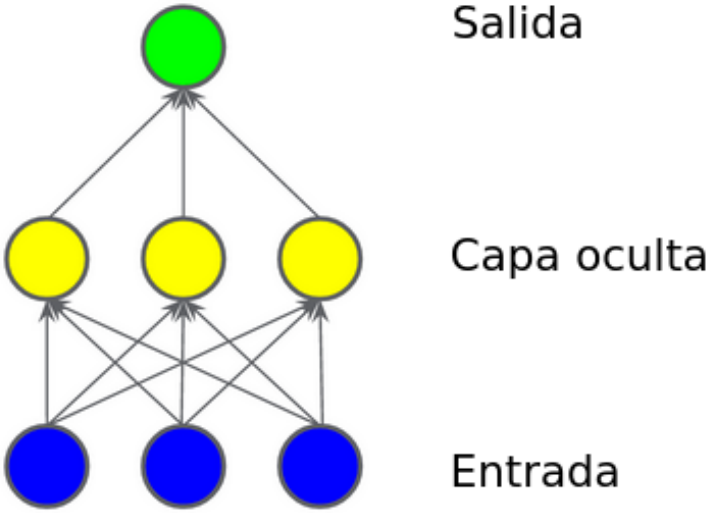


Figura 3.5. Grafo de un modelo de dos capas.

de entrada. Cada nodo amarillo (también llamado **neurona**) en la capa oculta es una suma ponderada de los valores de los nodos de entrada azul. El resultado es una suma ponderada de los nodos amarillos.

Para modelar un problema no lineal, se introduce directamente una no-linealidad, aplicando a cada nodo de la capa oculta una función no lineal o **función de activación**. En

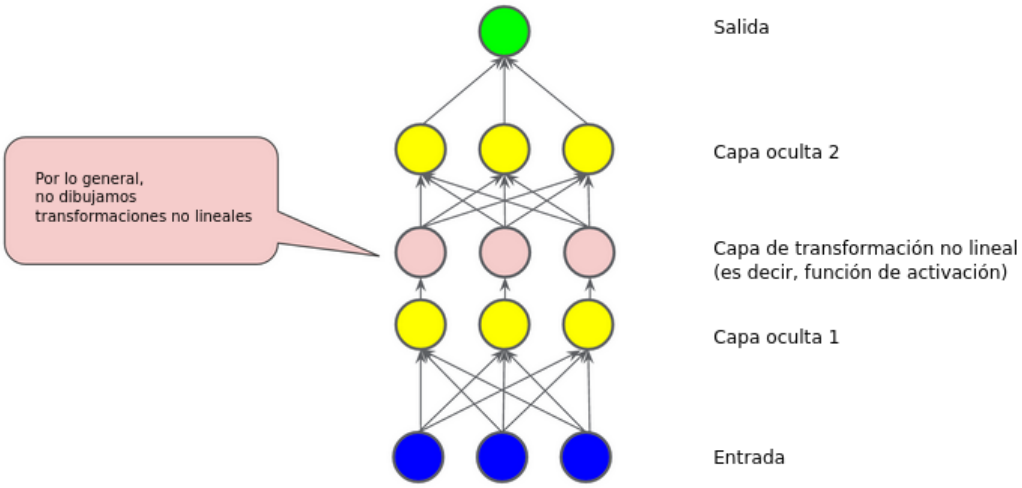


Figura 3.6. Modelo de tres capas con función de activación.

la figura 3.6, el valor de cada nodo en la capa oculta 1 se transforma mediante una función no-lineal antes de llegar a las sumas ponderadas de la siguiente capa.

Un ejemplo de red neuronal simple es el **perceptrón multicapa** (*multilayer perceptron*) formado por múltiples capas. Veremos un esquema de dos capas consistente de una capa oculta y otra de salida (Kubat, 2015), como se ilustra en la figura 3.7.

Las capas adyacentes están conectadas mediante enlaces neurona a neurona asociadas con un peso. El peso del enlace desde la j-ésima neurona oculta con la i-ésima neurona de salida

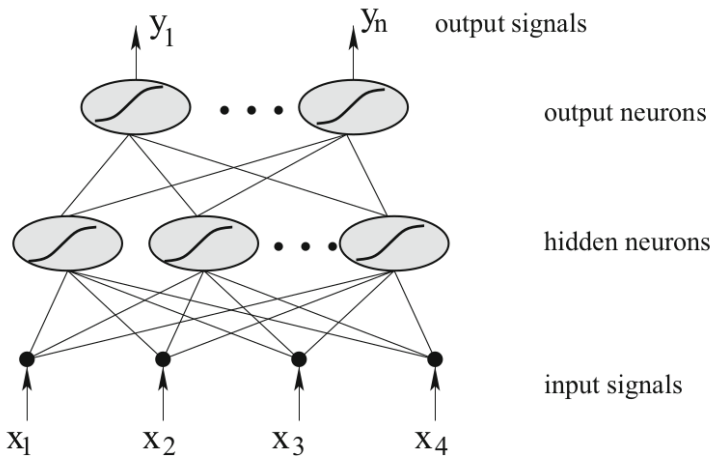


Figura 3.7. Red neuronal con dos capas conectadas. Figura tomada de Kubat (2015)

se denota $w_{ji}^{(1)}$ y el peso del enlace desde el k-ésimo atributo con la j-ésima neurona oculta se denota $w_{kj}^{(2)}$.

Cuando presentamos a la red neuronal un ejemplo $\vec{x} = (x_1, ..., x_n)$, sus atributos pasan a través de los enlaces a las neuronas. Los valores de x_k son multiplicados por los pesos asociados a los enlaces y la j-ésima neurona oculta recibe como entrada la suma ponderada:

$$\sum_k w_{kj}^{(2)} x_k. \tag{3.9}$$

Se evalúa esta suma en una función de activación elegida para aplicar la no-linealidad a la red neuronal:

$$f(\sum_k w_{kj}^{(2)} x_k). \tag{3.10}$$

Finalmente, la i-ésima neurona de salida recibe la suma ponderada de los valores provenientes de las neuronas en la capa oculta y nuevamente es evaluada en la función de activación. Así es como se obtiene el i-ésimo valor de salida de la red neuronal:

$$y_i = f(\sum_j w_{ji}^{(1)} f(\sum_k w_{kj}^{(2)} x_k)). \tag{3.11}$$

Este proceso de propagación de los valores de los atributos desde la entrada de la red a la salida se denomina **propagación hacia adelante** (*forward propagation*). Este tipo de redes neuronales pueden ser utilizadas para problemas de clasificación de imágenes, donde a cada neurona de salida se le asigna una clase y el valor devuelto por la i-ésima neurona de salida se interpreta como la cantidad de evidencia en apoyo a la i-ésima clase.

En el perceptrón multicapa los parámetros que afectan el desempeño de la red neuronal son el conjunto de pesos $w_{ji}^{(1)}$ y $w_{kj}^{(2)}$. Como se ha detallado en esta sección, el objetivo del entrenamiento es hallar los pesos que optimicen el desempeño de la red neuronal y la actualización de los pesos se ejecuta a través del optimizador.

Capítulo 4

Red neuronal WienerNet

En el capítulo 3 se han presentado elementos fundamentales del aprendizaje automático y funcionamiento de redes neuronales. En este capítulo nos centraremos en el estudio de la red neuronal que utilizamos para implementar el WF, llamada WienerNet.

Como se ha detallado en el capítulo anterior, el procedimiento de aprendizaje de una red neuronal consiste en enviarle ejemplos cuya solución se conoce para que realice el entrenamiento, y luego se espera que la red neuronal haga predicciones útiles sobre datos nunca antes vistos. Por lo tanto, siguiendo esta lógica, esperaríamos que una red neuronal que simule el WF reciba como ejemplos mapas de señal con ruido cuya etiqueta sea el mapa filtrado con WF, y que los utilice para aprender los pesos que optimicen el desempeño del WF.

Sin embargo, el enfoque de la arquitectura WienerNet que estudiamos aquí es diferente. Realizar el WF a través del método iterativo CG tiene un gran costo computacional, por lo que el objetivo de esta red es evitar enviar mapas filtrados para minimizar la pérdida. En su lugar se define una nueva función de pérdida, que presentaremos en la siguiente sección, que no requiere el uso de mapas filtrados, y cuya minimización nos dá la solución exacta de WF. De esta manera, se evita realizar el WF con el método tradicional a los mapas de entrenamiento, y sólo es utilizado para comparar el desempeño de la red neuronal.

4.1. Función de pérdida

Sea s la señal verdadera del FCR, que puede ser un mapa de anisotropías en la temperatura del FCR, y sea $d = s + n$ los datos contaminados con ruido que son enviados a la red neuronal como entradas, donde n es el ruido añadido a la señal. Notar que en este caso estamos asumiendo que la matriz de respuesta es la identidad $\mathbf{R} = I$.

Sea y la salida de la red neuronal (mapa filtrado). La aplicación de un filtro es matemáticamente una convolución, por lo que la red neuronal debe ser lineal en los datos d, para alguna matriz M que es aprendida durante el entrenamiento:

$$y = Md. \tag{4.1}$$

La red neuronal debe aprender a realizar el WF, por lo que luego del entrenamiento, la matriz M se espera que sea la matriz de WF en la ecuación 2.10, de forma tal que la salida de la red neuronal sea:

$$y_{WF} = S(S + N)^{-1}d, \tag{4.2}$$

donde S, N son las matrices de varianza-covarianza de la señal y el ruido respectivamente.

Durante el entrenamiento se actualizan los pesos de la red a través de la minimización del promedio de la función de pérdida $\langle J(s, d, y)\rangle$, donde el valor de expectación es sobre

realizaciones (s, d) en el conjunto de datos de entrenamiento. La función de pérdida que utilizamos se minimiza cuando $M = S(S + N)^{-1}$, y tiene la forma:

$$J(s, y) = \frac{1}{2}(y - s)^T A(y - s), \tag{4.3}$$

donde A es una matriz arbitraria definida positiva. Entrenar con esta función de pérdida implica minimizar la diferencia entre la salida de la red y y la señal verdadera s del FCR.

En lo que sigue, demostramos que, efectivamente, $\langle J \rangle$ es minimizada cuando $M = S(S+N)^{-1}$, primero reemplazando $y = Md = M(s + n)$:

$$\begin{aligned} \langle J \rangle &= \langle \frac{1}{2}(y - s)^T A(y - s) \rangle \\ &= \langle \frac{1}{2}((M - 1)s)^T A((M - 1)s) + \frac{1}{2}(Mn)^T A(Mn) \rangle \\ &= \frac{1}{2} Tr((M - 1)^T A(M - 1)S) + \frac{1}{2} Tr(M^T AMN). \end{aligned} \tag{4.4}$$

Derivando respecto de M:

$$\begin{aligned} \frac{\partial \langle J \rangle}{\partial M} &= A(M - 1)S + AMN \\ &= AM(S + N) - AS. \end{aligned} \tag{4.5}$$

Igualamos a cero y encontramos $M = S(S + N)^{-1}$.

Con este resultado se puede concluir que, si la red neuronal es entrenada con esta función de pérdida, cuando converja las predicciones que haga sobre datos nuevos será igual que aplicar el WF.

4.2. Red neuronal convolucional

WienerNet es una **red neuronal convolucional** (*convolutional neural network*, CNN) y en aprendizaje automático profundo (*deep learning*) estas redes son utilizadas usualmente para la detección de objetos (clasificar varios objetos en una imagen), segmentación de imágenes (clasificar cada píxel de acuerdo a la clase de objeto a la cual pertenece), entre otras.

La arquitectura de dichas redes consiste de capas convolucionales. En la primera capa las neuronas no están conectadas a cada pixel en la imagen de entrada. Si así lo fuese habría una gran cantidad de pesos/parámetros a optimizar; por ejemplo, para una imagen de 100x100 píxeles y 1000 neuronas en la primera capa, sería un total de 10 millones de conexiones únicamente en la primera capa, lo cual implica un alto costo computacional para el entrenamiento. En su lugar, están conectadas a píxeles dentro de un **campo receptivo** (*receptive field*). En la figura 4.1 se muestra la estructura de capas de una red neuronal convolucional.

En las redes neuronales multicapa que vimos en el capítulo 3, las capas consisten de una línea de neuronas y las imágenes de entrada deben ser aplanadas a 1 dimensión. A diferencia de éstas, en las redes neuronales convolucionales cada capa está representada por 2 dimensiones.

Una neurona localizada en una fila i y columna j de una dada capa es conectada a neuronas en la capa anterior localizadas en filas desde i a $i + f_h - 1$ y columnas desde j a $j + f_w - 1$, donde f_h y f_w son el largo y ancho del campo receptivo. Para que la capa tenga el mismo tamaño que la capa anterior se deben agregar ceros alrededor de la capa anterior, denominado *zero padding*, como puede verse en la figura 4.2.

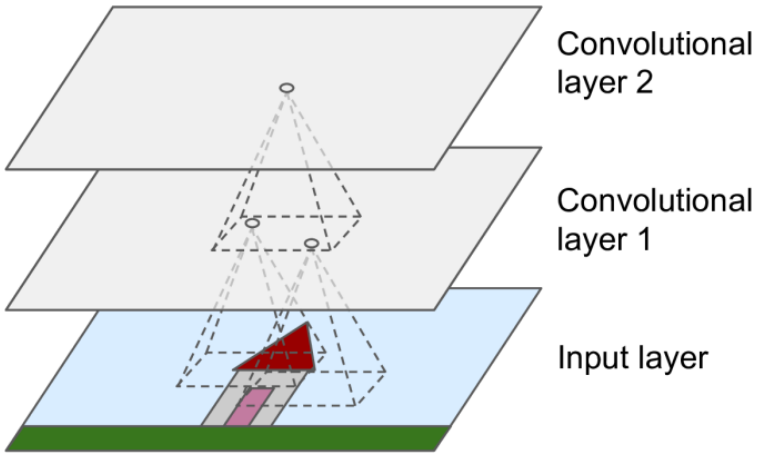


Figura 4.1. Capas de la red neuronal convolucional. Figura tomada de Géron (2019)

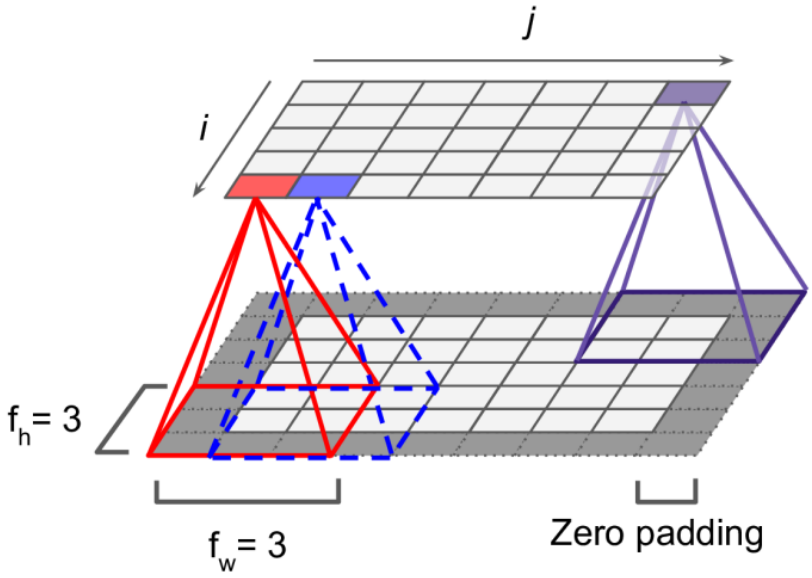


Figura 4.2. Conexiones entre capas. Figura tomada de Géron (2019)

Es posible conectar una capa con otra mucho más pequeña espaciando los campos receptivos donde el espaciamiento entre uno y otro se denomina ***stride***. Luego, como puede observarse en la figura 4.3, una neurona localizada en una fila i y columna j es conectada a neuronas localizadas en la capa anterior con filas desde $i \times s_h$ hasta $i \times s_h + f_h - 1$ y columnas desde $j \times s_w$ hasta $j \times s_w + f_w - 1$, donde s_h y s_w son los *strides* horizontal y vertical.

Los pesos de una neurona pueden representarse como una pequeña matriz del tamaño del campo receptivo llamado ***kernel***. Este kernel recorre la imagen de arriba-abajo, izquierda-derecha, según el *stride*. A medida que el kernel se desplaza se obtiene una nueva imagen filtrada por el mismo. Sin embargo, no tenemos un solo kernel sino un conjunto de ellos, denominados **filtros**, los cuales detectan características especifícas de la imagen y cada uno de ellos devuelve un **mapa de características** (*feature map*). Los filtros observan una región de la imagen de entrada del tamaño del campo receptivo.

El campo receptivo, también llamado contexto (*context*), entonces, es el área de la imagen de entrada que el filtro cubre.

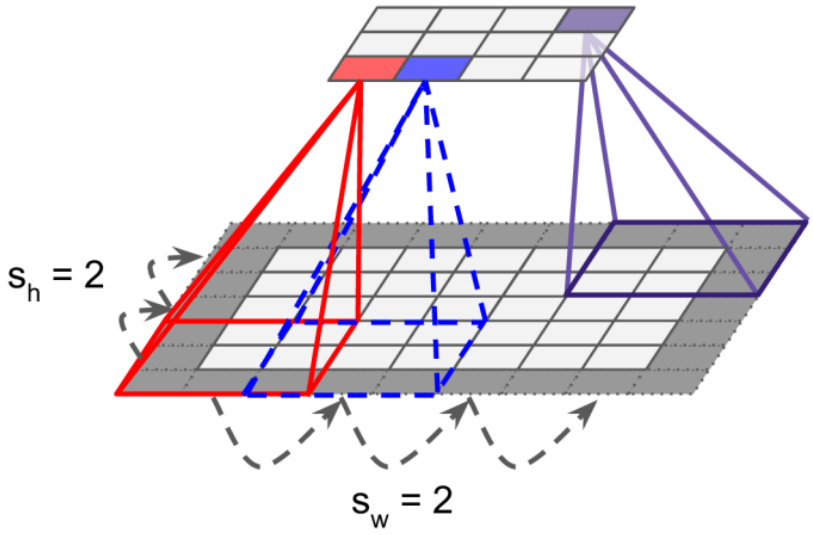


Figura 4.3. Reducción de la dimension con un stride de 2. Figura tomada de Géron (2019)

Al tener cada capa convolucional multiples filtros, y un mapa de características por filtro, es mas adecuado representar a las capas por objetos en 3 dimensiones. Una capa convolucional aplica simultánemente multiples filtros a las imagenes de entrada, haciendo posible la detección de multiples características en cualquier ubicación de la imagen, así si reconoce una característica en cierta ubicación, podrá reconocerla en cualquier otra. Por lo tanto, el número de pesos a entrenar se reduce significativamente respecto de una red neuronal multicapa.

Incluso las imágenes de entrada están compuestas de multiples subcapas: una por cada canal de color. Para imágenes a color (RGB) son típicamente tres: rojo, verde y azul. Para imágenes en la escala de grises sólo poseen un canal. Estos canales se pueden corresponder con las múltiples subfrecuencias que posee un mapa del FCR. En la figura 4.4 se ilustran capas convolucionales compuestas de mapas de características y una imagen de entrada con 3 canales.

Una neurona localizada en una fila i y columna j en una dada capa convolucional l es conectada a neuronas localizadas en la capa convolucional anterior $l-1$ con filas desde $i \times s_h$ hasta $i \times s_h + f_h - 1$ y columnas desde $j \times s_w$ hasta $j \times s_w + f_w - 1$, a través de todos los mapas de características (de la capa $l-1$). Notar, en la figura 4.4, que todas las neuronas localizadas en la misma fila i y columna j pero en diferentes mapas de características están conectadas a las mismas neuronas en la capa anterior.

En los paquetes para programación de redes neuronales, TENSORFLOW y KERAS, las imágenes de entrada que se envían a la red neuronal se expresan como tensores de 4 dimensiones, donde en la primer componente se especifíca el tamaño del minilote a utilizar, en la segunda y tercer componente el ancho y largo de la imagen y en la última componente se especifíca el número de canales: [batch size, largo, ancho, canales].

La salida de una neurona en una dada capa convolucional es la suma ponderada de las entradas:

$$z_{i,j,k} = b_k + \sum_{\mu=0}^{f_h-1} \sum_{\nu=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} . w_{\mu,\nu,k',k}, \tag{4.6}$$

donde

$$i' = i \times s_h + \mu, \tag{4.7}$$

$$j' = j \times s_w + \nu. \tag{4.8}$$

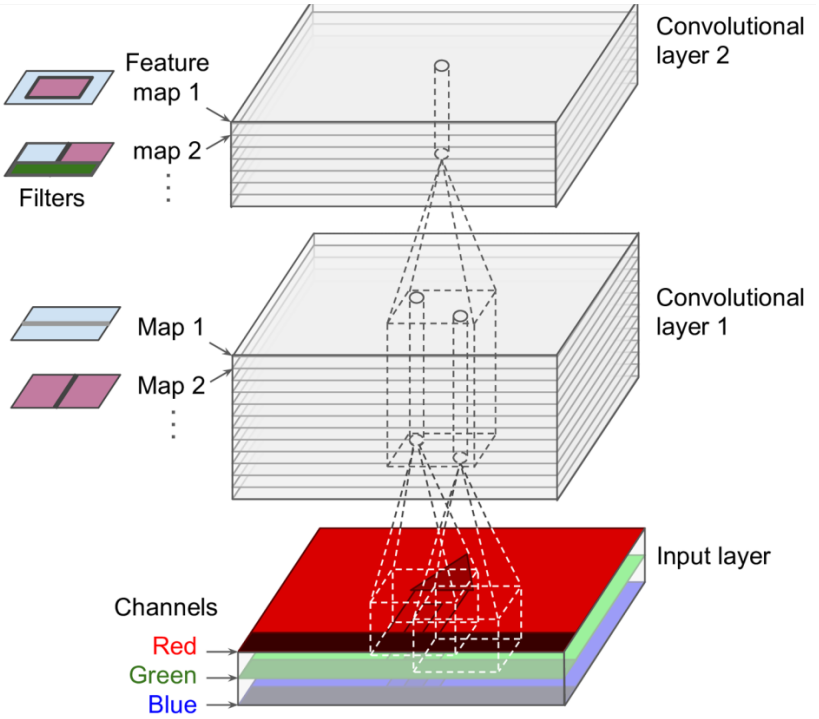


Figura 4.4. Capas convolucionales con múltiples mapas de características. Figura tomada de Géron (2019).

En esta ecuación:

- $z_{i,j,k}$ es la salida de la neurona localizada en la fila i, columna j en el mapa de características k de la capa convolucional l.
- s_h y s_w son los *strides* vertical y horizontal, f_h y f_w la altura y ancho del campo receptivo, y f_n el número de mapas de características en la capa anterior.
- $x_{i',j',k'}$ es la salida de la neurona localizada en la capa $l-1$, fila i', columna j', mapa de características k' (o canal k' si es la imagen de entrada).
- b_k es un término de ordenada al origen en el mapa de características k (en la capa l).
- $w_{\mu,\nu,k',k}$ es el peso que conecta una neurona en el mapa de características k de la capa l con una neurona localizada en la fila μ, ν (relativo al campo receptivo) y mapa de características k'.

Los pesos de una dada capa convolucional se presentan como un tensor de 4 dimensiones, cuyas componentes son $[f_h, f_w, f_{n'}, f_n]$.

4.3. Autoencoders

Un concepto importante en aprendizaje automático son las arquitecturas *encoder-decoder*. Para entender estos conceptos es preciso comprender lo que esperamos que la red neuronal aprenda. Como se mencionó en la sección previa, las redes neuronales convolucionales pueden tener múltiples objetivos. Por ejemplo:

- Clasificación y localización: clasificar el objeto con una etiqueta discreta y localizar donde está presente en la imagen con una caja alrededor del mismo. Éste es el caso de un objeto por imagen.
- Detección de objetos: se extiende la clasificación y localización a varios objetos en la imagen.
- Segmentación semántica (*Semantic segmentation*): cada píxel es clasificado en una clase correspondiente al objeto que pertenece. Diferentes objetos en una misma clase no se distinguen entre sí. La imagen de salida de la red es de la misma dimensión que la imagen de entrada, donde cada píxel es clasificado en una clase particular. Puede observarse un ejemplo de este caso en la figura 4.5.

Figura 4.5. Segmentación semántica. Diferentes objetos en una misma clase no se distinguen entre sí, por ejemplo, las clases "car" y "buildings". Figura tomada de Géron (2019)

Una red neuronal autoencoder se compone de dos partes: el ***encoder*** que aprende eficientes representaciones de los datos de entrada y el ***decoder*** que mapea esta representación en la salida de la red esperada.

Nos centramos en el caso de segmentación semántica donde no sólo es necesario saber "qué" objetos están presentes en la imagen sino que es igualmente importante saber en "dónde" se encuentran. Por tanto, los *encoders* sucesivamente reducen la dimensión de la imagen a través de capas convolucionales con *stride* > 2, aprendiendo eficientemente "qué" está presente en la imagen pero perdiendo la información de en "dónde" se encuentra. Los *decoders* recuperan esta información aumentando la dimensión de la imagen a través de capas convolucionales transpuestas, las cuales constituyen un proceso opuesto a las capas convolucionales antes vistas por el cual convierten imágenes de baja resolución en otras de mayor resolución a través del aprendizaje de parámetros.

4.4. Estructura de red neuronal tipo UNet

UNet es una red neuronal convolucional desarrollada para el análisis de imágenes biomédicas (Ronneberger et al., 2015). La arquitectura contiene dos partes simétricas de *encoders* y *decoders*. Los *encoders* se encargan de la contracción de la imagen para capturar el contexto en la misma a través de capas convolucionales y otras operaciones tales como *max pooling* no detalladas aquí. Por su parte, los *decoders* expanden la imagen permitiendo la localización de la información relevante a través de capas convolucionales transpuestas.

En la figura 4.6 cada bloque azul representa una capa con multiples filtros denotado arriba del bloque, y el conjunto de tres bloques es un *encoder*. A medida que la imagen se reduce a través de las capas del *encoder*, el campo receptivo recubre áreas de la imagen cada vez

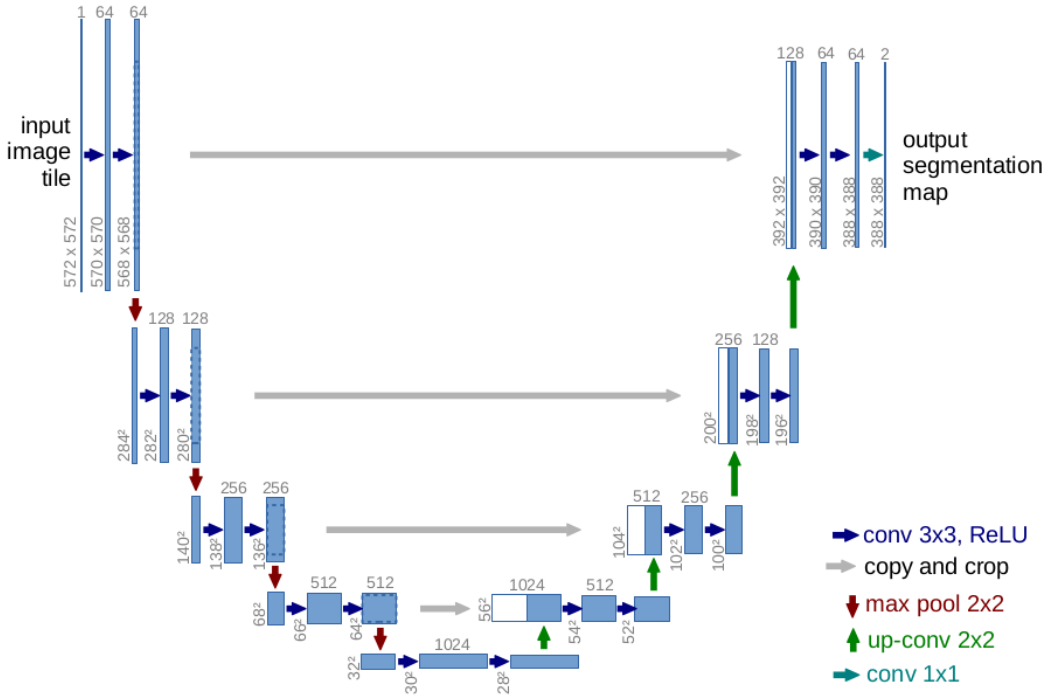


Figura 4.6. Arquitectura UNet para número de píxeles 572x572. Figura tomada de Ronneberger et al. (2015)

mayores, permitiendo a los filtros detectar los detalles. A su vez, el número de filtros aumenta a lo largo de los *encoders*, extrayendo características más complejas de la imagen.

Otra característica de este tipo de red simétrica son las ***skip connections*** las cuales transmiten información desde los *encoders* a los *decoders* a la misma altura en la arquitectura de la red. Esto se hace mediante la concatenación de los canales en las capas con la misma dimensión. La transmisión de mapas de características a la salida de los *decoders* permite recuperar detalles o características de la imagen en la predicción de la red. En la figura 4.6 se visualizan como líneas grises desde los *encoders* a los *decoders*.

La red neuronal WienerNet es una red neuronal basada en la estructura de UNet. La estructura es simétrica; se compone de *encoders* con una sola capa que sucesivamente reducen el tamaño de la imagen por un factor de 2 utilizando capas convolucionales con stride 2, y de *decoders*, también compuestos de una sola capa, que expanden la imagen por un factor de 2 y luego aplican una convolución de stride 1. En la figura 4.7 se muestra la estructura de la red neuronal WienerNet.

Una característica del filtro de Wiener, detallada en el capítulo 2, es que es lineal en el mapa de entrada, garantizando que un campo gaussiano sea transformado en otro campo gaussiano. Sin embargo, las redes neuronales convencionales, como se explica en el capítulo 3, tienen no linealidades a través de funciones de activación. Además la aplicación del filtro de Wiener depende de la máscara (zonas con ruido infinito, sin señal) y el ruido del experimento. La máscara rompe la homogeneidad estadística y no es invariante traslacional.

Por lo tanto, esta red particular se compone de dos caminos, uno lineal en el mapa de entrada, sin funciones de activación, y otro camino no lineal para la máscara con funciones de activación en cada capa, luego se multiplican las salidas de la capa no-lineal a la capa lineal.

La función de activación utilizada es la función lineal rectificada ReLU (*the Rectified Linear Unit function*), la cual es una función continua pero no derivable en 0, que transforma

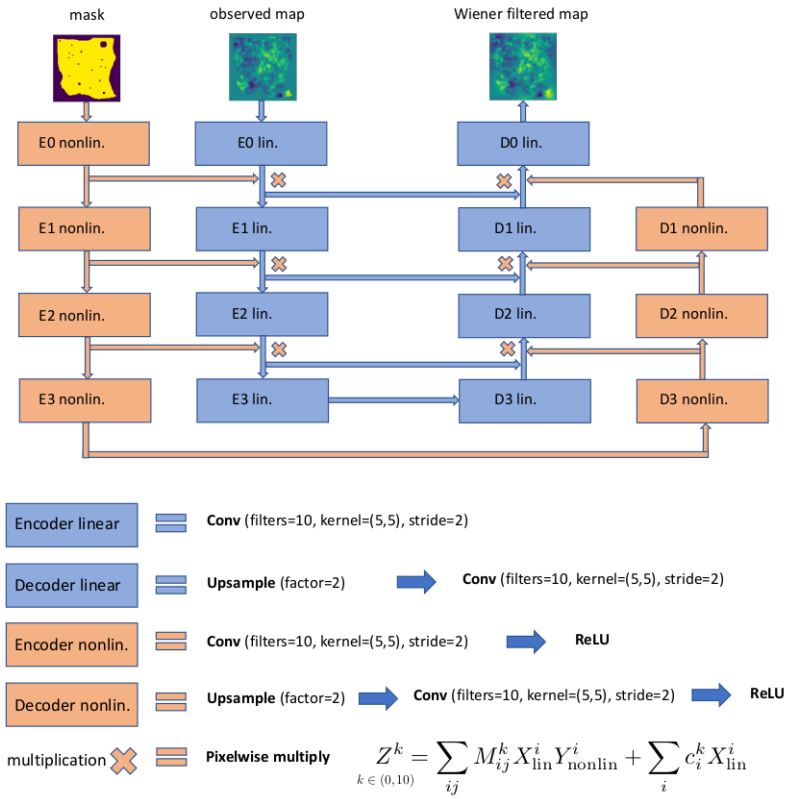


Figura 4.7. Arquitectura WienerNet. Figura tomada de Münchmeyer & Smith (2019)

los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran, como se ilustra en la figura 4.8. Esta función de activación es ampliamente usada en CNN ya que es simple computacionalmente, lo cual favorece el trabajo del optimizador, y prevalece los valores positivos en el flujo de las capas convolucionales;

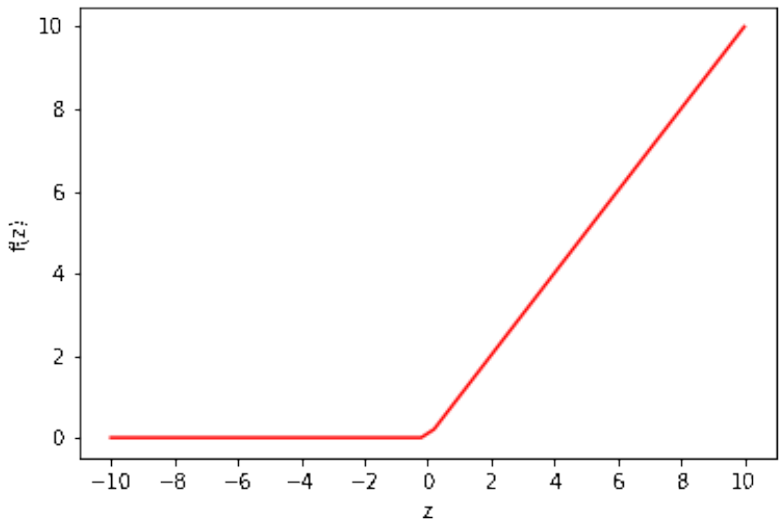


Figura 4.8. Función lineal rectificada ReLU.

$$f(z) = max(0, z). \tag{4.9}$$

En el siguiente capítulo se presentarán los resultados de la aplicación de WienerNet, la cual fue programada y modificada para mapas con diferentes número de píxeles.

Capítulo 5

Resultados

En los capítulos previos se han estudiado los conceptos básicos de aprendizaje automático y de redes neuronales. Luego se estudió particularmente la estructura de la red neuronal WienerNet. En esta sección mostraremos los resultados de aplicar WienerNet a mapas del FCR con diferentes número de píxeles haciendo particular énfasis en la eficiencia de la misma al realizar el WF. Fue preciso comparar los resultados de la red neuronal con los resultados del WF realizado con el método iterativo CG, para eso empleamos el código público NIFTy (*Numerical Information Field Theory*) (Selig et al., 2013) que realiza tal cálculo.

La red neuronal fue programada para el caso de mapas con ruido homogéneo y aproximación de cielo plano, por lo que simulamos estos mapas utilizando las siguientes librerias:

- CAMB (*Code for Anisotropies in the Microwave Background*): código númerico creado por Anthony Lewis y Antony Challinor (Lewis et al., 2000) que resuelve las ecuaciones de Boltzmann acopladas para las diferentes especies de partículas de un modelo cosmológico dado. Lo utilizamos para generar el espectro angular de potencias teórico dado un conjunto de parámetros cosmológicos.
- Healpy: paquete de python basado en HEALPix (*Hierarchical Equal Area isoLatitude Pixelation*) el cual es un algoritmo que realiza la pixelización de la 2-esfera y proyecciones de mapa. Lo utilizamos para la realización de los mapas del FCR y su proyección plana según los espectros teóricos generados con CAMB.

Los parámetros cosmológicos adoptados son los utilizados por defecto en el código CAMB, a saber: parámetro de Hubble, $H_0 = 67.5$ km/s/Mpc; parámetro de densidad de materia bariónica, $\Omega_b h^2 = 0.022$; parámetro de densidad de materia oscura fría, $\Omega_{CDM} h^2 = 0.122$; parámetro de curvatura, $\Omega_k = 0$; masa de neutrinos, $m_\nu = 0.06$; profundidad óptica, $\tau = 0.06$; amplitud del espectro de fluctuaciones escalares, $A_s = 2 \times 10^{-9}$; índice espectral escalar, $n_s = 0.965$; y cociente escalar-tensorial, r= 0.1.

5.1. Predicciones

En primer lugar, estudiamos los resultados de la red neuronal para mapas pequeños sin máscara, para eso los simulamos con:

- Número de píxeles = 28×28
- Resolución angular = 30 [arcmin]

Simulamos el espectro del ruido homogéneo n_l buscando la escala $\hat{l}$ tal que el número de modos donde $C_l > n_l$ es igual al número de modos donde $C_l < n_l$. De esta manera, el ruido

es constante en todos los modos y es igual a $n_l = C_l(\hat{l}) = 0.47$. En la figura 5.1 se presentan el espectro teórico para 28×28 píxeles y el espectro del ruido homogéneo.

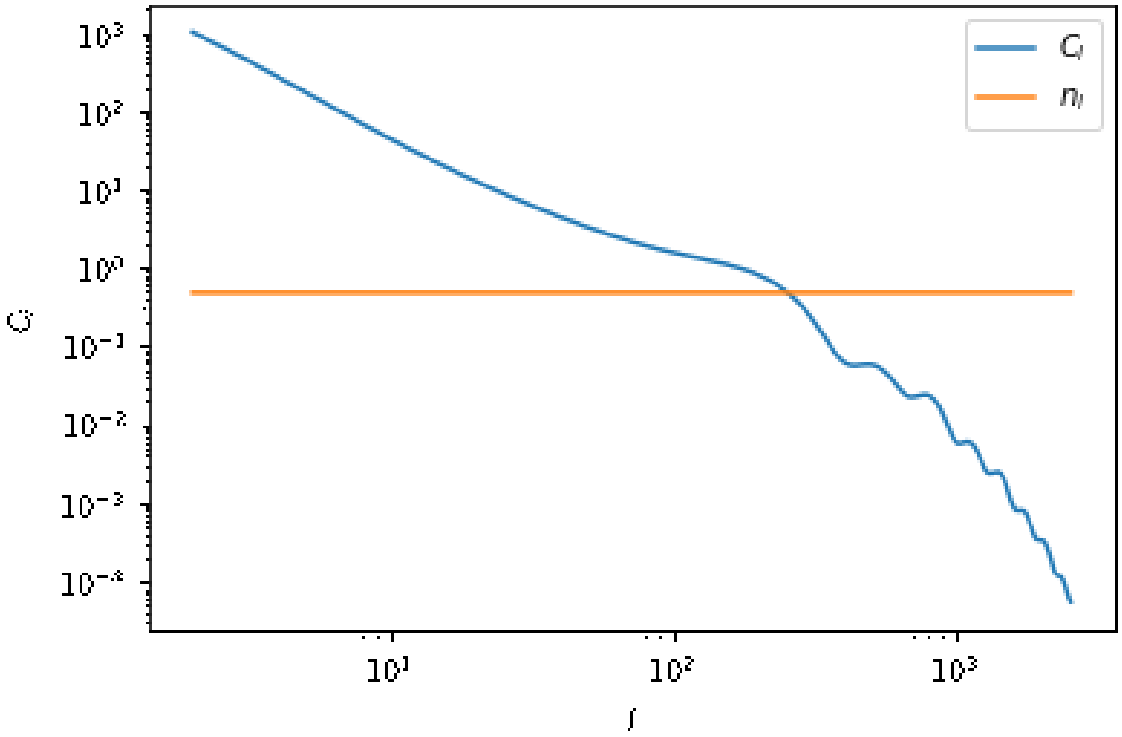


Figura 5.1. Espectro del FCR para 28×28 píxeles y ruido homogéneo.

Adaptamos la red neuronal para mapas con número de píxeles 28×28, la cual consiste de 4 *encoders* y 4 *decoders*. Las capas convolucionales son programadas con la clase de Keras "*Conv2d*", cuyos argumentos a especificar son el número de filtros, el tamaño del kernel, los *strides* horizontal y vertical, y la elección del *padding*. Si el *padding* es igual a *same* se incluyen ceros alrededor de la capa, en caso contrario el *padding* es igual es *valid*. Ver apéndice A para los detalles de la red neuronal.

Hicimos la siguiente elección de hiperparámetros:

- Tamaño del kernel = 5
- Tasa de aprendizaje = 0.0001
- Épocas = 100
- Filtros en cada capa = 5
- Padding en cada capa = valid

Utilizamos un conjunto de entrenamiento de 4000 mapas y un conjunto de validación de 1000 mapas. La red neuronal recibe como entradas ejemplos de mapas con ruido homogéneo aplicado y su correspondiente señal verdadera, que cumple el rol de la etiqueta del ejemplo.

Monitoreamos la minimización de la función de pérdida a lo largo de las épocas a través de una interfaz de Tensorflow llamada Tensorboard. Esta interfaz contiene múltiples herramientas para la visualización de las operaciones que ocurren en la red neuronal, también es útil para correr el mismo modelo con diferentes hiperparámetros y encontrar el conjunto que mejor ajusta a la función de pérdida y otras métricas.

Comparamos el modelo con diferentes tasas de aprendizaje. La elección de ésta es importante para que el entrenamiento sea óptimo; si es muy grande la función de pérdida diverge, si es muy pequeña el entrenamiento será lento y si se elige ligeramente grande la función de pérdida comenzará haciendo grandes progresos pero luego oscilará alrededor de un valor constante sin alcanzar un valor óptimo.

Evaluamos la función de pérdida 4.3 en el espacio de configuración de píxeles y tomamos $A = I$. En este caso, la función de pérdida es la suma de la diferencia de píxeles entre la predicción de la red y la señal al cuadrado:

$$J = \sum_{i}^{n_{pix}} (T_i^{CNN} - T_i^s)^2. \tag{5.1}$$

Con la elección de la tasa de aprendizaje que hicimos, en la figura 5.2 se puede observar el comportamiento de la función de pérdida a lo largo de las épocas, extraído de TENSORBOARD.

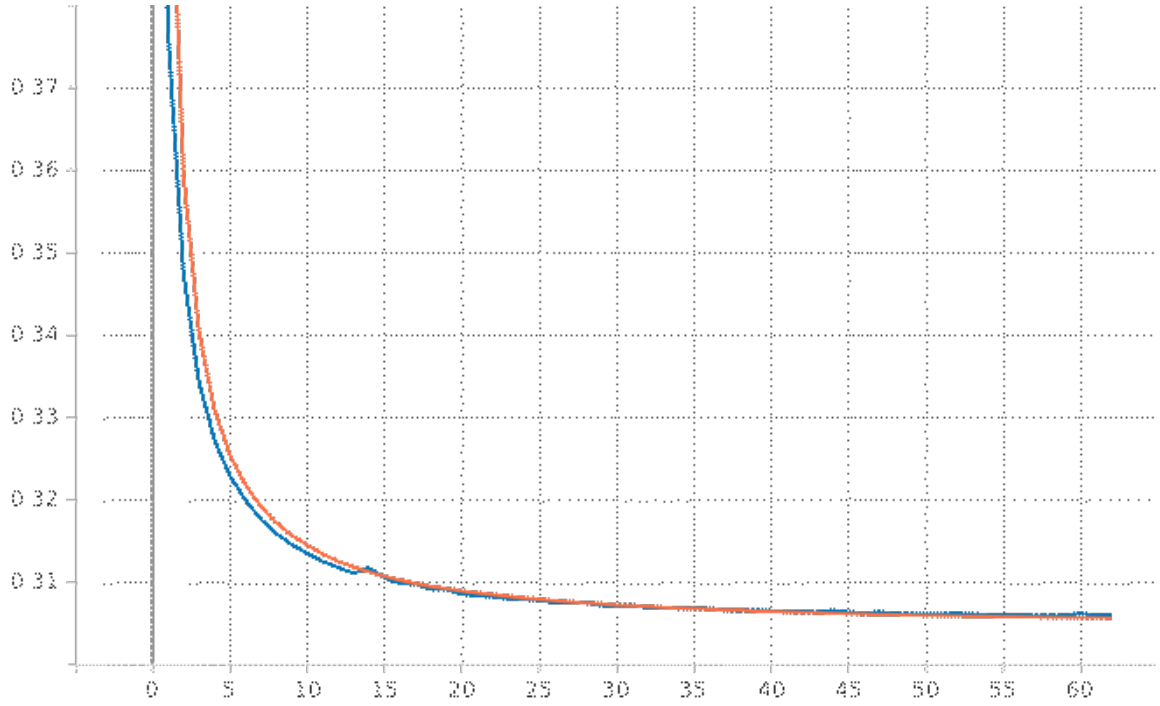


Figura 5.2. Función de pérdida vs épocas. La línea naranja corresponde a la pérdida del conjunto de entrenamiento y la azul la pérdida del conjunto de validación.

Utilizamos un objeto de KERAS llamado *early stopping* el cual detiene el entrenamiento de la red si la función de pérdida del conjunto de validación no mejora en el transcurso de 5 épocas. Notamos en el gráfico 5.2 que el entrenamiento se detuvo en la época 60, alcanzando un valor constante.

Cuando el entrenamiento de la red neuronal finaliza obtenemos un modelo optimizado el cual podemos utilizar para hacer predicciones sobre conjuntos de datos nuevos. Generamos 300 mapas como conjunto de prueba y realizamos predicciones sobre ellos con la función de KERAS "*model.predict()*".

En la figura 5.3 presentamos la predicción de la red de uno de los 300 mapas, en el primer panel tenemos el mapa de la señal generado con HEALPY, en el segundo panel la misma señal con rudio homogéneo aplicado y en el tercer panel visualizamos el mapa predicho por la red

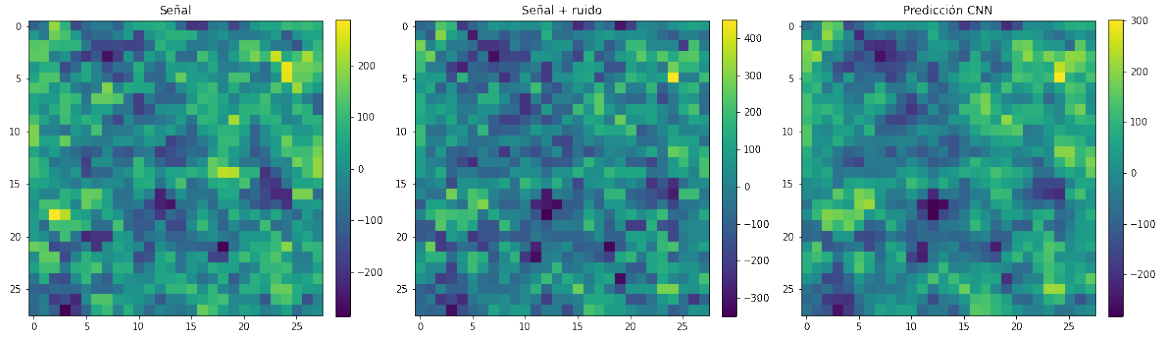


Figura 5.3. Mapa de 28 × 28 píxeles de señal verdadera, señal contaminada con rudio y predicción de la red neuronal.

neuronal para este ejemplo. Para este mismo mapa estudiamos la intensidad de los píxeles del mapa predicho por la red neuronal y el mapa de la señal verdadera, encontrando que no se apartan significativamente de una recta, como puede observarse en la figura 5.4.

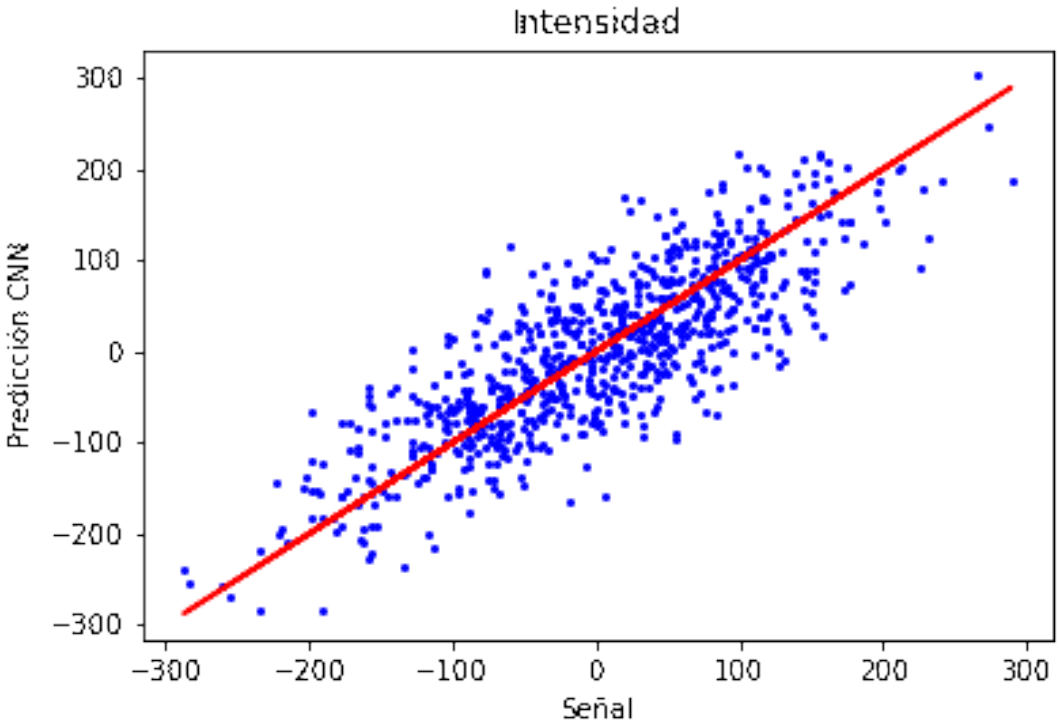


Figura 5.4. Intensidad de los píxeles, predicción de la red neuronal vs señal.

También lo analizamos mediante una histograma de la diferencia en la intensidad de los píxeles de la señal verdadera y la predicción de la red, con media = 0.29 y desviación estándar = 56.24. En la figura 5.5 notamos una concentración alrededor de cero y unos pocos valores con diferencias de hasta 150.

A partir de estos resultados, podemos inferir que la red neuronal predice como resultado un mapa cuyos píxeles no se apartan significativamente de el mapa de la señal verdadera.

La matriz de varianza-covarianza de los residuos definida en 2.4 y 2.11 contiene en la diagonal las varianzas en cada píxel promediadas sobre todos los mapas, por lo tanto, la traza

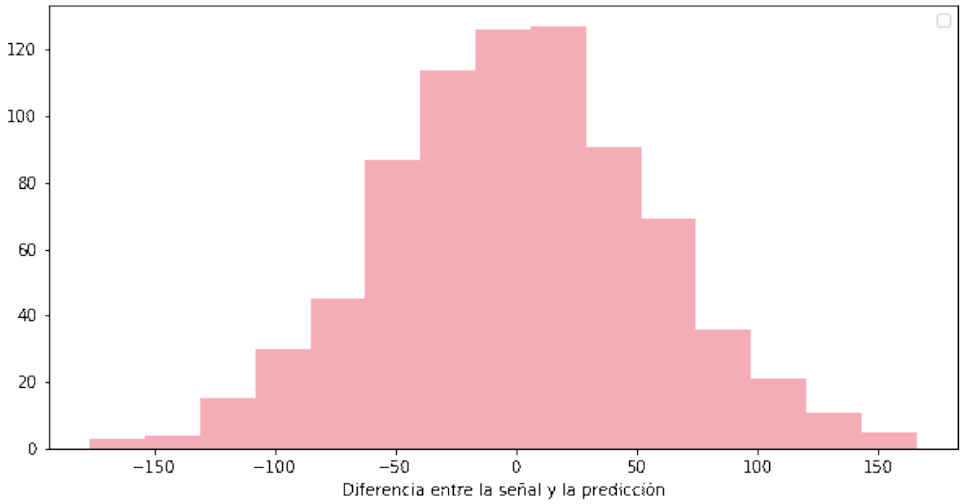


Figura 5.5. Histograma de la diferencia entre la señal y la predicción de la red neuronal.

de esta matriz resulta en una suma de las varianzas sobre todos los píxeles. Por otro lado, al minimizar la función de pérdida, de la ecuación 5.1, la red neuronal realiza un promedio en el conjunto de datos. Por lo tanto, comparamos la raíz cuadrada de ambas ecuaciones para corroborar teóricamente si la red neuronal está realizando efectivamente el WF.

Para eso evaluamos la función de pérdida con el modelo ya entrenado, utilizando la función de KERAS "*model.evaluate(x,y,..)*" la cual recibe como argumentos los mapas con ruido y sus correspondientes mapas de señal del conjunto de prueba:

- $\sqrt{Tr\langle rr^{\dagger}\rangle} = 0.5514$
- $\sqrt{\langle J\rangle} = 0.5523$

Ambos resultados coinciden con gran precisión lo que nos indica que la red neuronal empleada realiza el WF que esperamos.

Comparamos las predicciones de la red con las resultados del WF realizado con el método iterativo CG utilizando el código público Nifty. Esto nos permite evaluar que tan bueno es el desempeño del WF realizado con la red neuronal respecto del método tradicional.

En la figura 5.6 presentamos la comparación del mismo mapa que estábamos estudiando, en el tercer panel visualizamos el mapa predicho por la red neuronal y en el cuarto panel la realización del WF con el método tradicional.

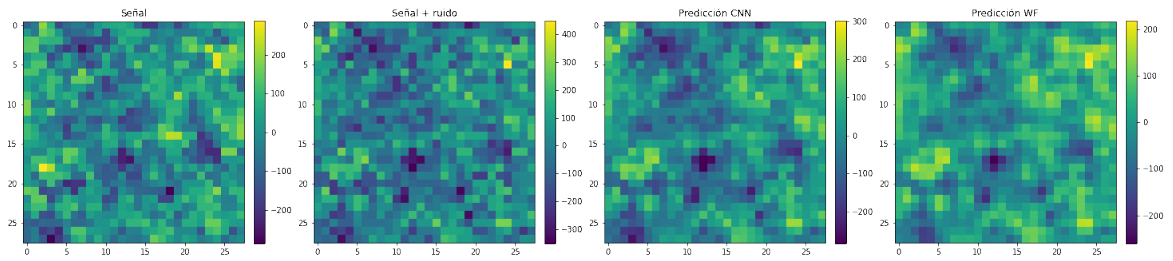


Figura 5.6. Mapa de señal verdadera, mapa de señal contaminada con ruido, mapa de predicción con la red neuronal y mapa de resultado del WF realizado con CG.

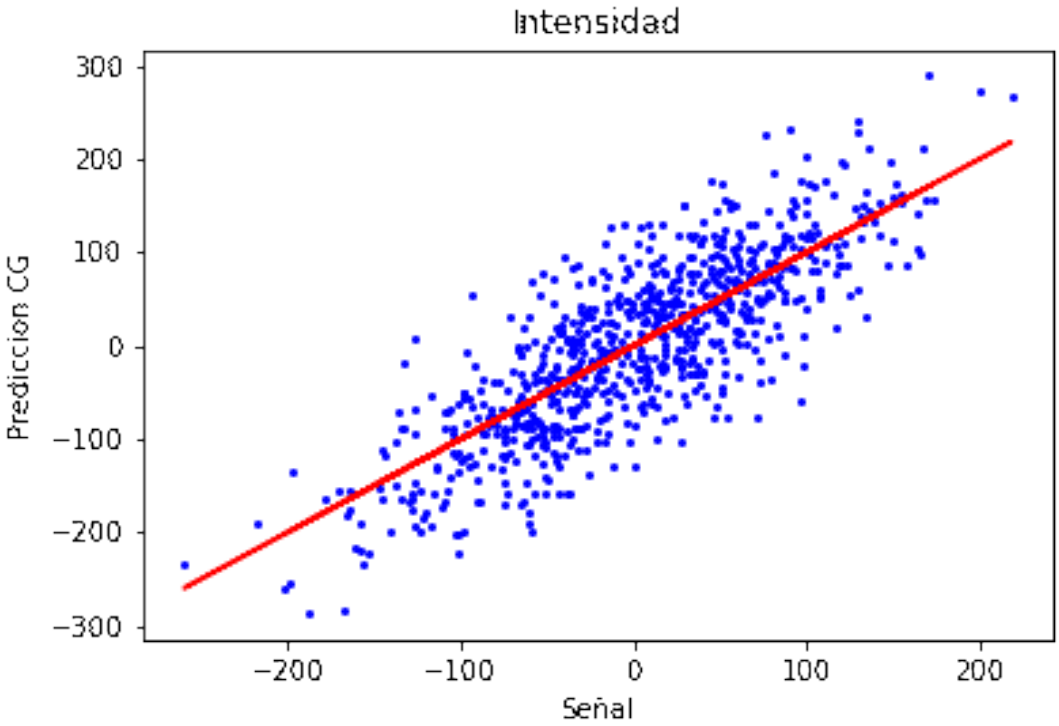


Figura 5.7. Intensidad de los píxeles, resultado de realizar el WF con el método CG vs señal.

En la figura 5.7 presentamos en el eje de abscisas la intensidad de los píxeles del mapa de la señal verdadera y en el eje de ordenadas la intensidad de los píxeles de la realización del WF con el método CG, hallando un comportamiento similar al de la figura 5.4.

Calculamos la diferencia en intensidad píxel a píxel para ambos métodos y realizamos un histograma con media = 0.29 y desviación estándar = 25.02, presentado en la figura 5.8. Se puede notar un comportamiento gaussiano, donde la mayoría de los valores se encuentran alrededor del cero y la máxima diferencia es de 75 para unos pocos píxeles.

Por último, calculamos la distribución de desviaciones estándar de la diferencia entre la señal verdadera y los resultados realizados con ambos métodos para los 300 mapas, como se puede observar en la figrua 5.9. Las medias de la desviación estándar son:

- $\sigma_{CNN} = 0.58$
- $\sigma_{WF} = 0.56$

Ambos resultados coinciden con gran precisión, por lo tanto, podemos concluir que la red neuronal WienerNet realiza el filtro de Wiener de forma satisfactoria y las predicciones sobre mapas nuevos se corresponden en gran acuerdo con los resultados esperados por el método tradicional. Esto nos indica que realizar el WF con redes neuronales es un método factible para reducir el ruido presente en la señal y, por ende, la reconstrucción de la señal original.

5.2. Eficiencia

En la sección anterior estudiamos en detalle las predicciones de la red neuronal para mapas pequeños de 28×28 píxeles y sin máscara. Esto nos fue útil para comprobar que la red neuronal entrenada reproduce los resultados esperados por el filtro de Wiener y comprender

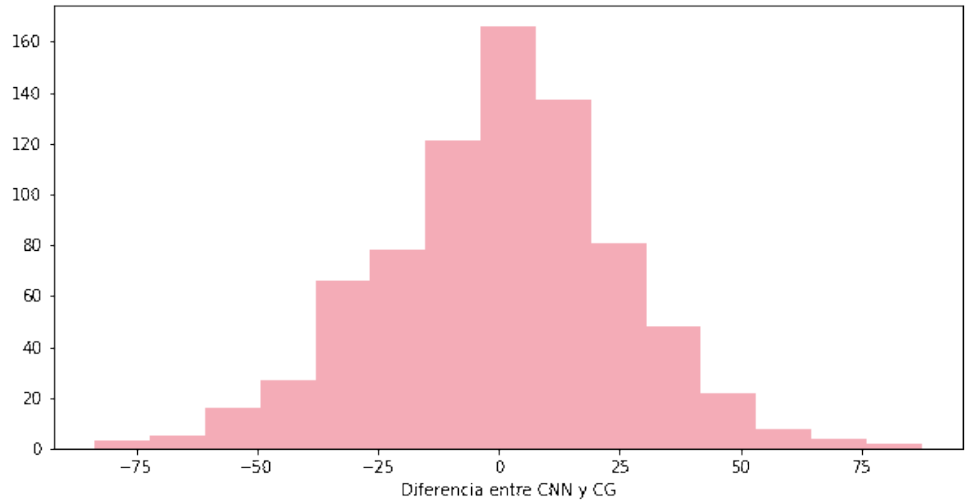


Figura 5.8. Histograma de la diferencia de intensidad entre la predicción de la red neuronal CNN y el resultado realizado con el método tradicional CG.

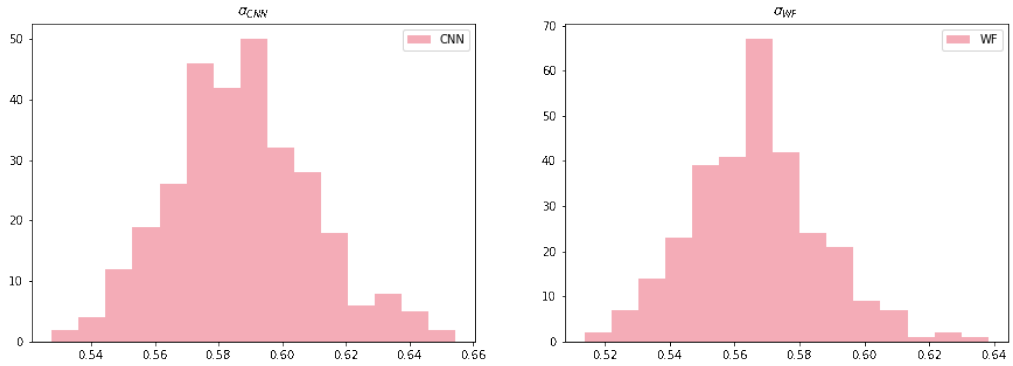


Figura 5.9. Histograma de la distribución de σ de la diferencia entre la señal y los resultados con ambos métodos para los 300 mapas. El panel de la izquierda corresponde a el resultado de la red neuronal y el panel de la derecha a el WF realizado con el método CG.

los detalles de la misma. En esta sección estudiamos mapas más realistas, lo cual implicó el diseño de estructuras más complejas de la red neuronal.

El fin de la implementación de aprendizaje automático, y específicamente de redes neuronales, para el WF proviene del costo computacional que implica el método iterativo CG. Estudiamos la eficiencia de WienerNet respecto del método tradicional analizando como escalea con el tiempo los resultados de ambos métodos para mapas de diferente número de píxeles. Aumentamos la dificultad aplicando una máscara a los distintos mapas y probamos distintos niveles de ruido para estudiar como repercute el mismo en los tiempos.

Para computar el espectro de los distintos niveles de ruido homogéneo calculamos las escalas angulares:

- $\hat{l}$: escala angular tal que el número de modos donde $C_l > n_l$ es igual al número de modos donde $C_l < n_l$.
- $\hat{l}_{low}$: escala angular tal que el número de modos donde $C_l > n_l$ es igual a $1/4$ del número

de modos donde $C_l < n_l$.

- $\hat{l}_{high}$: escala angular tal que el número de modos donde $C_l > n_l$ es igual a 3/4 del número de modos donde $C_l < n_l$.

Simulamos mapas con diferentes números de píxeles pero dejando fijo el tamaño angular del mapa (10 deg × 10 deg), como se puede ver en la tabla 5.1.

L_{size}	N_{pix}	$\hat{l}$	$\hat{l}_{low}$	$\hat{l}_{high}$	Resolución
10	56	713	504	873	10.71 arcmin
10	128	1629	1152	1996	4.68 arcmin
10	256	3254	2304	3991	2.34 arcmin
10	512	6517	4608	7982	1.17 arcmin

Tabla 5.1. Mapas simulados con distinto número de píxeles y niveles de ruido.

Generamos el espectro angular de potencias teórico para los diferentes número de píxeles y el espectro del ruido homogéneo para las diferentes escalas. Estos pueden verse en las figuras 5.10, 5.11, 5.12 y 5.13.

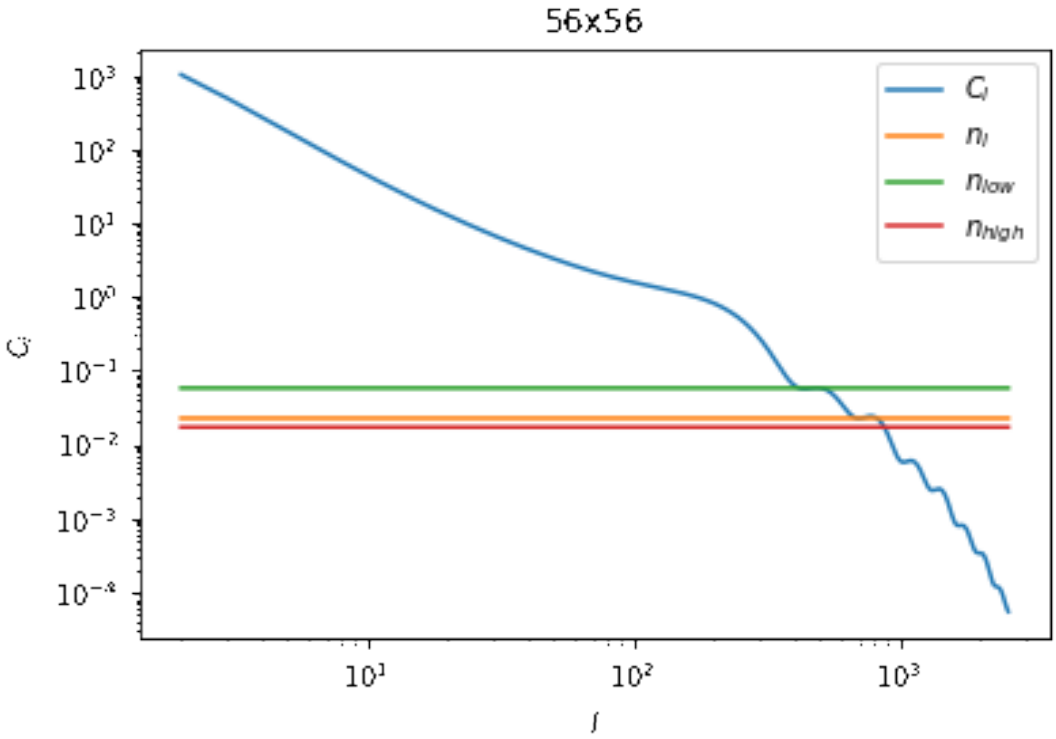


Figura 5.10. Espectro del FCR para 56 × 56 píxeles y distintos niveles de ruido.

Para el entrenamiento de las redes neuronales utilizamos los mismos hiperparámetros que la red neuronal de 28 × 28 píxeles. El conjunto de entrenamiento contiene 4000 mapas y el conjunto de validación 1000 mapas. Para el conjunto de prueba utilizamos 300 mapas. Ver apéndice A para las estructuras de las redes neuronales que diseñamos para cada caso específico.

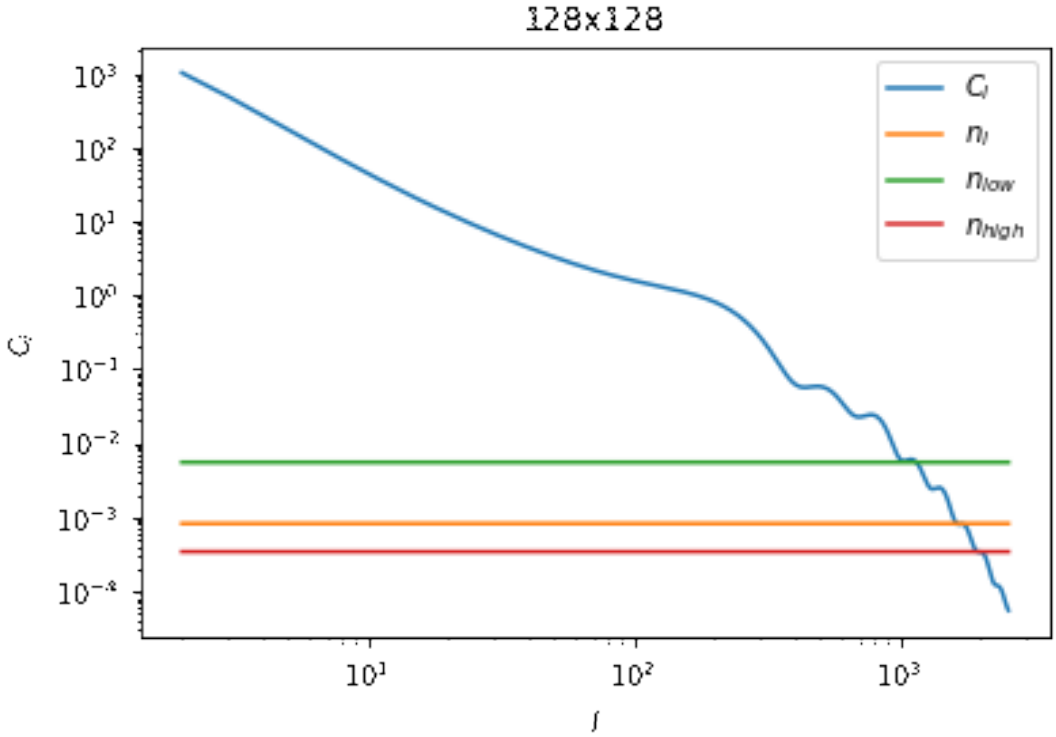


Figura 5.11. Espectro del FCR para 128 × 128 píxeles y distintos niveles de ruido.

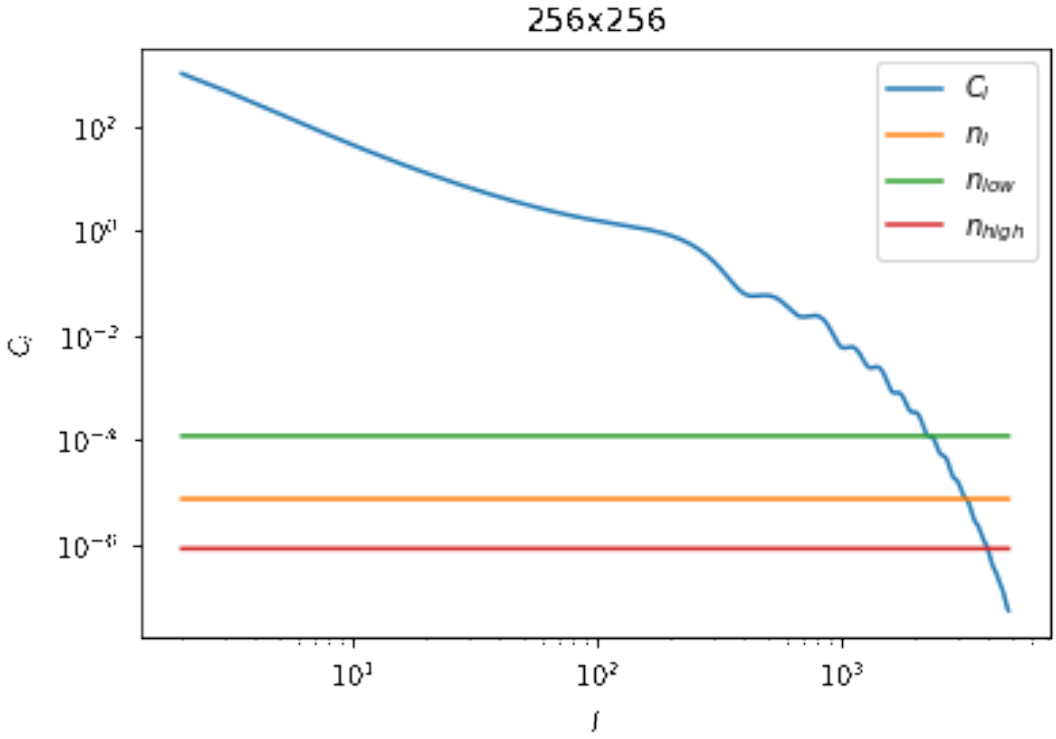


Figura 5.12. Espectro del FCR para 256 × 256 píxeles y distintos niveles de ruido.

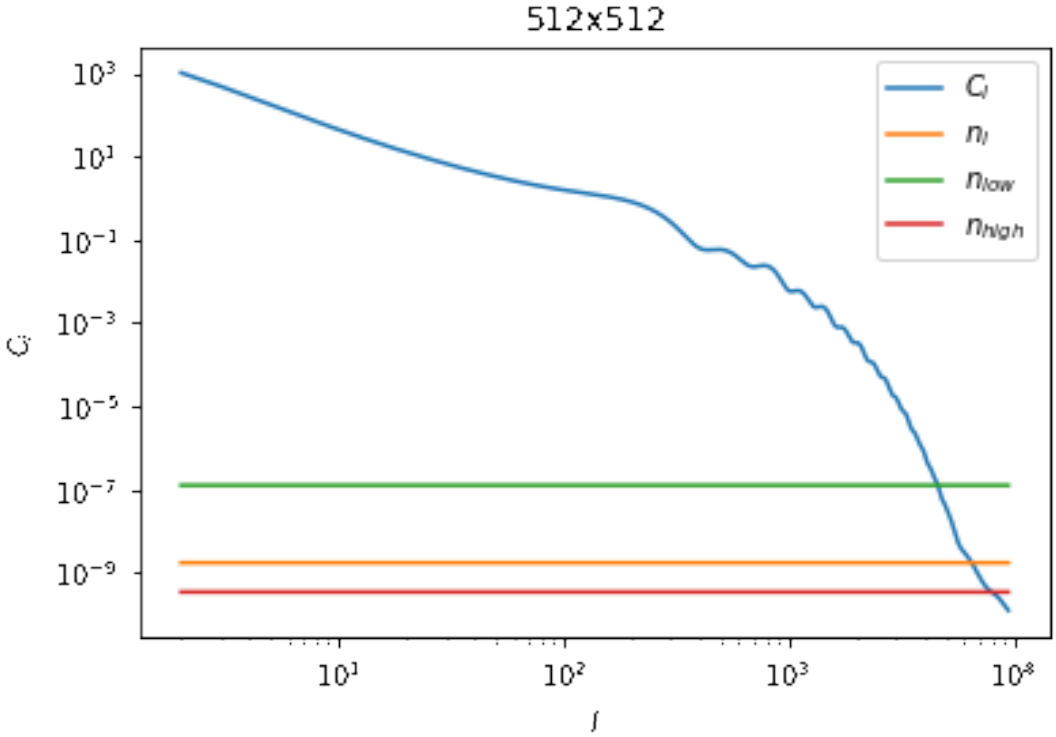


Figura 5.13. Espectro del FCR para 512×512 píxeles y distintos niveles de ruido.

Realizamos predicciones para los distintos mapas con máscara y nivel de ruido $\hat{l}$, para estudiar el desempeño de la red neuronal en presencia de la máscara.

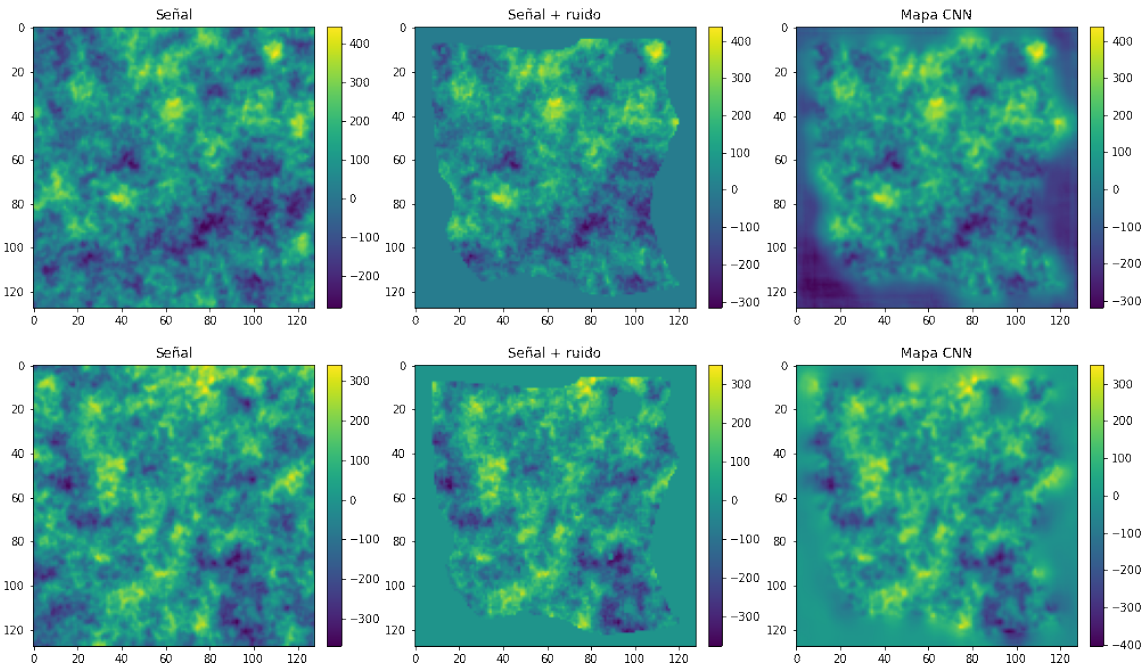


Figura 5.14. Mapas de señal, señal con ruido y máscara, y predicción de la red neuronal.

En la figura 5.14 presentamos los resultados para dos mapas de 128 × 128 píxeles. En el primer panel se encuentra el mapa de señal, en el segundo la señal contaminada con ruido y una máscara aplicada de ruido infinito, y en el tercer panel la predicción de la red neuronal. Visualmente el mapa predicho por la red neuronal concuerda en gran medida con el mapa de la señal en las zonas sin máscara.

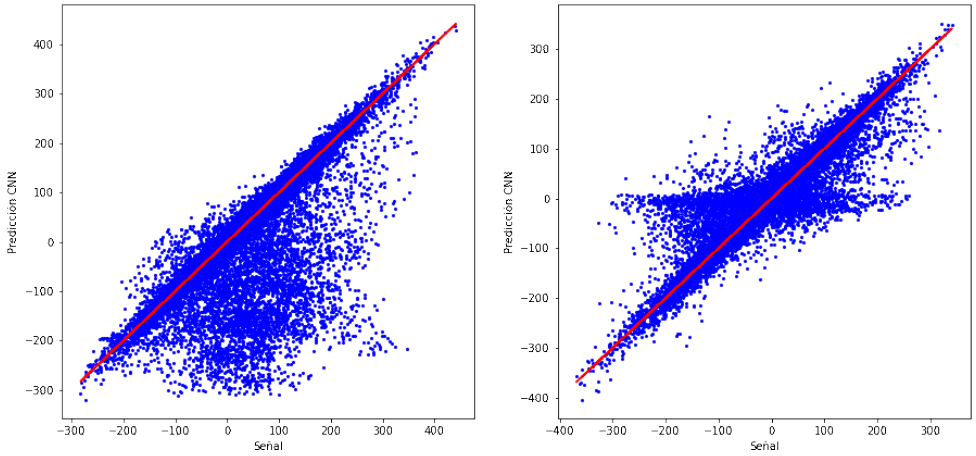


Figura 5.15. Intensidad de los píxeles para 2 mapas de 128 × 128 píxeles. Predicción red neuronal vs señal.

En la figura 5.15 se observa que la intensidad de los píxeles para los mapas predichos se encuentran en una recta respecto de la intensidad de los píxeles de la señal. Los píxeles que se apartan de la recta son los que se encuentran en la región de la máscara.

Podemos concluir que la red neuronal programada con un camino no-lineal para la máscara reproduce eficientemente los resultados del WF.

Para analizar la eficiencia de WienerNet con respecto al método tradicional, calculamos el tiempo que le toma a la red neuronal realizar las predicciones sobre el conjunto de prueba para los distintos niveles de ruido, los cuales se presentan en la tabla 5.2.

Tiempo [seg]				
l	56	128	256	512
$\hat{l}$	1.06	7.01	41.9	54.9
$\hat{l}_{low}$	1.11	5.71	43.11	58.49
$\hat{l}_{high}$	1.07	6.51	41.93	59.52

Tabla 5.2. Tiempo de cómputo en segundos para número de píxeles 56, 128, 256 y 512 para la red neuronal.

En la figura 5.16 se puede notar que para los mapas de 56 × 56 píxeles el tiempo de cómputo es del orden de 1 segundo, mientras que para los mapas de 512 × 512 píxeles cercano a un 1 minuto.

Ahora bien, calculamos el tiempo de cómputo de la realización del WF con el método iterativo CG para el mismo conjunto de mapas, a fin de poder realizar una comparación entre ambos métodos. Estos se presentan en la tabla 5.3.

En la figura 5.17 se observa que para mapas de 56x56 píxeles el tiempo de cómputo es

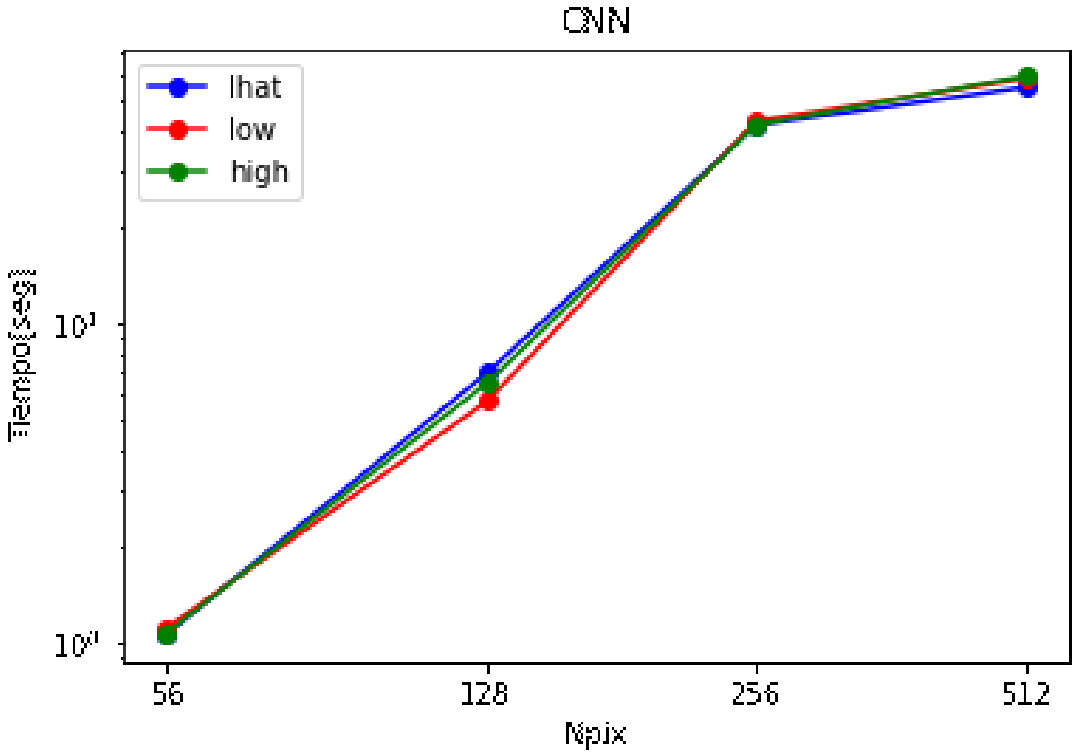


Figura 5.16. Escaleo con el tiempo de cómputo del número de píxeles para la CNN.

Tiempo[seg]				
l	56	128	256	512
$\hat{l}$	1.76	27.4	1570	759600
$\hat{l}_{low}$	1.11	10.5	383	73500
$\hat{l}_{high}$	1.77	45.6	4909	1682100

Tabla 5.3. Tiempo de cómputo en segundos para número de píxeles 56, 128, 256 y 512, realizado con el CG

del orden de ∼1.5 seg., resultado parecido al tiempo de cómputo de la red neuronal con los mismos mapas. A partir de mapas de 128×128 píxeles, el tiempo que le lleva realizar el WF con CG supera significativamente el tiempo de cómputo empleado por la red neuronal, llegando a tardar del orden de ∼2 horas para 256×256 píxeles y del orden de días para 512×512 píxeles.

Además, notamos que la realización del WF es más rápida para mapas con nivel de ruido correspondiente a la escala $\hat{l}_{low}$ y más lenta para el nivel de ruido correspondiente a la escala $\hat{l}_{high}$, mientras que la escala $\hat{l}$ se mantiene en el medio de ambos niveles, siendo este un comportamiento que se mantiene en todos los números de píxeles. Esto se debe a que si el espectro de la señal es predominante respecto del nivel de ruido, como ocurre para el caso de $\hat{l}_{high}$, el proceso de WF es más lento, caso contrario si hay mayor cantidad de modos donde predomina el ruido.

Gracias a estos resultados podemos concluir que efectivamente la realización del WF a través de un método que requiera inversión de matrices con CG es altamente costoso computacionalmente e implica una gran complicación para la explotación del análisis de datos del

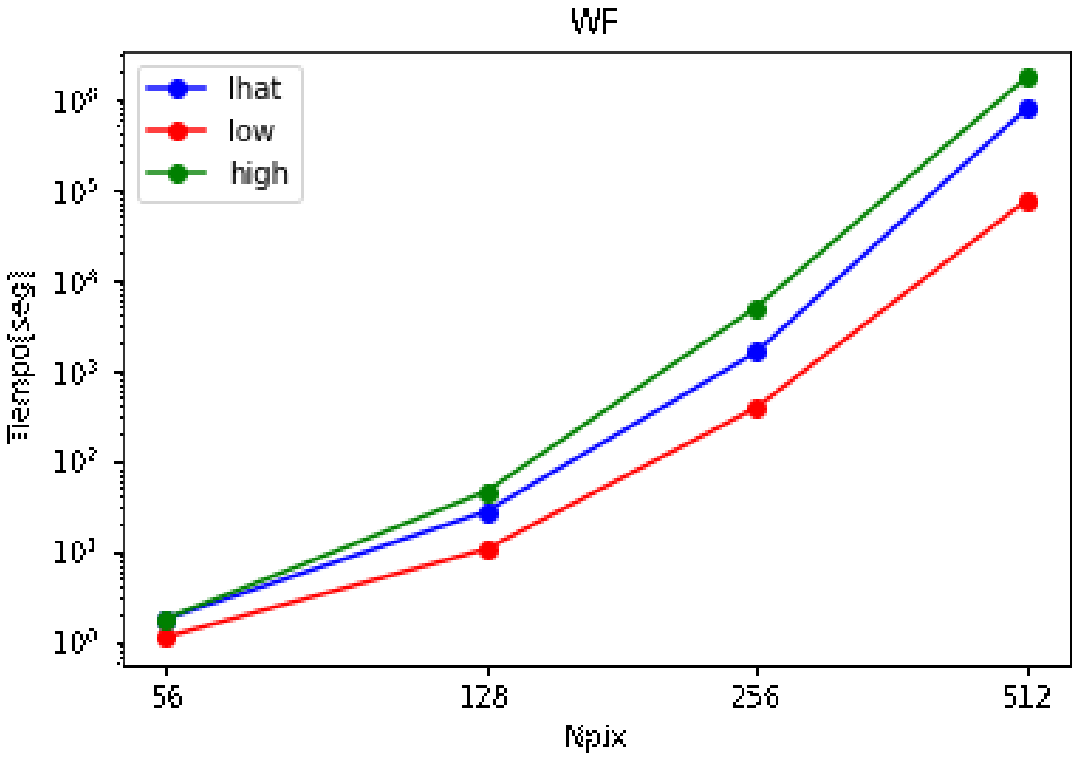


Figura 5.17. Escaleo con el tiempo de cómputo del número de píxeles para el método CG.

FCR. La importancia de la realización del WF hace que sea de particular interés desarrollar métodos alternativos para su implementación, donde las redes neuronales han demostrado durante este trabajo de Tesis reproducir resultados favorables y coherentes. Es en esta dirección donde buscamos extender la red neuronal WienerNet para que pueda ser aplicada a casos de mapas más realistas, y de interés en la generación de experimentos actuales.

Si bien las predicciones de la red neuronal WienerNet son extremadamente rápidas y eficientes, es necesario tener en cuenta que el entrenamiento de la misma conlleva un largo tiempo, dependiendo del tamaño de los mapas y la complejidad de la estructura.

l	Tiempo de entrenamiento [horas]
$\hat{l}$	3.51
$\hat{l}_{low}$	3.10
$\hat{l}_{high}$	4.52

Tabla 5.4. Tiempo de entrenamiento de la red neuronal de 56×56 píxeles.

l	Tiempo de entrenamiento [horas]
$\hat{l}$	34
$\hat{l}_{low}$	27
$\hat{l}_{high}$	26

Tabla 5.5. Tiempo de entrenamiento de la red neuronal de 128×128 píxeles.

En las tablas 5.4 y 5.5 se presentan los tiempos de entrenamiento de las redes neuronales

para 56×56 píxeles y 128×128 píxeles realizadas con el cluster Helios del Instituto de Estudios Avanzados de la Universidad de Princeton. Las redes neuronales diseñadas para mapas del FCR de 56×56 píxeles contienen 5 *encoders* y 5 *decoders*, mientras que las redes neuronales para mapas del FCR de 128×128 píxeles contienen 6 *encoders* y 6 *decoders*, por lo tanto es de esperar que con el aumento de la cantidad de *autoencoders* lleve más tiempo el tiempo de entrenamiento.

Para las redes neuronales correspondientes a 256×256 píxeles y 512×512 píxeles fue necesario recurrir a el uso de computadoras con tarjeta gráfica integrada (*graphics processing unit*, GPUs), las cuales poseen un procesador que acelera notablemente el tiempo de entrenamiento de las mismas. Los tiempos se presentan en las tablas 5.6 y 5.7.

l	Tiempo de entrenamiento [horas]
$\hat{l}$	0.51
$\hat{l}_{low}$	0.57
$\hat{l}_{high}$	1.36

Tabla 5.6. Tiempo de entrenamiento de la red neuronal de 256×256 píxeles con la utilización de GPUs.

l	Tiempo de entrenamiento [horas]
$\hat{l}$	2.18
$\hat{l}_{low}$	1.03
$\hat{l}_{high}$	0.56

Tabla 5.7. Tiempo de entrenamiento de la red neuronal de 512×512 píxeles con la utilización de GPUs.

Capítulo 6

Discusión y conclusiones

En esta Tesis se realizó un estudio detallado sobre técnicas de aprendizaje automático útiles para el análisis de datos cosmológicos. Estudiamos el mecanismo de aprendizaje de las redes neuronales, cómo se estructuran, y cómo, a partir de éstas, se pueden obtener resultados de interés para los problemas que afronta la cosmología actualmente. Para eso fue necesario el aprendizaje de los paquetes diseñados para la programación de redes neuronales tales como TENSORFLOW, y específicamente la librería KERAS, con la cual programamos las estructuras que se detallan en el apéndice A.

Los paquetes mencionados están desarrollados en lenguaje de programación PYTHON, por lo que también fue preciso un profundo aprendizaje del mismo ya sea para la elaboración de los códigos correspondientes a las estructuras de la red neuronal, como el posterior análisis de los datos presentado en el capítulo 5.

Siguiendo esta línea, estudiamos el caso concreto de la realización del filtrado de Wiener, el cual consiste esencialmente de un filtro que reduce el ruido presente en la señal de entrada. Este mecanismo es de particular interés en el área de procesamiento de señales, ya que es el procedimiento estadístico óptimo para la obtención de información útil a partir de un conjunto de datos incompleto y con ruido, como se detalla en el capítulo 2.

Realizar el filtro de Wiener requiere de la inversión de matrices como se puede apreciar en la ecuación 2.10, las cuales se realizan a través de un método iterativo que utiliza el *conjugate gradient*. Esto requirió el estudio numérico de la ejecución de este método mediante el código público NIFTY.

En la literatura se detalla que el costo computacional del filtro de Wiener es muy alto y constituye un cuello de botella para el análisis de datos cosmológicos. Siendo éste un procedimiento necesario y útil para la reconstrucción de la señal del FCR estudiamos la realización del filtro de Wiener mediante la red neuronal WienerNet, como se indica en detalle en el capítulo 4. Para eso, fue necesario generar un conjunto de entrenamiento para enviar como entradas a la red neuronal, compuesto por mapas de temperatura del FCR. Esto requirió el aprendizaje de la simulación de los mapas con las librerias CAMB y HEALPY.

Habiendo estudiado el desempeño de la red neuronal, comparamos su eficiencia para reconstruir la distribución de temperatura del FCR a nivel de mapas con respecto al método con CG. Hallamos efectivamente que el tiempo de cómputo del WF mediante el método CG es excesivamente alto, llegando a alcanzar horas, para mapas con número de píxeles cada vez mayores, mientras que el tiempo de cómputo de la red neuronal no alcanza los minutos, como mostramos en las figuras 5.16 y 5.17.

Concluímos, con estos resultados, que la red neuronal WienerNet es un método eficiente para realizar el filtro de Wiener. La importancia de este resultado radica en que los experimentos actuales deben prescindir de la utilización del filtro de Wiener por su costo computacional

con el método CG, a pesar de ser necesario para que el análisis de los datos sea lo más óptimo posible. Su implementación con aprendizaje automático puede ser de utilidad para la próxima generación de instrumentos, pero ésta fue realizada únicamente para mapas con ruido homogéneo y aproximación de cielo plano para la proyección de los mapas. Por lo tanto, nuestro interés como trabajo futuro es desarrollar extensiones de esta red neuronal para mapas con ruido inhomogéneo en los píxeles y múltiples subfrecuencias, como los que van a ser producidos por el experimento QUBIC (*Q-U Bolometric Interferometer for Cosmology*)[(i)], siendo así un caso más realista. También se espera generalizar estos resultados a mapas de la polarización del FCR.

Luego de este análisis pretendemos estudiar la implementación de técnicas de aprendizaje automático y redes neuronales a otros problemas del análisis de datos cosmológicos como la realización de mapas de temperatura y polarización a partir de los datos ordenados por tiempo.

[(i)] https://www.qubic.org.ar/

Apéndice A

Arquitectura detallada de red neuronal

La estructura de la red neuronal se encuentra definida en la figura 4.7. Todos los *encoders* y *decoders* consisten de una sola capa convolucional con *padding* "valid", por lo que a lo largo de las operaciones en red, los mapas pierden información de sus bordes. Extendemos periódicamente los mapas de entrada de modo tal que la salida de la red tenga el tamaño esperado.

A.1. Npix=28

Extendemos la imagen $28 + 2 \times 28 = 84$

Capas	**input**	**output**	**tamaño kernel**	**filtros**	**stride**	**padding**
Encoder0	84×84	80×80	5×5	5	1	valid
Encoder1	80×80	38×38	5×5	5	2	valid
Encoder2	38×38	17×17	5×5	5	2	valid
Encoder3	17×17	7×7	5×5	5	2	valid
Decoder3	7×7	10×10	5×5	5	2	valid
Decoder2	10×10	16×16	5×5	5	2	valid
Decoder1	16×16	28×28	5×5	5	2	valid
Decoder0	28×28	28×28	5×5	1	1	same

Tabla A.1. Arquitectura de red neuronal para 28×28 píxeles con tamaño de kernel 5×5.

A.2. Npix=56

Extendemos la imagen $56 + 2 \times 56 = 168$

Capas	**input**	**output**	**tamaño kernel**	**filtros**	**stride**	**padding**
Encoder0	168×168	164×164	5×5	5	1	valid
Encoder1	164×164	80×80	5×5	5	2	valid
Encoder2	80×80	38×38	5×5	5	2	valid
Encoder3	38×38	17×17	5×5	5	2	valid
Encoder4	17×17	7×7	5×5	5	2	valid
Decoder4	7×7	10×10	5×5	5	2	valid
Decoder3	10×10	16×16	5×5	5	2	valid
Decoder2	16×16	28×28	5×5	5	2	valid
Decoder1	28×28	56×56	5×5	5	2	valid
Decoder0	56×56	56×56	5×5	1	1	same

Tabla A.2. Arquitectura de red neuronal para 56×56 píxeles con tamaño de kernel 5×5.

A.3. Npix=128

Extendemos la imagen $128 + 2 \times 128 = 384$

Capas	**input**	**output**	**tamaño kernel**	**filtros**	**stride**	**padding**
Encoder0	384×384	380×380	5×5	5	1	valid
Encoder1	380×380	188×188	5×5	5	2	valid
Encoder2	188×188	92×92	5×5	5	2	valid
Encoder3	92×92	44×44	5×5	5	2	valid
Encoder4	44×44	20×20	5×5	5	2	valid
Encoder5	20×20	8×8	5×5	5	2	valid
Decoder5	8×8	12×12	5×5	5	2	valid
Decoder4	12×12	20×20	5×5	5	2	valid
Decoder3	20×20	36×36	5×5	5	2	valid
Decoder2	36×36	68×68	5×5	5	2	valid
Decoder1	68×68	132×132	5×5	5	2	valid
Decoder0	132×132	128×128	5×5	1	1	valid

Tabla A.3. Arquitectura de red neuronal para 128×128 píxeles con tamaño de kernel 5×5.

A.4. Npix=256

Extendemos la imagen $256 + 2 \times 256 = 768$

Capas	**input**	**output**	**tamaño kernel**	**filtros**	**stride**	**padding**
Encoder0	768×768	764×764	5×5	5	1	valid
Encoder1	764×764	380×380	5×5	5	2	valid
Encoder2	380×380	188×188	5×5	5	2	valid
Encoder3	188×188	92×92	5×5	5	2	valid
Encoder4	92×92	44×44	5×5	5	2	valid
Encoder5	44×44	20×20	5×5	5	2	valid
Encoder6	20×20	8×8	5×5	5	2	valid
Decoder6	8×8	12×12	5×5	5	2	valid
Decoder5	12×12	20×20	5×5	5	2	valid
Decoder4	20×20	36×36	5×5	5	2	valid
Decoder3	36×36	68×68	5×5	5	2	valid
Decoder2	68×68	132×132	5×5	5	2	valid
Decoder1	132×132	260×260	5×5	5	2	valid
Decoder0	260×260	256×256	5×5	1	1	valid

Tabla A.4. Arquitectura de red neuronal para 256×256 píxeles con tamaño de kernel 5×5.

A.5. Npix=512

Extendemos la imagen $512 + 2 \times 128 = 768$

Capas	**input**	**output**	**tamaño kernel**	**filtros**	**stride**	**padding**
Encoder0	768×768	764×764	5×5	5	1	valid
Encoder1	764×764	380×380	5×5	5	2	valid
Encoder2	380×380	188×188	5×5	5	2	valid
Encoder3	188×188	92×92	5×5	5	2	valid
Encoder4	92×92	44×44	5×5	5	2	valid
Encoder5	44×44	20×20	5×5	5	2	valid
Decoder5	20×20	36×36	5×5	5	2	valid
Decoder4	36×36	68×68	5×5	5	2	valid
Decoder3	68×68	132×132	5×5	5	2	valid
Decoder2	132×132	260×260	5×5	5	2	valid
Decoder1	260×260	516×516	5×5	5	2	valid
Decoder0	516×516	512×512	5×5	1	1	valid

Tabla A.5. Arquitectura de red neuronal para 512×512 píxeles con tamaño de kernel 5×5.

Bibliografía

Ahn C. P., et al., 2012, ApJS, 203, 21

Bennett C. L., et al., 2003, ApJS, 148, 1

Caldeira J., Wu W. L. K., Nord B., Avestruz C., Trivedi S., Story K. T., 2019, Astronomy and Computing, 28, 100307

Chollet F., 2017, Deep Learning with Python. Manning Publications, New York, NY

Colless M., et al., 2001, MNRAS, 328, 1039

Dodelson S., 2003, Modern Cosmology. Academic Press, Elsevier Science

Elsner F., Wandelt B. D., 2012, arXiv e-prints, p. arXiv:1211.0585

Géron A., 2019, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc.

Kubat M., 2015, An Introduction to Machine Learning, 1st edn. Springer Publishing Company, Incorporated

Lewis A., Challinor A., Lasenby A., 2000, ApJ, 538, 473

Mishra A., Reddy P., Nigam R., 2019, arXiv e-prints, p. arXiv:1908.04682

Münchmeyer M., Smith K. M., 2019, arXiv e-prints, p. arXiv:1905.05846

Penzias A. A., Wilson R. W., 1965, ApJ, 142, 419

Planck Collaboration et al., 2016, A&A, 594, A8

Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P., 2007, Numerical Recipes 3rd Edition: The Art of Scientific Computing, 3 edn. Cambridge University Press, http://www.amazon.com/Numerical-Recipes-3rd-Scientific-Computing/dp/0521880688/ref=sr_1_1?ie=UTF8&s=books&qid=1280322496&sr=8-1

Ravanbakhsh S., Oliva J., Fromenteau S., Price L. C., Ho S., Schneider J., Poczos B., 2017, arXiv e-prints, p. arXiv:1711.02033

Ronneberger O., Fischer P., Brox T., 2015, arXiv e-prints, p. arXiv:1505.04597

Selig M., et al., 2013, aap, 554, A26

Seljak U., 1998, ApJ, 503, 492

Smith K. M., Zahn O., Doré O., 2007, Phys. Rev. D, 76, 043510

Smoot G. F., et al., 1992, ApJ, 396, L1

Zaroubi S., Hoffman Y., Fisher K. B., Lahav O., 1995, ApJ, 449, 446