Universidad Zaragoza

Facultad de Ciencias Universidad Zaragoza

Universidad de Zaragoza

FACULTAD DE CIENCIAS

INTELIGENCIA ARTIFICAL APLICADA AL TRATAMIENTO DE IMÁGENES GENERADAS POR MICROSCOPÍA DE TRANSMISIÓN ELECTRÓNICA

Trabajo de Fin de Grado – Grado en Física

Autor:

Jorge SIMÓN AZNAR

Directores:

Luis MARTÍN MORENO

Eduardo SÁNCHEZ BURILLO

Julio de 2023

Índice

1. Introducción

La Inteligencia Artificial (IA) es uno de los temas de los que más se ha hablado en los últimos años, especialmente este último tras la llegada de ChatGPT, una poderosa herramienta capaz de aprender estructuras del lenguaje humano y así proporcionar respuestas útiles generando contenido coherente en función de las preguntas y las instrucciones que recibe. ChatGPT pertenece al campo de la IA conococido como *Procesamiento del Lenguaje Natural* (NLP por sus siglas en inglés), el cual, además de permitir desarrollar modelos como ChatGPT, también tiene aplicaciones como generar subtítulos en tiempo real, traducir textos entre idiomas, etc. Por supuesto, hay muchas otras aplicaciones de la IA más allá del NLP, destacando entre otras el reconocimiento de imágenes, empleado por ejemplo por *Google Maps* o para la conducción autónoma de los vehículos más novedosos, así como el conocido *Big Data*.

Hace mucho tiempo que el ser humano empezó a soñar con la posibilidad de crear máquinas que fueran capaces de comportarse como personas, de pensar, de sentir... Inspirados por el complejo funcionamiento de las neuronas del cerebro humano en 1943 McCulloch y Pitts [1] desarrollaron el primer modelo de *Red Neuronal*, un clasificador binario capaz de reconocer dos categorías diferentes. El problema era que ciertos parámetros tenían que ser introducidos a mano por las personas. No fue hasta 1958 con la aparición del *Perceptrón* de Frank Rosenblatt [2] que se pudieron aprender dichos parámetros de forma automática. Sin embargo, se dieron cuenta de que este modelo no era más que un clasificador lineal y que no podía resolver problemas no lineales, además de que no tenían las herramientas ni potencia necesarias en sus ordenadores para crear redes neuronales tan profundas. Hubo que esperar hasta 1986 con la llegada del nuevo algoritmo *Backpropagation* [3] para que el campo de las Redes Neuronales (RRNN) resucitara.

De esta forma, con las investigaciones realizadas hasta la fecha y con la aparición en la última década del conocido *Deep Learning* es como hemos llegado hasta el punto en el que nos encontramos ahora mismo, capaces de utilizar la IA en un abanico sumamente amplio de áreas de conocimiento como son la medicina, las finanzas, el marketing, etc.

En este trabajo nos centraremos en el campo del *Reconocimiento de imágenes*, en concreto de imágenes generadas por microscopía de transmisión electrónica de barrido, haciendo uso de diferentes técnicas de IA. En cuanto a la estructura, este se compone de un resumen donde se presenta la motivación y metodología, una sección de introducción a los conceptos que se van a tratar, la exposición de los resultados obtenidos y, finalmente, un apartado conclusivo.

2. Resumen

Una de las aplicaciones más demandadas en estos tiempos es, sin duda alguna, el *Reconocimiento de Imágenes*, debido a su gran utilidad en diversas áreas de la ciencia y la tecnología. Con el desarrollo del *Deep Learning* [4] en las últimas décadas y la aparición de modelos como las *Redes Neuronales Convolucionales* (CNN) el avance en este campo ha sido exponencial, pudiendo aplicarlo en innumerables profesiones, como por ejemplo en medicina para la detección de aberraciones en pruebas como resonancias magnéticas o radiografías, así como la detección de tumores; en astronomía al analizar imágenes de telescopio; en astrofísica para la detección de partículas o incluso en situaciones más cotidianas como la automatización de un proceso industrial.

El objetivo principal del presente trabajo es la implementación de distintas técnicas con IA para el filtrado de ruido de imágenes generadas por *Microscopía electrónica de transmisión de barrido* (STEM) de una película delgada de un ferroeléctrico sobre un buffer conductor, cuya estructura cristalina posee dos orientaciones distintas. Por tanto, nuestros objetivos se centran en el filtrado del ruido, además del reconocimiento de inestabilidades en la muestra debidas a la deriva térmica, discontinuidades, movimientos oscilatorios, aberraciones del microscopio, etc. Y finalmente, la validación del modelo para ambas orientaciones cristalinas. Para ello han sido diseñados varios modelos de *Redes Neuronales* (RRNN) mediante el uso del lenguaje de programación *Python*, que permite beneficiarse de la plataforma *TensorFlow* con sus librerías de *Keras*, así como otras librerías que facilitan los calculos vectoriales como *NumPy* y la representación gráfica como *MatPlotLib*.

3. Conceptos básicos

Antes de empezar con el desarrollo del trabajo, es necesaria la introducción de conceptos básicos que serán clave para la elaboración y comprensión de este. En primer lugar, explicaré los conceptos relacionados con la estructura de las *RRNN*, desde las neuronas que la componen hasta parámetros y funciones a tener en cuenta para el correcto funcionamiento de estas, así como los modelos a implementar. Además, se mostrarán otras dos técnicas de filtrado: *Singular Value Decomposition* (SVD) [5] y *Transformada de Fourier* [6].

3.1. Redes Neuronales

Las *RRNN* [7][8] son una de las herramientas más poderosa de la *IA*, especialmente en los ámbitos del NLP y el reconocimiento de imágenes. Su nombre se debe a que la inspiración para llevar a cabo su desarrollo no fue otra que la red neuronal del tejido cerebral de los seres vivos. La motivación residía en que si este tipo de conexiones funciona en nosotros, ¿por qué no iba a funcionar en una máquina?

Por tanto, una red no es más que una sucesión de neuronas organizadas en capas, donde la primera capa o capa de entrada es la denominada *Input Layer*, la capa final o de salida *Output Layer* y las capas intermedias o capas ocultas *Hidden Layers*. Una red se puede componer de una o varias capas y son las redes con numerosas capas ocultas las que se denominan *Deep Learning*. Volviendo a la analogía con las neuronas de nuestro tejido cerebral y tejidos nerviosos, la forma en la que estas se relacionan consiste en lo siguiente: Las neuronas reciben un impulso electroquímico de otras neuronas a través de sus dendritas (*input)* y, este impulso, si tiene el "poder" necesario como para activar la neurona, será transmitido a otra a través del axón (*output*). Extrapolando este modelo a una red neuronal artificial, podemos pensar que nuestras redes deben tener una variable que mida el "poder" de la señal y otra que active la neurona si se supera el umbral. Es en este punto cuando Rosenblatt introduce los *weights* o pesos $w_1, w_2...$, números reales que indican la relevancia del *input* respecto del *output*. De esta forma, la información de salida de la neurona dependerá de la suma de los valores de entrada ponderados con los respectivos pesos $\sum_j w_j x_j$, a la que se le aplicará una función no lineal llamada *función de activación*, $f(z)$.

Si escribimos esto último de forma vectorial, dada una capa r, donde $r \in 1,...,R$ con R el número de capas de la red, y dada una neurona k, donde $k \in 1,...,N$ con N el número de neuronas de la capa r y considerando el *output* x_j^{r-1} de la capa anterior $r-1$, donde $j \in 1,...,M$ con M el número de neuronas de la capa anterior, entonces el cálculo realizado por dicha neurona será:

$$z_k^r = \sum_{j=1}^{M} w_{kj}^r x_j^{r-1} + b_k^r, \tag{3.1}$$

donde w_{kj}^r y b_k^r son los *weights* y *bias* respectivamente. Finalmente, el *output* de la neurona vendrá dado por $f(z_k^r)$. Si nos fijamos bien, podemos observar como se trata transformaciones lineales seguidas de transformaciones no lineales y son estos parámetros w_{kj}^r y b_k^r los que hay que entrenar para que la red aprenda, ya que el *output* final dependerá de todos los *weights* y *bias* de la red.

Al final, lo que hace la red es, dadas unas variables de entrada, generar una salida. El objetivo es que esta salida se parezca lo máximo posible a la variable objetivo del problema. Esto se consigue minimizando una función de coste (sección 3.1.2) con algún algoritmo de optimización (sección 3.1.3).

3.1.1. Función de activación

La función de activación [9] es la transformada no lineal mencionada en la sección previa. Esta se encarga de dar un valor u otro a nuestra señal de salida dependiendo del valor resultante de la transformación lineal dada en 3.1, es decir, la señal de entrada o *input*.

Existen numerosos tipos de función de activación y cada uno de ellos es más eficaz o conveniente dependiendo del problema. Las funciones más comunes son las funciones *sigmoide* y *ReLU*. Para entender de una forma más intuitiva el funcionamiento de dichas funciones de activación vamos a ver como se comporta la función escalón, usada en el algoritmo del *Perceptrón*:

$$f(z_k^r) = \begin{cases} 1 & \text{si } z_k^r > 0 \\ 0 & \text{en cualquier otro caso} \end{cases} \tag{3.2}$$

Como vemos se trata de una función muy sencilla, si la suma de los *inputs* ponderados con sus pesos es mayor que 0, la salida es 1, sí no es 0. Sin embargo, esta función no es eficaz para las redes actuales, ya que no es diferenciable, lo que supone un problema a la hora de aplicar descenso de gradiente, un término que veremos más adelante. Veamos qué forma tienen las funciones más empleadas y cómo se comportan para distintos valores de z:

- *Sigmoide* : la función sigmoide es una de las funciones más utilizadas, sobre todo en modelos en los que tenemos que predecir el resultado como una probabilidad o en clasificación binaria, ya que para valores negativos de z esta se aproxima a 0 y, a medida que z crece, el valor de la función se aproxima a 1.

$$f(z) = \frac{1}{1+e^{-z}} \tag{3.3}$$

- *ReLU* : la función *ReLU* (Rectified Lineal Unit) es la más empleada a día de hoy y se utiliza en casi todas las RRNN, implementándose en las capas ocultas, nunca en la capa final. El resultado de esta función es 0 cuando z es menor que 0 y $f(z)$ es igual a z cuando z es superior o igual que 0.

$$f(z) = \begin{cases} z & \text{si } z > 0 \\ 0 & \text{si } z < 0 \end{cases} \tag{3.4}$$

Aunque las funciones *Sigmoid* y *ReLu* resultan ser de las funciones más efectivas a la hora de entrenar redes, podemos encontrar situaciones en las que conviene utilizar otro tipo de funciones ya que puede aparecer un problema llamado *vanishing gradients*, el cual supone un estancamiento en la evolución de los pesos y bias debido al desvanecimiento del valor de las derivadas parciales durante el entrenamiento de redes con varias capas (notar que la *ReLU* tiene derivada nula para $x < 0$ y que en la *sigmoide* las derivadas se aproximan en seguida a 0 en cuanto $|x|$ crece un poco). Por lo que, en nuestro caso, se ha empleado en las capas ocultas la función *Leaky ReLU*, que es una mejora del valor predeterminado principal de *ReLU*, es decir, para valores negativos de z no es 0, si no que es una función lineal con pendiente entre 0 y 1, típicamente más cerca de 0 que de 1, lo que permite que no perdamos esa información y evitemos los *vanishing gradients*.

3.1.2. Función coste

La función de coste o también llamada *Loss* no es más que la diferencia o error entre los datos de salida de la red o predicciones y los datos de entrada. Es decir, es la función que evalúa la eficiencia de la red y que depende de todos los *weights* y *bias* de la red y que por tanto, buscaremos minimizar entrenando dichos parámetros. Como vemos, aunque su definición es sencilla, esta es de gran importancia a la hora de implementar un modelo.

Al igual que la función de activación, existen distintas funciones de coste y su utilización varía también dependiendo del problema. Las más comunes son *Categorical Cross-Entropy* , *Binary Cross-Entropy* y *Mean Squared Error* (MSE). En nuestro caso se ha empleado la función (MSE) ya que es una función sencilla y calcula el error cuadrático medio entre los valores reales y los predichos, por lo que es fácil de interpretar y correcta para nuestro problema.

$$C(w,b) = \frac{1}{N}\sum_{j}^{N}|x_j - y_j|^2, \tag{3.5}$$

donde N es el número de datos de entrada a la red y x_j e y_j representan las predicciones y los valores reales respectivamente.

3.1.3. Algoritmos de Optimización

Como hemos visto en el apartado anterior, el objetivo del entrenamiento de una red es disminuir la función coste, es decir, aumentar su acierto en las predicciones a base de modificar todos los *weights* y *bias* de la red. Pero, ¿cómo modificamos todos estos parámetros de manera que la red esté lo más optimizada posible? Podríamos pensar que una forma sería inicializar de forma aleatoria todos estos hasta que se de la combinación adecuada, sin embargo, si nos fijamos en la ecuación 3.5, nos damos cuenta de que, aunque se trata de una función sencilla, extrapolada a todas las neuronas de todas las capas de una red se vuelve un problema complejo de optimizar debido a su gran número de parámetros. Por tanto, en vez de dejarlo en manos del azar, lo que se usa es un algoritmo que cambia directamente el valor de cada uno de estos parámetros, paso a paso. A este algoritmo se le conoce como *Descenso de gradiente* [10] y es uno de los más importantes en la creación de RRNN modernas.

Entonces, el algoritmo *Descenso de gradiente* lo que hace es iterar y desplazarse por el espacio multidimensional de *weights* y *bias* hasta que encuentra un mínimo. Y para ello lo que hace es calcular el gradiente de la función coste o *Loss*, que es la que queremos minimizar, y se desplaza dando pequeños saltos en contra de este ya que el gradiente indica la dirección de máxima variación. Así, sean el peso y *bias* j y la iteración i tenemos:

$$w_j^{i+1} = w_j^i - \alpha\frac{\partial C}{\partial w_j}, \tag{3.6}$$

$$b_j^{i+1} = b_j^i - \alpha\frac{\partial C}{\partial b_j}, \tag{3.7}$$

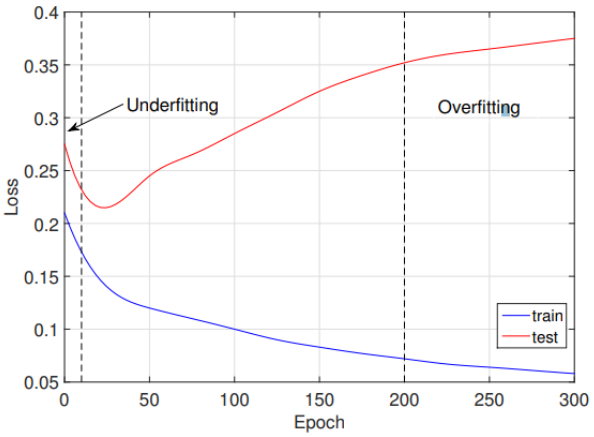
donde α es el llamado *learning rate*, un hiperparámetro de la red que controla el tamaño del salto. Si este es muy grande la red puede ser que aprenda más rápido pero también puede ser que al iterar se salga de la cuenca de un mínimo y que la red acabe no aprendiendo, sin embargo, si este es muy pequeño el proceso será muy lento y supondrá un coste computacional muy elevado.

Sin embargo, si observamos el esquema de una red neuronal nos damos cuenta de que el cambiar uno de los parámetros afecta a los de las siguientes capas, ya que las operaciones que se producen en las neuronas dependen de estos, lo que complica el proceso de optimización. Para solucionar este problema y calcular de forma efectiva el gradiente de la función de coste se utiliza el algoritmo *Backpropagation* [3], sin duda alguna, uno de los algoritmos más importantes y por el cual la IA que conocemos es como es hoy en día. Este algoritmo calcula todas las derivadas parciales de la función de coste desde la capa del *output* hasta la capa *input* de la red mediante la regla de la cadena, lo que nos permitirá, en primer lugar, calcular los parámetros de activación de cada capa de la red (*weights y bias*) y en segundo lugar, recorrerla en sentido inverso para calcular el gradiente de la función *Loss*.

Aunque el *Descenso de gradiente* presenta gran efectividad en redes con *datasets* pequeños, cuando tratamos de entrenar grandes cantidades de datos este resulta extremadamente lento. Por ello se han diseñado otros algoritmos, como el *Stochastic Gradient Descent* (SGD) [10][11], que supone que el gradiente de un subconjunto (*mini batches*) de los datos de entrada es semejante al de todos los datos. De esta forma se introduce un nuevo hiperparámetro denominado *mini batch size*, el cual determina el tamaño del subconjunto de datos escogidos al azar. Cabe destacar que, aunque el SGD es uno de los algoritmos más eficientes y empleados hoy en día, existen otros que pueden ser más efectivos dependiendo del problema, como por ejemplo *Adam* [10][12] y *RMSProp* [10][13], los cuales hemos utilizado en el desarrollo del trabajo.

3.1.4. Underfitting y Overfitting

Los conceptos de *underfitting* y *overfitting* [14] son unos de los problemas más comunes a la hora de entrenar redes neuronales, ya que la elección de los hiperparámetros como el número de épocas (*epochs*), el *learning rate*, el *mini batch size*, el número de neuronas y el número de capas es una elección que depende del programador de la red puesto que el valor de estos no se rige por ninguna regla. Por ello, es importante saber interpretar los resultados e identificar si en el entrenamiento de nuestra red se está dando uno de estos dos conceptos.



El *underfitting* se da cuando el modelo no puede alcanzar una función de *loss* lo suficientemente baja en el entrenamiento con los datos que denominamos *training set*. Es decir, la red falla durante su aprendizaje y no es capaz de reconocer los patrones de dichos datos. Por otro lado tenemos el *overfitting*, que aparece cuando el modelo aprende demasiado bien los patrones de los datos de entrenamiento pero no es capaz de reconocer la información subyacente a estos, por lo que falla prediciendo otros datos, obteniendo así una función coste baja para el entrenamiento pero elevada para otros datos. El *underfitting* es fácil de identificar, ya que se suele obtener una función coste o pérdida elevadas, sin embargo, para identificar el *overfitting* hacemos uso de otro set de datos, al que llamamos *validation set*. Existen numerosos métodos para solucionar estos problemas. El más sencillo de todos es añadir un mayor tamaño de *dataset*, sin embargo, esto puede aumentar notablemente el tiempo de entrenamiento, por lo que no siem-

pre es recomendable. Otro método es el *early stopping*, el cual consiste en pausar el entrenamiento de la red en un punto determinado. Para ello, se programa para que se detenga cada ciertas *epochs* y así poder observar los parámetros *training loss* y *validation loss*. Finalmente, la técnica que hemos empleado en este trabajo es la de *DropOut*, cuyo principio es "desconectar" algunas neuronas de la capa que elijamos. De esta forma se consigue que la red aprenda aquella información más robusta de los datos y deje de lado la información innecesaria, como el ruido, algo que en nuestro trabajo resulta de gran importancia.

3.2. Redes densas y convolucionales

En las redes tradicionales basadas en el concepto de *feedforward* [15], cada neurona de cada capa está conectada a todas las neuronas de la capa siguiente, por lo que dichas neuronas toman la salida de cada una de las neuronas de la capa anterior, realizan las transformaciones lineal (ec. 3.1) y no lineal (en caso de activación sigmoide ec. 3.3) y emiten la señal a cada una de las neuronas de la siguiente capa, como sucede en el esquema de la derecha. A este tipo de red es a la que denominamos red neuronal *densa*.

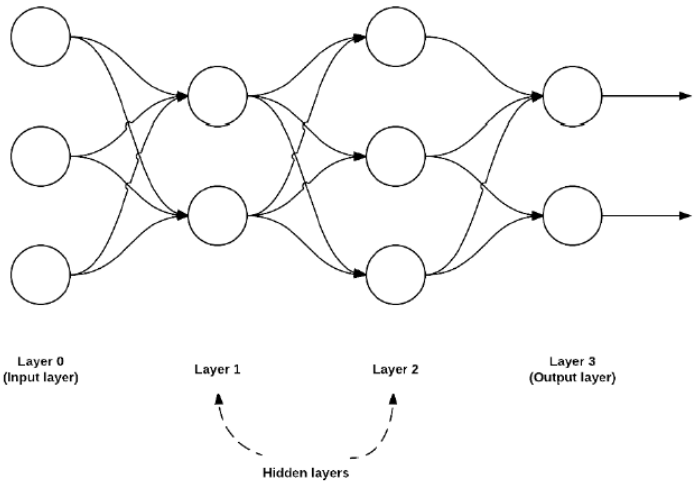


Figura 3.1: Estructura de red densa. Fuente: [8]

Mientras que las CNN (*Convolutional Neural Network*) [16][17] no utilizan capas densas hasta la última (de manera opcional), como podemos observar en la figura 3.2, sino que utilizan capas convolucionales que aplican operaciones de convolución a las entradas, permitiéndoles extraer características locales y adquirir representaciones jerárquicas de las imágenes. Las capas convolucionales se componen de filtros o *kernels*, pequeñas regiones de la matriz original que se desplazan sobre esta realizando operaciones matemáticas locales para generar mapas de características que se fusionan en capas subsiguientes para obtener representaciones más abstractas de la imagen.

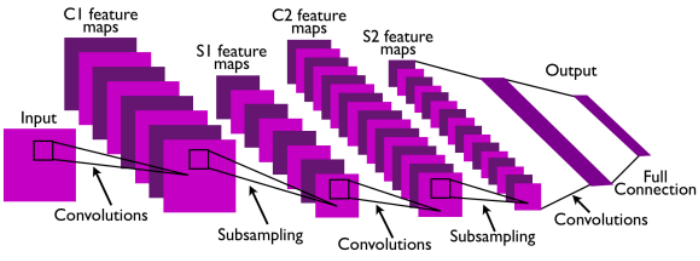


Figura 3.2: Estructura de red convolucional. Fuente: [17]

En concreto, en cada capa se aplica un tipo de filtro y son estos los que, durante el entrenamiento de la red, deben aprender a detectar bordes, utilizar estos bordes para identificar formas y progresivamente ir aumentando la complejidad hasta llegar a detectar características y expresiones de más alto nivel, como la forma de una cara, un ojo, una sonrisa, etc. Las principales ventajas que nos ofrece este tipo de red son la invarianza local y la composicionalidad. La primera de estas nos permite clasificar una imagen dependiendo de qué objeto

aparece en ella sin importar en qué posición de la imagen se encuentre, mientras que la segunda hace referencia a la capacidad de la red para profundizar y aprender características complejas a partir de unas de más bajo nivel, como son los píxeles.

Ambas redes resultan de gran utilidad y su efectividad, al igual que los hiperparámetros que escojamos, dependerá del tipo de problema al que nos enfrentemos y de lo que sea más adecuado, si aprender patrones globales como hacen las redes *densas* o aprender patrones locales como sucede con las redes *convolucionales*.

En nuestro caso hemos aplicado ambos tipos de red y estudiaremos los resultados para ver cuál de ellas funciona mejor para el filtrado del ruido de nuestras imágenes.

3.3. Autoencoder

Un *autoencoder* [18] [19] es una red neuronal simétrica basada en el aprendizaje no supervisado y que se compone de tres partes, *encoder* $\mathbb{R}^n \longrightarrow \mathbb{R}^m$, *decoder* $\mathbb{R}^m \longrightarrow \mathbb{R}^n$ y una representación latente de las características de los datos, que sería la capa intermedia de la red. Puesto que su estructura es la unión de dos redes, esta se entrena de la misma forma, aplicando los mismos algoritmos de optimización, las mismas funciones de coste y de activación. La función principal de este tipo de red es representar el *input* de la manera más fiel y precisa posible, almacenando de alguna manera los rasgos principales en el espacio latente.

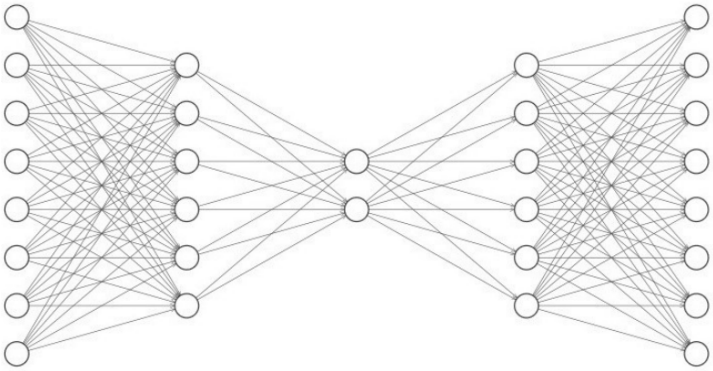


Figura 3.3: Estructura de la red autoencoder (Fuente: [20])

Aunque pueda parecer de poca utilidad ya que su función es reconstruir un objeto que en un principio ya tenemos, en realidad se trata de una red con gran efectividad en determinados problemas, como por ejemplo la reducción de la dimensionalidad; la clasificación de imágenes, llegando a reducir el tiempo de clasificación en un factor 1000 sacrificando un 7.4 % de acierto en el ejemplo del MNIST; la detección de anomalías y, finalmente, el ejemplo que hemos tratado en este trabajo, la reducción de ruido.

3.4. Transformada de Fourier

La *Transformada de Fourier* es una de las técnicas matemáticas más conocidas y empleadas en el mundo de la física y la ingeniería ya que nos permite conocer con mayor detalle algunas de las características de un amplio tipo de señales. Se trata de la transformación de una señal en el dominio temporal o espacial al dominio frecuencial. Es en este dominio donde hallamos un conjunto de amplitudes comple-

jas, denominadas coeficientes de las Series de Fourier. De esta forma, una señal unidimensional se puede descomponer en un conjunto infinito de señales de tipo seno con diferentes frecuencias, amplitudes y fases.

Como hemos mencionado antes, su uso se extiende a lo largo de un gran número de problemas, sin embargo, nosotros nos centraremos en el filtrado del ruido de una señal [6]. Cuando se aplica la *Transformada de Fourier* a una señal contaminada con ruido se obtiene un espectro que muestra las amplitudes y fases de las diferentes frecuencias presentes en la señal, donde encontraremos también las del ruido. Por tanto, para eliminarlo podemos realizar un filtrado en el dominio de la frecuencia. Esto implica identificar las componentes de frecuencia correspondientes al ruido y atenuarlas o eliminarlas por completo, mientras se conservan las componentes de frecuencia de interés. Para visualizar su uso hemos desarrollado un sencillo ejemplo. Supongamos una función gaussiana del tipo $e^{-\frac{1}{2}x^2}$, donde x tomará 1000 valores entre -10 y 10 de forma equiespaciada. A la función le añadimos ruido aleatorio a partir de una distribución normal con media 0 y desviación estándar 0.1. Así, si aplicamos la *Transformada de Fourier* a la señal contaminada, localizamos la frecuencia de corte a partir de la cual atenuaremos la señal y diseñamos un filtro que lleve a 0 la amplitud de la señal a partir de dicha frecuencia, conseguimos recuperar la señal original a partir de la señal contaminada como se muestra en la figura:

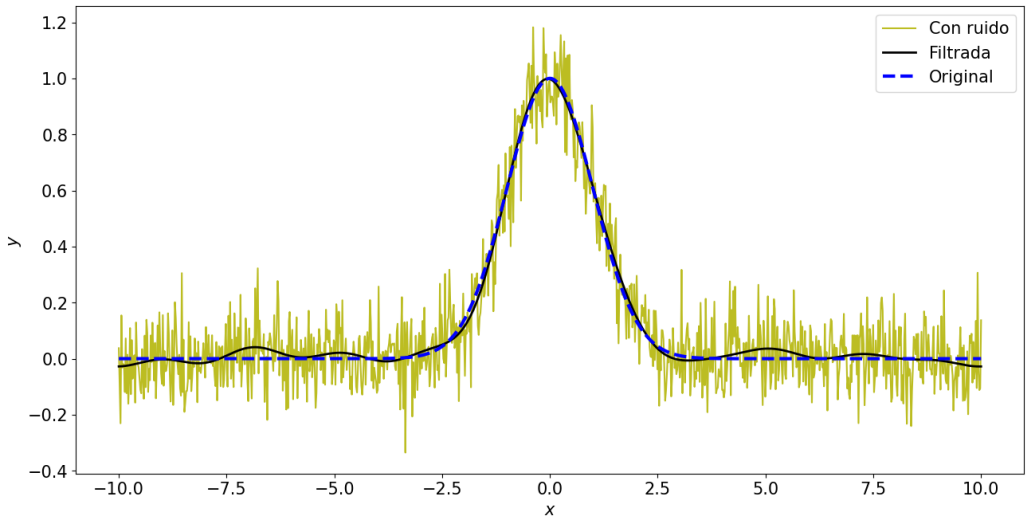
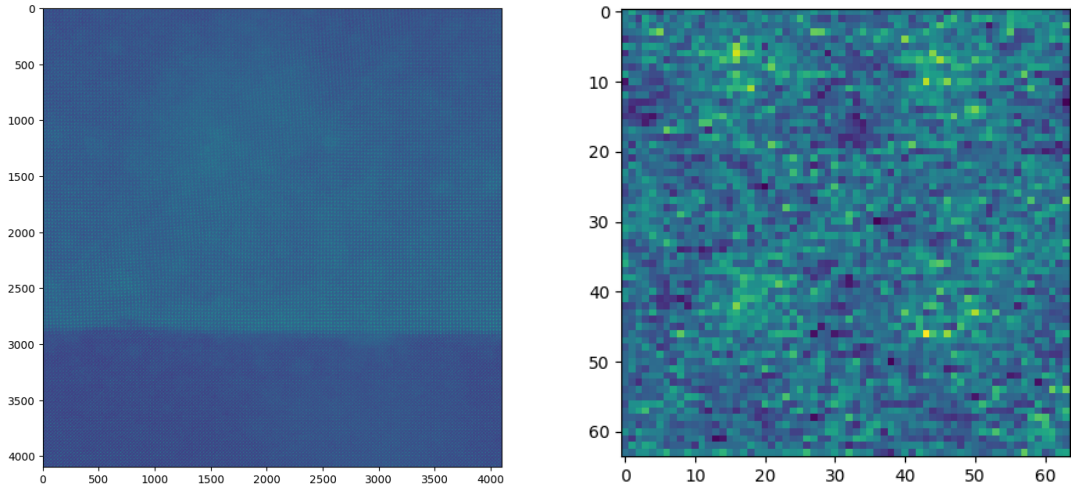


Figura 3.4: Filtrado del ruido de una gaussiana $e^{-\frac{1}{2}x^2}$ corrupta con frecuencia de corte 0.011 mediante la *Transformada de Fourier*

Podemos observar como la señal filtrada no es exactamente la señal original, sino que todavía encontramos cierto ruido en ella. Eso sucede porque hemos aplicado un filtro sencillo colocado “a ojo”, por lo que es posible que hayamos dejado algo de ruido o incluso que hayamos cortado antes de tiempo, por lo que podríamos estar perdiendo información de la señal original. Sin embargo, se aprecia notablemente como la mayor parte del ruido ha sido eliminado, recuperando así con bastante éxito la función gaussiana.

4. Resultados

En esta sección expondremos los resultados obtenidos mediante el uso de las técnicas introducidas. Como ya hemos comentado en apartados anteriores, el problema se centraba en filtrar el ruido de imágenes generadas por *Microscopía electrónica de transmisión de barrido* (STEM) de una película delgada de un ferroeléctrico sobre un buffer conductor para así ser capaces de detectar defectos en la muestra. Para poder llevar a cabo el trabajo, se nos proporcionó un conjunto de imágenes con las características ya mencionadas, unas de ellas con un tamaño de 9 pm por pixel y otras de 12 pm. Además, estas imágenes venían también diferenciadas dependiendo de la orientación de la estructura cristalina. Debido al gran número de imágenes y, por tanto, de datos, decidimos trabajar con las imágenes de 12 pm por pixel, que a su vez tenían un tamaño de 4096x4096 píxeles ya que fraccionando tan solo estas imágenes podíamos obtener un número de datos suficiente para entrenar nuestras redes. A continuación, se muestra la primera de las imágenes para que así se aprecie la magnitud de los resultados obtenidos.



(a) Imagen original de 4096x4096 (b) Recorte de la imagen original de 64x64

Figura 4.1: Imagen generada por STEM

4.1. Singular Value Decomposition (SVD)

Comenzamos el trabajo mediante el filtrado de las imágenes con la técnica *Singular Value Decomposition* [5], ya que este era el método utilizado por los investigadores que nos proporcionaron las imágenes. Se trata de una técnica sencilla cuyo fundamento se basa en la descomposición de una matriz (la imagen en nuestro caso) en sus componentes principales, de manera que

$$A = U\Sigma V^T \tag{4.1}$$

Siendo A la matriz de píxeles de nuestra imagen, U y V matrices unitarias que representan las bases de nuestra matriz A y Σ una matriz diagonal semidefinida positiva cuyas componentes son los valores singulares, por lo que serán no negativos. Estos valores singulares de la matriz diagonal se encuentran ordenados de mayor a menor en la diagonal principal, por lo que representan la importancia relativa de cada elemento de la base.

Se trata de una técnica muy empleada en la reducción de la dimensionalidad, sin embargo, también es útil para la reducción del ruido ya que las componentes principales , es decir los valores singulares más grandes, son aquellas que contienen la información más importante, mientras que los valores singulares más pequeños serán aquellos que contengan la información innecesaria, como por ejemplo el ruido. El problema reside en que nuestra matriz diagonal tiene dimensiones de 4096x4096, es decir, 4096 valores singulares, por lo que no podemos saber a simple vista a partir de qué valor empieza a ser prescindible la información. Para hacernos una idea de la importancia de estos valores tendremos que hacer uso de la *Varianza Explicada*, ya que esta se refiere a la proporción de la varianza que contiene cada valor singular respecto de la varianza total, es decir, a mayor varianza explicada, mayor será la influencia de dicha componente. Por tanto, haciendo el siguiente cálculo:

$$Expl.Var_i = \frac{\sigma_i^2}{\sum_n \sigma_n^2} \tag{4.2}$$

Donde σ son los valores singulares, obtenemos la varianza explicada del i-ésimo valor singular. De esta forma podemos estimar su importancia:

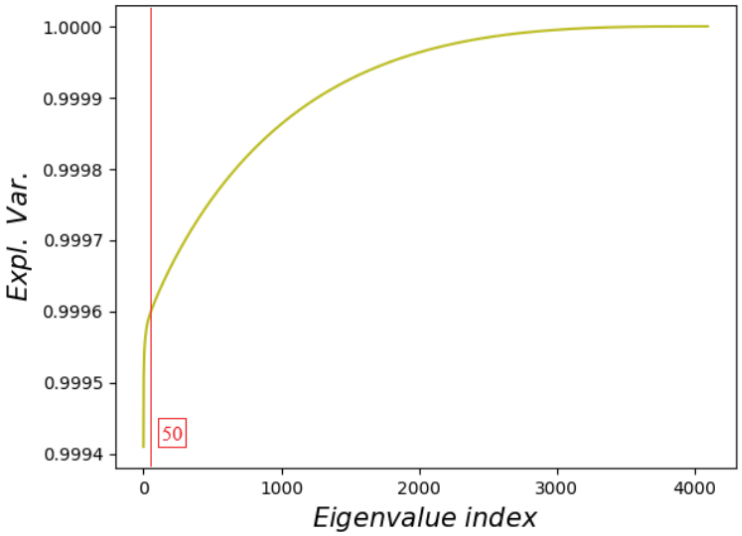
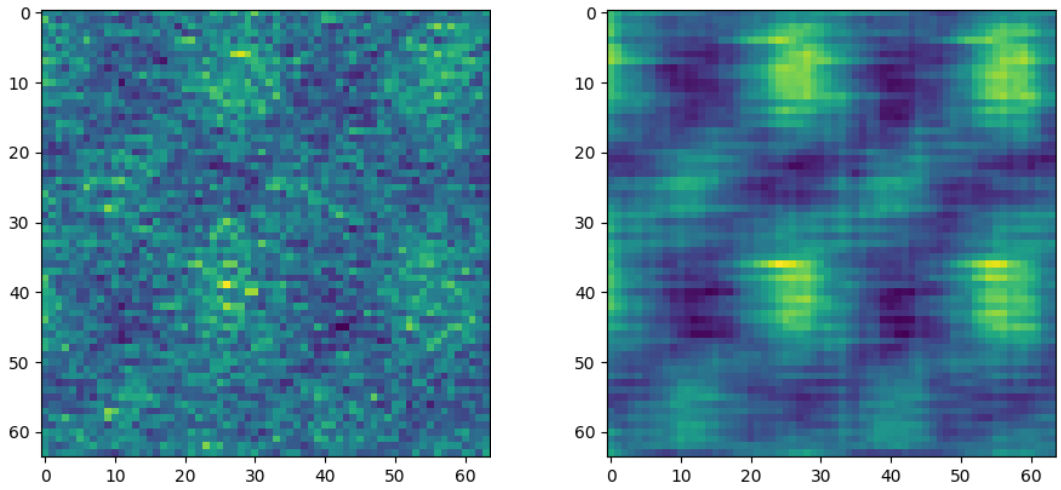


Figura 4.2: Varianza explicada de los valores singulares

Se observa como la mayor parte de la varianza explicada (99.96 %) se encuentra en los 50 primeros valores singulares. Además, a partir de este se produce un cambio en la forma de la función, por lo que supondremos que es en este punto en el que empieza a tomar importancia el ruido. De esta forma, aplicando el método SVD cogiendo sólo los 50 primeros valores obtenemos el siguiente resultado:



(a) Original (b) Filtrada

Figura 4.3: Reducción de ruido mediante SVD con 50 primeros valores singulares

Se aprecia una gran reducción del ruido, permitiéndonos diferenciar los distintos átomos de la lámina. Sin embargo, todavía existe la presencia del ruido en la imagen y, aunque si que se puede localizar la posición de los átomos, no permitiría distinguir apenas los dos tipos.

4.2. Dense Autoencoder

Como hemos introducido anteriormente, un *autoencoder* es un tipo de red neuronal cuya entrada y salida son iguales, por lo que tiene como objetivo reconstruir lo más fielmente posible los datos de entrada. Dichas redes pueden ser utilizadas para el filtrado de imágenes [21], ya que lo que permite dicha función es la estructura de *encoder*, *espacio latente* y *decoder*.

En este primer apartado veremos los resultados obtenidos a partir de los distintos modelos de *Dense Autoencoder*. En este caso, los datos de entrenamiento, validación y test han consistido en un conjunto de recortes de 64x64 píxeles de la imagen presentada anteriormente, en concreto, 4096 recortes. Los hiperparámetros escogidos para los distintos modelos implementados son los siguientes:

Hiperparámetro	**Model 1**	**Model 2**
epochs	300	
mini batch	10	
dataset (tr, val, test)	3645, 410, 41	
nºneuronas / capa	(4096,100,50,15,50,100,4096)	(4096,100,50,25,15,25,50,100,4096)
optimizador	Adam	
learning rate	0.0001	
f. activación	ReLU	
loss	MSE	
métrica	RMSE	

Tabla 4.1: Hiperparámetros de los modelos 1 y 2

Hiperparámetro	**Model 3**	**Model 4**
epochs	300	
mini batch	10	
dataset (tr, val, test)	3645, 410, 41	
nºneuronas / capa	(4096, 100, 50, 15, 50, 100, 4096)	
optimizador	Adam	RMSProp
learning rate	0.0001	
f. activación	Leaky ReLU	
loss	MSE	
métrica	RMSE	

Tabla 4.2: Hiperparámetros de los modelos 3 y 4

Se puede observar como los modelos se parecen en gran medida, cambiando en cada uno de ellos uno o dos hiperparámetros. Es necesario comentar que para todos ellos se ha empleado la función de activación *sigmoide* (ec. 3.3) en la última capa de la red. Con todos estos valores y las 3645 imágenes usadas para entrenar las redes se obtienen un total de 835111 parámetros, es decir, el número de *weights* y *bias* que queremos encontrar. Así, una vez entrenadas las redes, obtenemos los siguientes resultados:

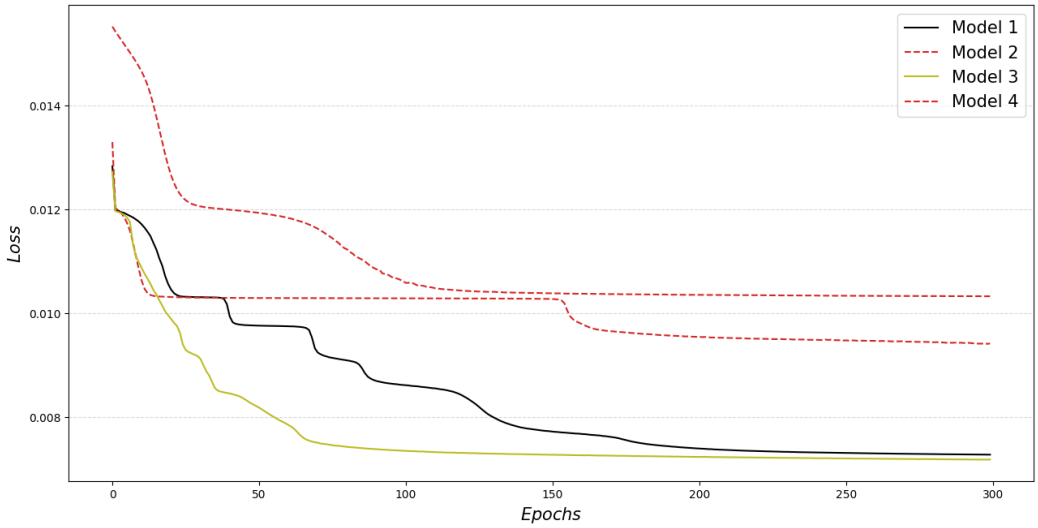


Figura 4.4: Comparación de la función *loss* de los modelos 1, 2, 3 y 4.

Observamos como los modelos 2 y 4 no consiguen reducir la función de coste tanto como los modelos 1 y 3, por lo que a priori cabe esperar que los resultados serán peores. En cuanto a los modelos 1 y 3, se aprecia una evolución correcta, por lo que si que parece que hayan entrenado bien las redes, notando una ligera mejora al aplicar la función de activación *Leaky ReLU* en vez de *ReLU*, hecho que quizá se deba a que, como mencionamos en la introducción teórica, esta función no trunca los valores negativos de z (3.4), sino que toma valores muy pequeños.

Una vez analizados estos resultados, podemos pensar que el modelo 3 es el más óptimo para el desarrollo del trabajo, sin embargo, al comparar la función de coste del set de *training* con el set de *validation* observamos como a partir de la época 70 se empieza a producir cierto *overfitting* (ver panel izquierdo de 4.5). Aunque este no es tan apreciable como el visto en la introducción, es conveniente eliminarlo. Para ello hemos aplicado un *DropOut* del 20 % tras la primera capa, es decir, por cada época

se desconecta un 20 % de las neuronas de la primera capa de manera aleatoria, obteniendo el siguiente resultado:

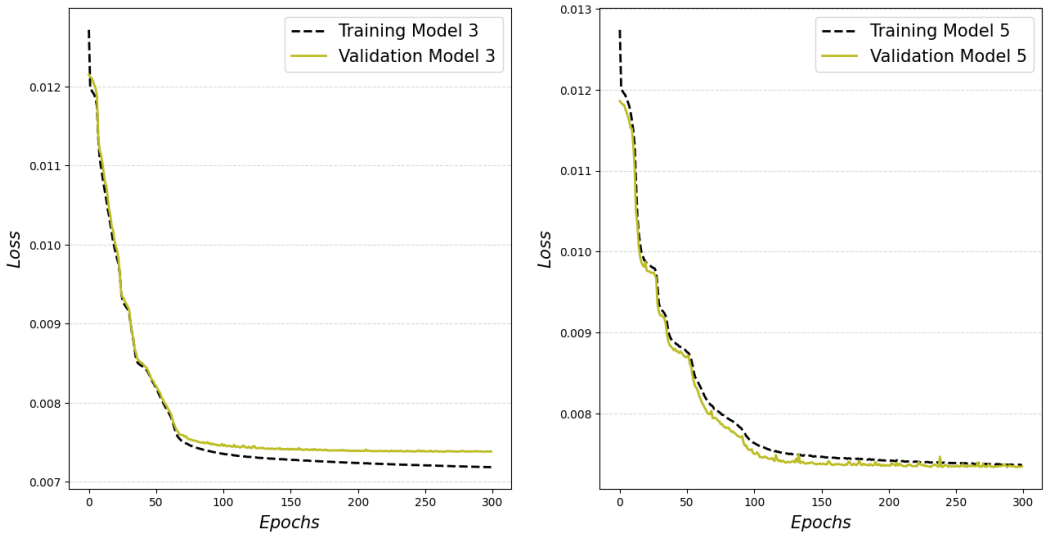


Figura 4.5: Efecto del DropOut sobre el modelo 3 (renombrado como Modelo 5).

Como vemos, hemos conseguido eliminar el efecto del *overfitting*. Por tanto, una vez estudiado el comportamiento de la función de *loss*, veamos como de eficaces son los modelos a la hora de filtrar ruido:

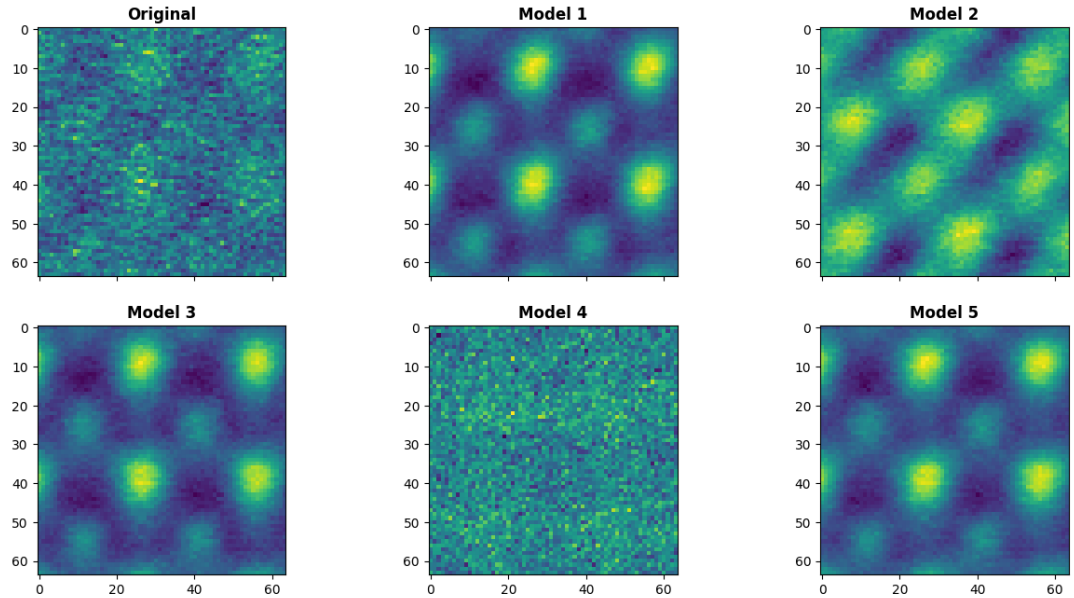
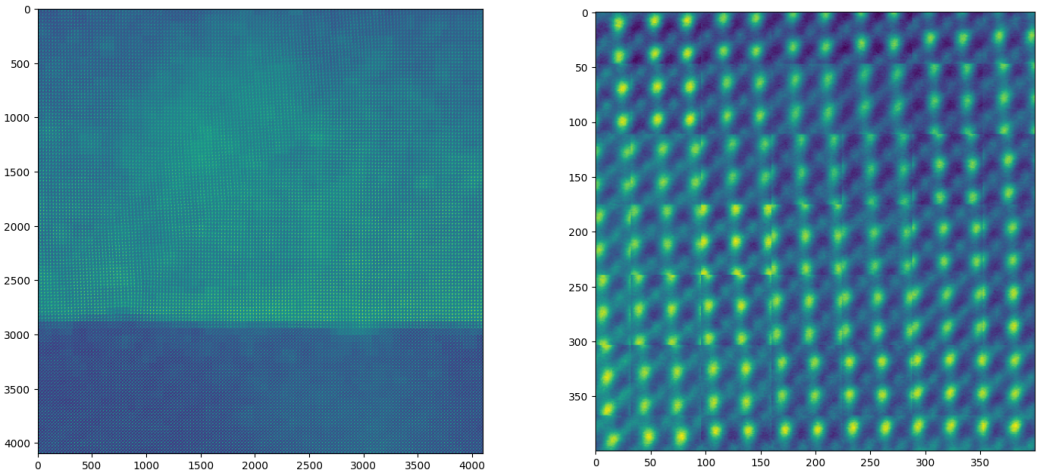


Figura 4.6: Comparación del filtrado de ruido de los distintos modelos implementados.

Los modelos 2 y 4, como era de esperar, no han conseguido extraer la información subyacente de la imagen original, por lo que no han sido capaces de eliminar el ruido correctamente. En cuanto a los modelos 1, 3 y 5, la diferencia es inapreciable. Los tres modelos han conseguido disminuir la función de coste al mismo valor prácticamente y han conseguido recuperar la imagen original de la misma manera. Por lo que, ya que el modelo 5 es el que presenta la capa de *DropOut* para evitar el *overfitting*, será el que usemos para reconstruir la imagen original 4.1a.

Sin embargo, encontramos un problema en nuestros resultados. Y es que, como se aprecia en la figura 4.7b, podemos observar una cuadrícula, como si fuera la "costura" de la reconstrucción de la imagen a partir de los recortes de 64x64 píxeles. Este fenómeno, el cual no se tuvo en cuenta a la hora de preparar el *dataset*, se debe a que los datos que se le introducen a la red son los recortes realizados de manera consecutiva, es decir, donde termina uno empieza el siguiente, por lo que la red aprende de cada una de las imágenes por separado pudiendo interpretar las intensidades de los extremos de cada recorte ligeramente diferentes, produciendo este efecto.



(a) 4096x4096 (b) 400x400

Figura 4.7: Imagen filtrada por modelo 5

Para solucionar el problema hemos tenido que entrenar la red con unos nuevos parámetros. En concreto, hemos realizado 50000 recortes, también de 64x64 píxeles, de manera aleatoria. De esta forma, la cantidad de datos de entrenamiento aumenta considerablemente, de 3645 a 44500, por lo que no será necesaria una capa de *DropOut* ya que otra forma de solucionar el *overfitting* es aumentando los datos de entrenamiento. Finalmente, puesto que el número de datos es mucho mayor, tendremos que reducir el número de épocas, en nuestro caso a 100. A dicho nuevo modelo, de ahora en adelante, lo denominaremos modelo *random*.

Aplicando estos cambios en el entrenamiento de nuestra red, obtenemos los siguientes resultados:

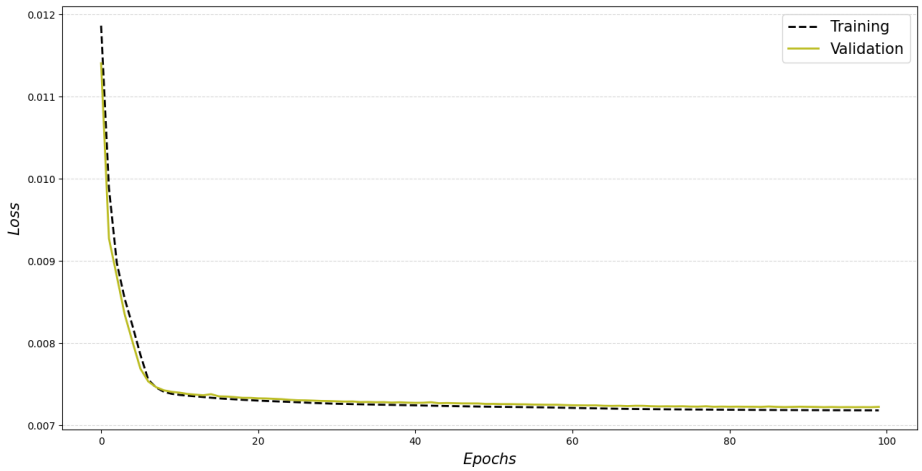
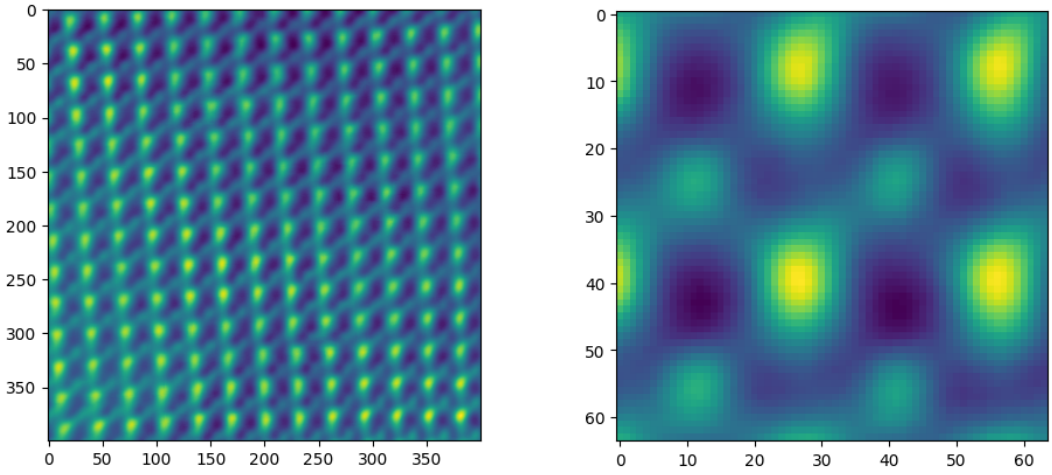


Figura 4.8: Evolución de la función *loss* en el modelo *random*.



(a) 400x400 (b) 64x64

Figura 4.9: Imagen filtrada por el modelo *random*

En la figura 4.8 vemos como no solo no se produce *overfitting*, sino que se alcanzan unos valores de la función de coste más bajos que mediante los modelos anteriores, lo cual es un indicativo de que la red es más efectiva. Ahora, fijándonos en la figura 4.9 observamos como, además de eliminar las marcas de "costura" al reconstruir la imagen principal, hemos conseguido eliminar en mayor cantidad el ruido, produciendo una suavidad en los contornos de los átomos que no encontrábamos en las imágenes filtradas por los otros modelos.

4.3. Redes Neuronales Convolucionales

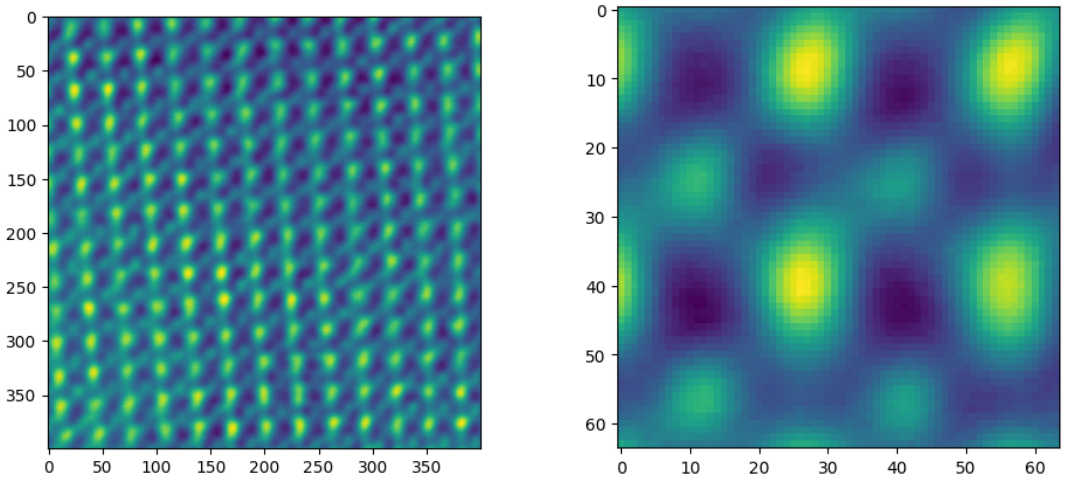
Como hemos introducido anteriormente, además de las redes neuronales *densas*, en nuestro caso aplicadas con la estructura de *autoencoder*, también hemos empleado redes neuronales *convolucionales* [22]. En este apartado presentaremos los resultados obtenidos mediante el uso de esta técnica, así como los hiperparámetros escogidos para ello.

Tras observar que, para el modelo de *autoencoder denso*, la mejor forma de preparar los datos de entrenamiento era hacer recortes de la imagen original 4.1a de manera aleatoria ya que se obtenía un mejor resultado, decidimos utilizar el mismo *training set* para nuestra red convolucional. Por tanto, disponemos de 50000 imágenes de dimensión 64x64 píxeles ordenadas de forma aleatoria. Además, los hiperparámetros utilizados son los mismos que para el modelo 3 de red densa presentados en la tabla 4.2, a excepción del número de épocas y el batch size que, debido al aumento de datos de entrenamiento y al cambio en la estructura de la red, hemos modificado a 10 y 128 respectivamente. Y al igual que estos modelos, la función *sigmoide* (ec. 3.3) ha sido utilizada en al capa densa. En cuanto a la estructura de la red, esta es la siguiente:

Input	(64, 64, 1)
Conv2D	(64, 64, 8)
MaxPooling2D	(32, 32, 8)
Conv2D	(32, 32, 16)
MaxPooling2D	(16, 16, 16)
Conv2D	(16, 16, 32)
MaxPooling2D	(8, 8, 32)
Conv2D	(8, 8, 64)
Flatten	(4096)
Dense	(4096)
Output	(64, 64, 1)

Tabla 4.3: Estructura *red neuronal convolucional*

Con la estructura ya presentada, procedemos a analizar los resultados:



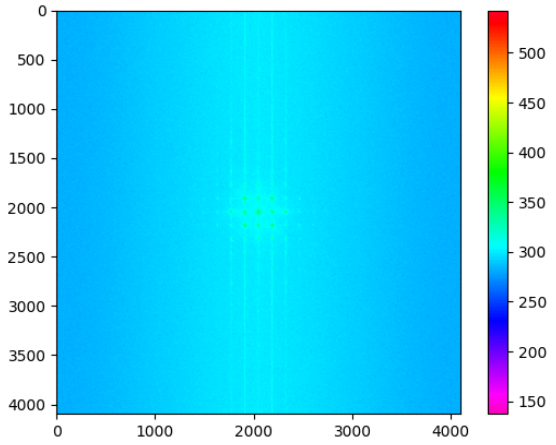
(a) 400x400 (b) 64x64

Figura 4.10: Imagen filtrada por el modelo *convolucional random*

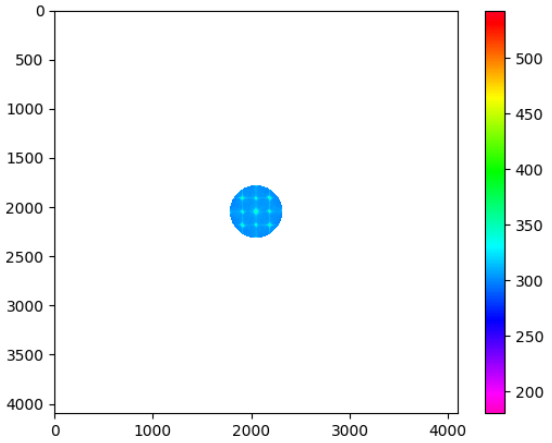
Observamos como los resultados son muy semejantes a los obtenidos mediante el modelo de autoencoder *dense random*, siendo la diferencia entre estos prácticamente inapreciable.

4.4. Filtrado de ruido mediante la Transformada de Fourier

Finalmente, veremos como la *Transformada de Fourier* puede ser utilizada para el filtrado de nuestras imágenes de un modo similar al mostrado en el ejemplo 3.4. En este caso, nuestra *transformada de Fourier* será una transformada bidimensional, por lo que la función en el espacio de frecuencias dependerá de dos variables, K_x y K_y. Para calcular la transformada de la matriz de píxeles que representa la imagen hemos empleado la función proporcionada por la librería *numpy*, *numpy.fft.fft2*. Una vez hallada dicha transformada, en primer lugar hemos centrado las frecuencias, es decir, las frecuencias con mayor amplitud han sido colocadas en el centro mediante la función *numpy.fft.fftshift*. De esta forma nos resulta más sencillo aplicar un filtro. Por tanto, una vez tenemos la transformada centrada calculamos la magnitud para así poder visualizarla y hacernos una idea de cómo debe ser el filtro:

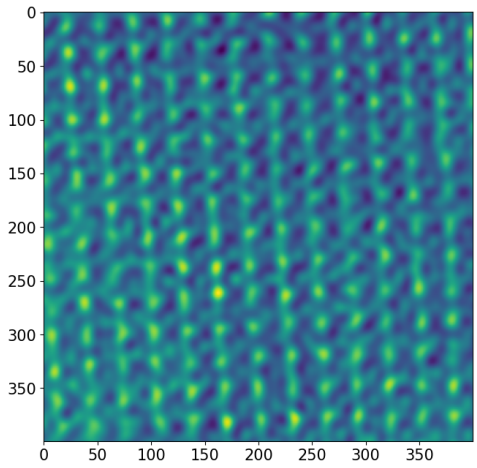


(a) Magnitud de la Transformada de Fourier

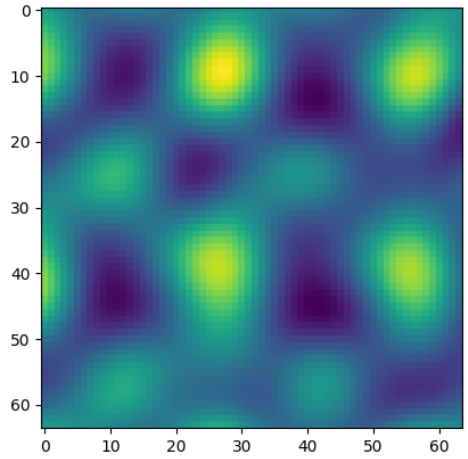


(b) Filtro Paso-Bajo sobre la magnitud de la TF

Como vemos, las componentes frecuenciales con mayor amplitud se encuentran en el centro, aquellas que tienen un contenido frecuencial más significativo y que corresponden a la estructura espacial de la red cristalina, por lo que aplicando un filtro circular que deje pasar las frecuencias que se encuentran dentro de él deberá bastar para eliminar las frecuencias más altas, en este caso las del ruido. El filtro diseñado tiene unas dimensiones de 280 píxeles de radio, aproximadamente lo que se aprecia a simple vista que abarcan las frecuencias centrales. Recomponiendo la imagen tras haber aplicado el filtro obtenemos los siguientes resultados:



(a) 400x400



(b) 64x64

Figura 4.12: Imagen filtrada mediante la *Transformada de Fourier*

Se observa como, efectivamente, el método de la *Transformada de Fourier* es eficaz para el filtrado de ruido. De hecho, fijándonos en la figura 4.12b se puede apreciar una suavidad en los contornos similar a la generada por el modelo *random* del autoencoder *dense*, así como de la red *convolucional*. Sin embargo, las formas presentan cierta distorsión, haciéndose más apreciable al mirar un mayor número de átomos, como en el caso de la figura 4.12a.

4.5. Comparación de resultados

A lo largo de esta sección hemos ido viendo los resultados que hemos generado mediante el uso de las diferentes técnicas; SVD, autoencoder denso, red neuronal convolucional y transformada de Fourier.

Todos estos métodos pueden ser empleados para el filtrado de ruido o *denoising*, sin embargo, hemos podido ver como algunos resultan ser más eficaces que otros. En este apartado vamos a comparar los distintos modelos, observando recortes aleatorios de la imagen filtrada y calculando una métrica denominada *Peak Signal to Noise Ratio* (PSNR) [23] para cuantificar de alguna manera el ruido eliminado por cada método.

El término PSNR es una expresión que relaciona el valor máximo que puede tomar una imagen con la distorsión producida por el ruido que afecta a la calidad de esta. Puesto que el rango dinámico de valores de una imagen puede ser muy amplio, normalmente se expresa el valor en decibelios (dB), significando que a mayor valor de dB, mayor eliminación de ruido, por lo que la expresión para calcularlo es

$$PSNR = 10log_{10}(\frac{max^2}{MSE}) \tag{4.3}$$

$$MSE = \frac{1}{NM}\sum_{0}^{N-1}\sum_{0}^{M-1}||f_{i,j} - g_{i,j}||^2 \tag{4.4}$$

donde *max* es el valor máximo de la imagen sin ruido, en nuestro caso la imagen filtrada, N y M son las dimensiones de nuestras imágenes (NxM) y $f_{i,j}$ y $g_{i,j}$ son los valores de los píxeles que componen las matrices que representan la imágenes filtrada y original respectivamente. Es esta naturaleza logarítmica la que nos permite diferenciar imágenes que varían ligeramente. Ahora veamos si este término coincide con nuestra percepción:

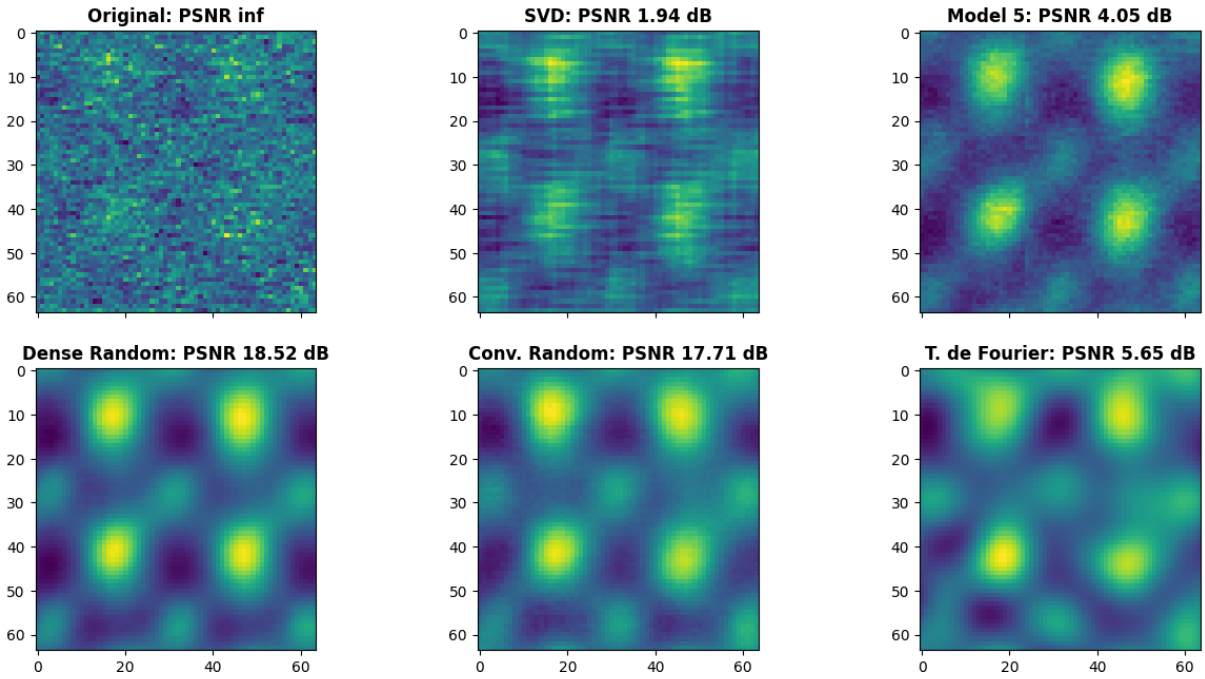


Figura 4.13: Comparación de resultados con valores de PSNR (mirar cabecera)

Vemos como, efectivamente, los modelos *random* son aquellos que mayor valor de PSNR han obtenido, algo que esperábamos ya que a simple vista se observa un mejor resultado. En cuanto al modelo 5, aunque este identifica correctamente la estructura de la red cristalina y representa los distintos átomos con gran precisión, no consigue eliminar del todo el ruido, además de la existencia de las marcas de "costura" que comentamos al analizar sus resultados. La filtración realizada por el método SVD es

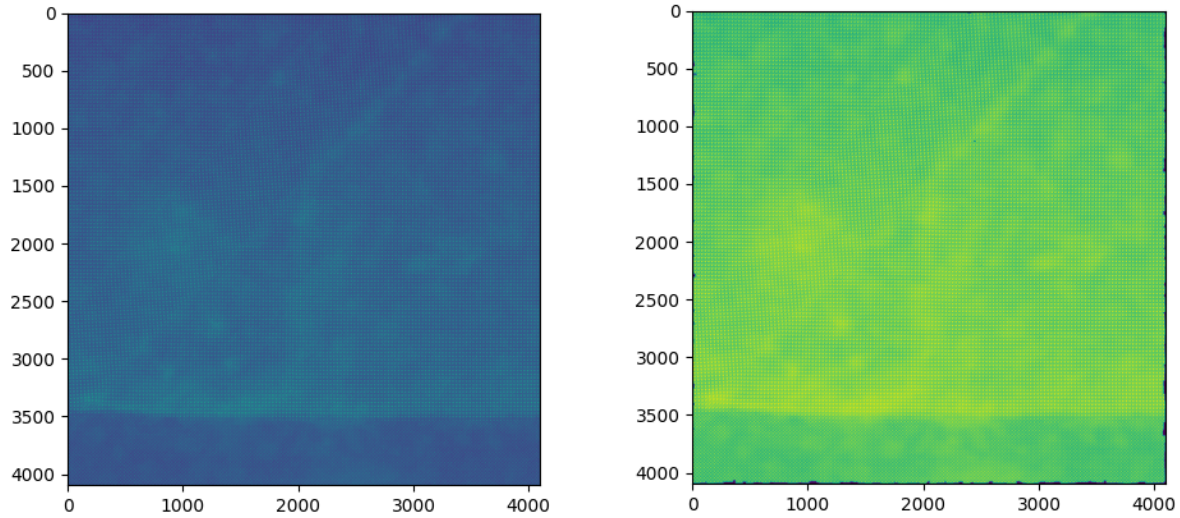
insuficiente, por lo que el valor obtenido es el más reducido y, por último, puesto que esta métrica cuantifica la proporción de ruido eliminado, el modelo de la transformada de Fourier obtiene un resultado más elevado que el modelo 5, a pesar de distorsionar levemente las formas de los átomos.

En la siguiente sección procederemos a analizar en detalle la imagen al completo, así como el resto de imágenes que nos han sido proporcionadas, en busca de dislocaciones e imperfecciones para comprobar si la red es capaz de reconstruirlas también. Pero para ello es necesaria la elección de uno de los modelos. En nuestro caso, debido a los valores de PSNR mostrados, nos hemos decantado por el modelo de *Autoencoder Dense Random*, a pesar de que a la vista es prácticamente idéntico al modelo de *Red Convolucional Random*. Es en este punto donde mayor importancia cobra el poder cuantificar el ruido de una imagen, ya que hay ocasiones en las que determinar si una imagen conserva mejor calidad que otra de forma subjetiva se vuelve una tarea complicada. Cabe comentar que el uso de este término no siempre resulta favorable, ya que imágenes tan disparatadas como la generada por el modelo 4 4.6 da un resultado mayor que el modelo SVD, a pesar de que es una imagen completamente ruidosa sin ningún tipo de información.

5. Aplicación de los resultados

Al comienzo de este trabajo hemos introducido que una de las principales motivaciones era desarrollar un modelo que eliminase el ruido de las imágenes generadas por STEM que tenemos a nuestra disposición. Como hemos visto a lo largo del desarrollo de los modelos, todos los resultados expuestos hasta el momento eran con respecto a la imagen 4.1a, la cual habíamos tomado como referencia. Por tanto, una cuestión importante es si el modelo que hemos generado es válido también para el resto de imágenes y es en este capítulo donde vamos a comprobarlo.

En primer lugar, comenzamos analizando una imagen de la misma película delgada, cuya estructura cristalina posee la misma orientación que la imagen de referencia, pero ha sido tomada en un punto distinto.

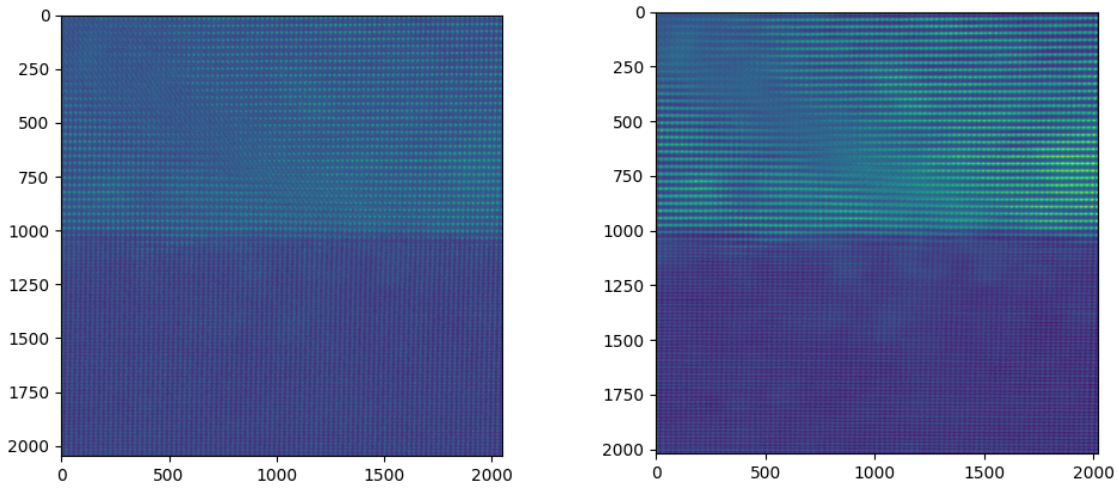


(a) Original (b) Filtrada

Figura 5.1: Imagen filtrada mediante *Autoencoder Dense Random*

Como podemos observar, ocurre algo que llama la atención a simple vista y es que al representar

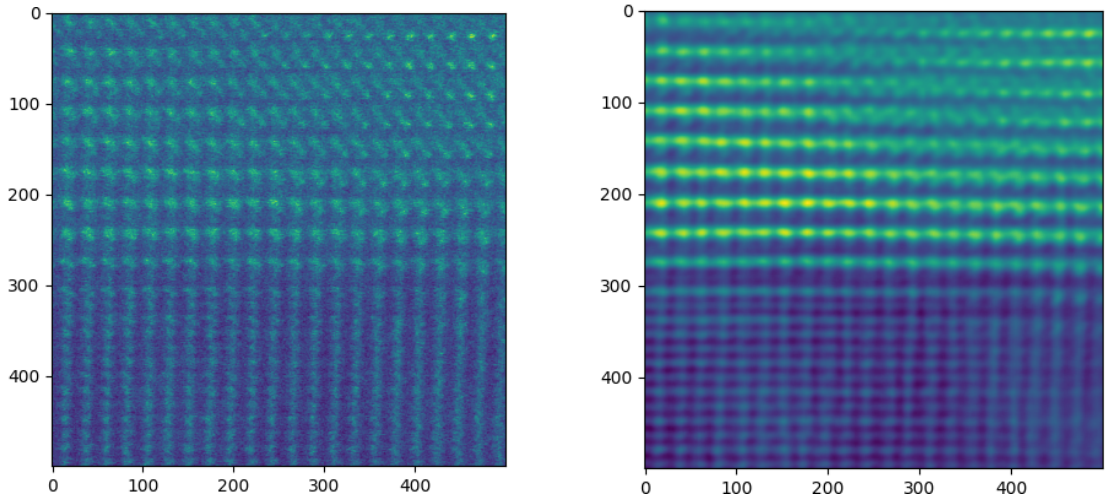
la imagen se cambian los colores, pasando de ser azul con tonalidades verdes a verde con tonalidades azules. Este cambio se debe a que al realizar los recortes introducidos en la red *Dense Random* de manera aleatoria, un pequeño segmento interior de imagen no fue recortado, por lo que a la hora de representarla la gama de colores cambia ya que hay unos píxels con valor nulo. Sin embargo, si representamos zonas concretas, como sucede en 4.9, el color original se recupera. Fijándonos detenidamente en ambas imágenes podemos apreciar que la red neuronal ha identificado la estructura a la perfección, presentando de este modo las mismas "manchas", cambios de intensidad, etc. Por lo tanto, podemos confirmar que el modelo es válido para imágenes que presentan las mismas características que nuestra imagen de referencia 4.1a. Veamos si sucede lo mismo cuando la orientación de la estructura cristalina ha cambiado.



(a) Original (b) Filtrada

Figura 5.2: Imagen filtrada mediante *Autoencoder Dense Random* con orientación cristalina diferente

Como vemos, en este caso hemos podido evitar el cambio de color, debido a que ninguna zona interior se ha visto afectada. Sin embargo, si nos aproximamos se puede apreciar como este filtrado no es tan claro, ya que, a pesar de que no aparenta haber la robustosidad o pixelación que produce el ruido en una imagen, las figuras se encuentran un tanto distorsionadas, siendo más complicado identificar los átomos que en el caso de la orientación inicial, sobre todo en la parte inferior de la imagen.

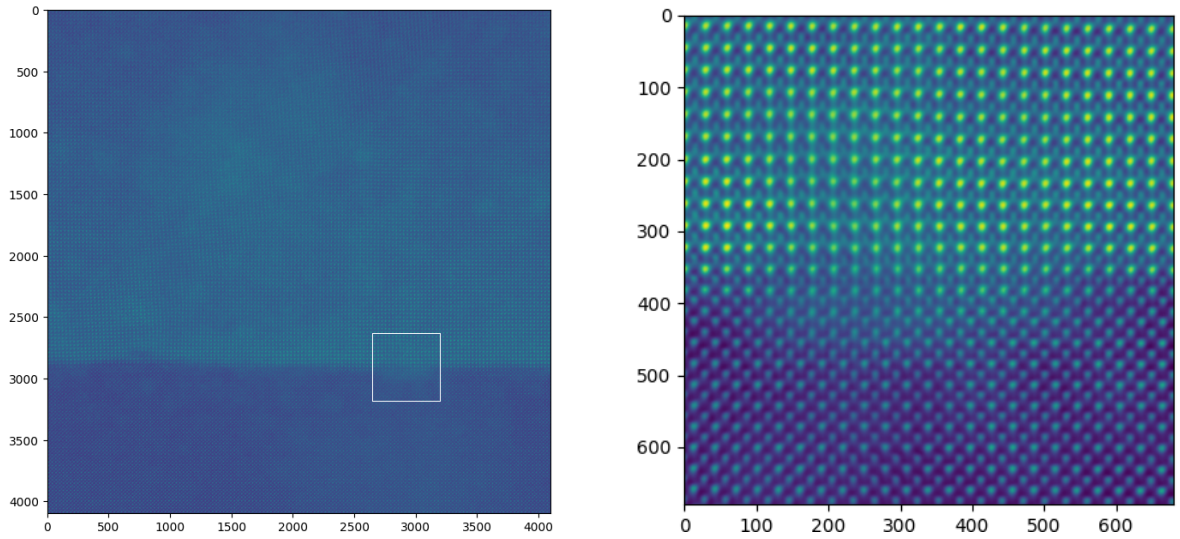


(a) Original (b) Filtrada

Figura 5.3: Recorte 500x500 de la figura 5.2

Por tanto, aunque el hecho de que la estructura cristalina no sea la misma haga que el modelo resulte menos efectivo, vemos como nuestra red trabaja correctamente con imágenes que no poseen exactamente las mismas características que nuestra imagen original, respetando de alguna manera la disposición de los átomos en la red y aquellas zonas que presentan perturbaciones, como es el caso de la esquina superior derecha de las figuras que acabamos de mostrar, donde se ve claramente una imperfección y nuestro modelo la detecta y la recrea sin problema alguno.

Finalmente, volviendo a nuestra imagen de referencia 4.1a, hemos observado como la red recrea los patrones de las imágenes perfectamente, sin embargo, no hemos comprobado qué sucede con estas zonas de distorsión cuando nos acercamos. Veamos si podemos visualizar una de estas zonas y si nuestro modelo nos permite especular sobre los posibles motivos. En concreto, vamos a observar la zona marcada por el cuadro blanco:



(a) 4096x4096 Original (b) 700x700 Filtrada

Figura 5.4: Observación de distorsión en imagen filtrada por modelo *Autoencoder Random*

Se puede apreciar claramente la zona de transición entre la sección verde y la azul, además de la borrosidad aparente del límite en la zona enmarcada por el recuadro blanco. Este cambio en la tonalidad de las imágenes se debe a que se trata de la imagen de una sección transversal de la muestra, es decir, de la profundidad, y por ello podemos observar dos capas diferentes ya que se trata de un ferroeléctrico sobre un buffer conductor. Por lo tanto, podemos observar como sí que es posible distinguir con precisión las distintas zonas de la lámina e incluso los distintos átomos, como vemos en la figura 4.9b, por lo que queda comprobada la utilidad de nuestra red para examinar detalladamente este tipo de imágenes.

6. Conclusiones

A lo largo de este trabajo se han visto distintas técnicas de *Inteligencia Artificial*, así como métodos clásicos para el tratamiento de imágenes. El reconocimiento de imágenes es uno de los campos más populares hoy en día y con mayor empleabilidad por la sociedad y hemos podido comprobar como el uso de determinadas *redes neuronales* no demasiado complejas puede permitirnos realizar tareas en cuestión de dos horas que antiguamente podía llevar días.

Se han diseñado distintos modelos de RRNN que han sido capaces de reconocer la estructura cristalina de las imágenes y de aprender la información subyacente a esta, eliminando la información innecesaria, como por ejemplo el ruido, a pesar de que se trataba de una estructura con forma de retícula y no presentaba contornos bien definidos, como sería el caso de los famosos dígitos del dataset MNIST. Una prueba más de la efectividad de dichos modelos es el correcto resultado obtenido a pesar de las limitaciones técnicas a la hora de desarrollar el trabajo, ya que la herramienta empleada ha sido mi ordenador portátil con un procesador Intel Core i5 y una gráfica GeForce GTX 1050, el cual nos ha permitido entrenar las redes que han resultado ser más efectivas en 2 horas en el caso del *dense autoencoder* y 47 minutos en el caso de la *red neuronal convolucional*. Este hecho puede suponer una gran diferencia si se entrenan las redes con un *dataset* de gran tamaño, por lo que, aunque en el trabajo hayamos escogido la primera de las redes, no descartamos que el modelo convolucional pueda presentar unos mejores resultados.

Sin embargo, durante el desarrollo del trabajo también hemos podido comprobar la dificultad a la hora de medir y entender los resultados en el campo de la reducción de ruido mediante estos métodos. Para cuantificar la bondad de los modelos obtenidos hemos usado el PSNR (ver ec. 4.3). Sin embargo, no es menos cierto que también conviene basarse en factores algo más subjetivos, como es la evaluación a ojo de las figuras filtradas. Por ejemplo, nótese que, dada una imagen ruidosa, otra imagen que fuera casi idéntica a la misma cambiando solo el valor de un píxel tendría un valor del PSNR enorme, mientras que seguiría siendo igual de ruidosa que la imagen original. Por tanto, hemos tenido que usar el factor humano, la subjetividad, para decidir finalmente qué resultados son más óptimos, ya que no hemos podido encontrar una técnica que sea 100 % efectiva. Hecho que nos lleva a concluir que el ojo sigue siendo uno de los métodos más fiables a la hora de comparar resultados en el campo del reconocimiento de imágenes ya que, aunque existen RRNN mejores que el ser humano a la hora de reconocer caras, estas han sido entrenadas a partir de datos escogidos y etiquetados por personas.

Es innegable el potencial que la *Inteligencia Artificial* tiene, ya que tan solo en este trabajo hemos podido observar lo poderosas que son las redes frente a las técnicas clásicas de tratamiento de señales y como sus resultados nos facilitan enormemente el estudio de estas, pudiendo analizar de una forma más profunda, en nuestro caso, lo que sucede en las láminas observadas. Y esta no es más que una diminuta parte del verdadero poder de la IA, de lo que se ha descubierto y de lo que queda por descubrir.

Bibliografía

[1]W. S. McCulloch y W. Pitts, "A logical calculus of the ideas immanent in nervous activity", The bulletin of mathematical biophysics **5**, 115-133 (1943).

[2]F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.", Psychological Review **65(6)**, 386-408 (1958).

[3]D. E. Rumelhart, G. E. Hinton y R. J. Williams, "Learning representations by back-propagating errors", Nature **323**, 533-536 (1986).

[4]F. Chollet, *Deep Learning with Python, second edition* (Manning Publications Co., 2021).

[5]G. W. Stewart, "On the Early History of the Singular Value Decomposition", SIAM Review **35**, 551-566 (1993).

[6]L. Ruiz Fernández, "La transformada de Fourier. Aplicación al filtrado de imágenes", Escuela Técnica Superior de Ingeniería Geodésica, Cartográfica y Topográfica de la Universidad Politècnica de Valencia.

[7]M. Nielsen, *Neural Networks and Deep Learning* (Determination Press San Francisco, 2015).

[8]D. A. Rosebrock, *Deep Learning for Computer Vision with Python, Starter Bundle* (PYIMAGESEARCH, 2019).

[9]L. Llano, A. Hoyos Palacio, F. Arias y J. Velásquez, "Comparación del Desempeño de Funciones de Activación en Redes Feedforward para aproximar Funciones de Datos con y sin Ruido.", RASI **4**, 79-88 (2007).

[10]S. Ruder, "An overview of gradient descent optimization algorithms", CoRR **abs/1609.04747** (2016).

[11]L. Bottou, "Stochastic Gradient Learning in Neural Networks", en Proceedings of Neuro-Nîmes 91 (1991).

[12]D. P. Kingma y J. Ba, "Adam: A Method for Stochastic Optimization", (2017).

[13]D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison y G. E. Dahl, "On Empirical Comparisons of Optimizers for Deep Learning", CoRR **abs/1910.05446** (2019).

[14]H. Zhang, L. Zhang e Y. Jiang, "Overfitting and Underfitting Analysis for Deep Learning Based End-to-end Communication Systems", en 2019 11th International Conference on Wireless Communications and Signal Processing (WCSP) (2019), págs. 1-6.

[15]M. H. Sazlı, "A brief review of feed-forward neural networks", Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering, 10.1501/commua1-2_0000000026 (2006).

[16]M. D. Zeiler y R. Fergus, "Visualizing and Understanding Convolutional Networks", CoRR **abs/1311.2901** (2013).

[17]Y. LeCun, K. Kavukcuoglu y C. Farabet, "Convolutional networks and applications in vision", en Proceedings of 2010 IEEE International Symposium on Circuits and Systems (2010), págs. 253-256.

[18]U. Michelucci, "An Introduction to Autoencoders", CoRR **abs/2201.03898** (2022).

[19]D. Bank, N. Koenigstein y R. Giryes, "Autoencoders", CoRR **abs/2003.05991** (2020).

[20]F. Charte, "Autoencoders ¿Qué son, para qué sirven y cómo funcionan?", Universidad de Jaén (2021).

[21]H. C. Burger, C. J. Schuler y S. Harmeling, "Image denoising: Can plain neural networks compete with BM3D?", en 2012 IEEE Conference on Computer Vision and Pattern Recognition (2012), págs. 2392-2399.

[22]L. Gondara, "Medical Image Denoising Using Convolutional Denoising Autoencoders", en 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW) (dic. de 2016).

[23]O. S. Faragallah, H. El-Hoseny, W. El-Shafai, W. A. El-Rahman, H. S. El-Sayed, E.-S. M. El-Rabaie, F. E. A. El-Samie y G. G. N. Geweid, "A Comprehensive Survey Analysis for Present Solutions of Medical Image Fusion and Future Directions", IEEE Access **9**, 11358-11371 (2021).