eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Predicción de estilo y/o autor de obras de arte

Autor

Igor Sorarrain Rebollar

2022

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Predicción de estilo y/o autor de obras de arte

Autor

Igor Sorarrain Rebollar

Director

Manuel Graña Romay

Resumen

El objetivo principal de este trabajo de fin de grado (TFG) se basa en el reconocimiento, tanto de estilos de obras de arte, como de tratar de predecir el artista que se encuentra detrás de cada obra pictórica. Para ello se han analizado y utilizado algunas de las arquitecturas de aprendizaje profundo para clasificación de imágenes más relevantes como *VGG-16*, *ResNet-34*, *ResNet-50* y *DenseNet-121*. Por otro lado, se ha creado un modelo desde cero para realizar dichas predicciones, y se han comparado los resultados obtenidos por esta red con los modelos anteriormente mencionados.

Todos estos modelos son en realidad redes neuronales convolucionales o *Convolutional Neural Network* (CNN) en inglés. Los modelos utilizados tendrán que predecir el estilo de una obra de arte, y catalogarlo dentro de los 20 estilos pictóricos seleccionados para este trabajo. En cuanto a predecir el artista, se han utilizado los mismos modelos, y el mismo número de casos, 20 artistas en total a predecir. Además, se han realizado algunas variaciones en estos experimentos con el fin de mejorar los resultados, ya sea fusionando estilos artísticos que estén históricamente relacionados, como calculando un *learning rate* adecuado para optimizar el aprendizaje de los modelos.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

1.1. Clasificación de imágenes

La clasificación de imágenes es uno de los apartados más importantes del *Computer Vision*, siendo este una parte del famoso *Machine learning*. Básicamente contamos con una base de datos de imágenes, y tenemos la tarea de clasificar las imágenes dentro de un número determinado de clases. Por ejemplo, imaginemos que contamos con un conjunto de cien imágenes, con fotos de coches, edificios, árboles y perros; y tenemos la única función de determinar a qué clase pertenece cada imagen. A simple vista, parece una tarea sencilla, y que cualquier persona podría hacer sin ningún problema. Ahora supongamos que en vez de cien imágenes y cuatro clases distintas, tenemos cien mil imágenes, y tenemos que determinar a qué raza pertenece el perro, qué clase de coche es o qué tipo de árbol tenemos delante. Distinguir imágenes en cientos de clases no solo sería una tarea mucho más difícil de realizar para una persona, si no que llevaría muchísimo más tiempo. Es por eso que la clasificación de imágenes es una tarea muy importante y complicada, ya que el propio ordenador tiene que ser capaz de aprender las características de las propias imágenes y sacar conclusiones de forma autónoma.

En este caso, las imágenes que vamos a clasificar son obras de arte, y no solo vamos a intentar determinar el estilo pictórico al que pertenece, sino que también intentar deducir la persona a la que pertenece dicha obra. Esta tarea sería bastante difícil para una persona que no sea experta en arte, y es por ello una opción muy atractiva para el ámbito del *Computer Vision*. Para ello, contaremos con varias redes neuronales convoluciona-

les (CNN)[Ciresan et al., 2011], ya que estas son una de las mejores herramientas que se pueden utilizar para la clasificación de imágenes.

1.2. Utilización de diferentes modelos

El *Deep Learning*, que es una de las técnicas de *Machine Learning*, consiste en hacer que un ordenador sea capaz de aprender y mejorar a realizar ciertas tareas, utilizando redes neuronales artificiales, basándose en el comportamiento del cerebro humano. Los modelos de *Deep Learning* pueden aprender automáticamente a partir de archivos como imágenes, texto o audio, y no precisan de una selección de características realizada previamente. Es por ello, que la clasificación de imágenes es una tarea apropiada para este ámbito, y por eso se utilizarán las redes neuronales convolucionales (CNN).

Como ya se ha mencionado, utilizaremos distintos modelos de CNN para la clasificación de estilos y artistas. Entre ellos se encuentran *ResNet-34*, *ResNet-50*, *DenseNet-121* y *VGG-16*, los cuales veremos más a fondo en la sección 4. La elección de utilizar modelos de CNN se basa en que este tipo de arquitecturas resulta uno de los métodos más eficaces para la clasificación de imágenes [Eva Cetinic, 2018]. Esto se debe a que las redes neuronales convolucionales están compuestas por neuronas artificiales, las cuales simulan a las neuronas de la corteza visual primaria de un cerebro. Por otro lado, este tipo de redes también es utilizada para la clasificación de audio [Hershey et al., 2017] o para clasificar objetos en 3D [Wang et al., 2019]. Además compararemos los resultados obtenidos por dichos modelos con un modelo creado desde cero.

1.3. Objetivos

El objetivo principal de este trabajo consiste en comparar el desempeño de distintas arquitecturas basadas en CNN, además de compararlos con un modelo creado desde cero.

Para clasificar obras de arte, se han tenido en cuenta los siguientes 20 estilos : "*Abstract Expressionism*, *Art Informel*, *Art Nouveau (Modern)*, *Baroque*, *Cubism*, *Early Renaissance*, *Expressionism*, *High Renaissance*, *Impressionism*, *Mannerism (Late Renaissance)*, *Naïve Art (Primitivism)*, *Neoclassicism*, *Northern Renaissance*, *Post-Impressionism*, *Realism*, *Rococo*, *Romanticism*, *Surrealism*, *Symbolism* y *Ukiyo-e*". Esta elección se debe a

que son los 20 estilos del dataset utilizado con más imágenes (cada uno de estos estilos cuenta con al menos 1200 obras en el dataset).

Por otro lado, como veremos en la sección 6, esta combinación de estilos causa bastantes dependencias entre ellos, debido a que muchos de los 20 estilos están relacionados entre sí, ya que por ejemplo el *Post-Impressionism* deriva del *Impressionism*, o que contamos con cuatro ramas distintas del renacimiento: *Early Renaissance, High Renaissance, Mannerism (Late Renaissance)* y *Northern Renaissance*. Para reducir las confusiones, se ha experimentado fusionando algunos estilos en uno solo, quedándonos con los siguientes 13: "*Abstract-Expressionism-Informel, Art Nouveau (Modern), Baroque-Rococo, Cubism, Impressionism, Naïve Art (Primitivism), Neoclassicism, Realism, Renaissance, Romanticism, Surrealism, Symbolism* y *Ukiyo-e* "

En cuanto a los artistas a predecir, se ha querido mantener el mismo número (20), por lo que se ha elegido a los 20 artistas con más obras (tienen 497 o más cuadros en el dataset, [1] como se explica en la sección 5): "*Ivan Aivazovsky*, *Gustave Dore*, *Rembrandt*, *Claude Monet*, *Pierre-Auguste Renoir*, *Albrecht Durer*, *Ivan Shishkin*, *Giovanni Battista Piranesi*, *Raphael Kirchner*, *Paul Cezanne*, *John Singer Sargent*, *Zdislav Beksinski*, *Camille Pissarro*, *Boris Kutsodiev*, *Ilya Repin*, *Martiros Saryan*, *Pyotr Konchalovsky*, *Pablo Picasso*, *Marc Chagall* y *Paul Gauguin*."

[1] El dataset se encuentra en el siguiente enlace: https://github.com/somewacko/painter-by-numbers/releases/tag/data-v1.0

2. CAPÍTULO

Desarrollo y gestión del proyecto

En este capítulo, se describe la gestión del proyecto estimada, así como las tareas que se han llevado a cabo y los problemas que han aparecido a lo largo del proyecto, causando desviaciones.

2.1. Gestión del proyecto estimada y tareas

Este proyecto ha sido subdividido en distintas tareas para llevarse a cabo. Dicha estructura se puede observar en la figura 2.1, donde se representa el trabajo en un esquema de *Estructura de Desglose de Trabajo*. Como se puede observar, el proyecto cuenta con cuatro tareas principales:

1. Gestión
2. Preparación
3. Implementación
4. Documentación

En el apartado de Gestión, se deciden y analizan las tareas que se van a llevar a cabo a lo largo del proyecto. Es decir, se trata de organizar el trabajo antes de empezar con él. Dentro de este apartado encontramos las subtareas de *planificación de tareas* y *decidir*

entorno de trabajo. En la primera, desglosamos el trabajo en las tareas que se van a realizar, y en la segunda decidimos las herramientas (software) que van a ser utilizadas.

Dentro de la preparación tenemos el primer contacto con el ámbito donde se va a trabajar. Aquí, las tareas principales se basan en leer artículos sobre trabajos previamente realizados dentro de la predicción de estilo y autor de obras de arte, además de otros artículos relacionados con el tema. También hay que conseguir los datos con los que se va a trabajar, así como hacer unas primeras pruebas con los datos acomodados y redes vistas en artículos.

En cuanto a la implementación, entramos de lleno con lo visto hasta ahora, se pone en práctica lo aprendido y si es necesario se realiza alguna mejora o añadido. También se solucionan los errores que puedan ocurrir.

Por último tenemos la documentación, la cual consiste en plasmar todo el trabajo realizado. Por un lado, tenemos la memoria, la cual contiene todo lujo de detalles acerca del proyecto. Y por otro lado, tenemos la presentación, que contiene las mismas ideas que la memoria, pero mucho más resumidas y directas (además de la preparación de la misma).

En la tabla 2.1, se encuentra una estimación de dedicación de las horas necesitadas para acabar el proyecto. Además, para contrastar la estimación, la misma tabla cuenta con una columna con las horas reales necesitadas. Por otro lado, tenemos la figura 2.2, donde se puede observar el diagrama *Gantt* que muestra una estimación de las tareas que se van a llevar a cabo según la semana y el mes.

2.2. Problemas e imprevistos

A lo largo del proyecto, han aparecido numerosos problemas e imprevistos, causando desviaciones en las horas estimadas. Estos son los imprevistos que más han afectado al desarrollo del proyecto, y algunas de las soluciones llevadas a cabo:

1. **Imágenes y dataset:** El dataset que se ha utilizado cuenta con 103.253 imágenes, las cuales llegan a ocupar demasiado espacio en el ordenador. El problema consiste en que descargar y descomprimir archivos muy grandes (alrededor de 50 GB) lleva bastante tiempo y espacio, y a la hora de cargar y procesar las imágenes hace que el ordenador pueda saturarse y que su rendimiento caiga. Para solucionar esto, se ha utilizado el mismo dataset, con el mismo número de imágenes, pero con las

imágenes reducidas en tamaño. De esta forma conseguimos que el espacio no sea un problema, aunque la efectividad de las predicciones puede caer debido a la pérdida de información al reducir las imágenes.

2. **Entorno de trabajo:** Otro de los problemas que más tiempo ha llevado, ha sido conseguir un entorno de trabajo adecuado. En un principio, se había utilizado el entorno de *Jupyter Notebook* con *python* y utilizando la CPU del ordenador. Debido a las limitaciones de recursos, y al excesivo tiempo necesitado para ejecutar un ciclo (más de una hora), se observó que utilizar la GPU en el ámbito de clasificación de imágenes era bastante más efectivo, y requería menos tiempo. Tras varias instalaciones de paquetes necesarios para utilizar la GPU en *Jupyter Notebook*, se comprobó que el tiempo mejoraba, pero no lo suficiente. Es por ello, que finalmente se ha utilizado la plataforma de *Google Colab*, reduciendo mucho los tiempos, aunque debido a las limitaciones de tiempo que hay en la plataforma, a veces se ha tenido que utilizar *Jupyter Notebook*.

3. **Seguridad de los datos:** Para evitar perder los datos (por ejemplo al sobrescribirse), la base de datos con imágenes se ha guardado tanto en la red *Google Drive*, como en un ordenador, además, está disponible en https://github.com/somewacko/painter-by-numbers/releases/tag/data-v1.0. De esta forma, los datos se encuentran bastante protegidos. Por otro lado, los artículos, programas, resultados y documentos creados y utilizados están guardados en distintos sitios, como en *Google Drive*, *Dropbox* y en un ordenador personal.

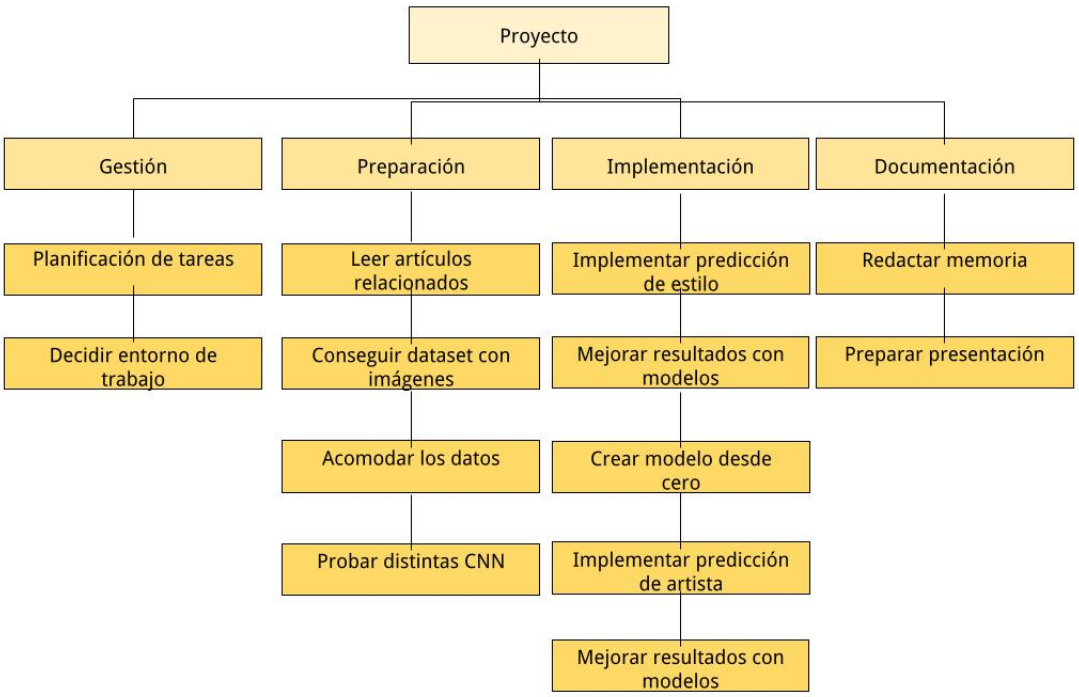

Figura 2.1: Estructura de desglose del trabajo (EDT) o *Work Breakdown Structure* (WSB) en inglés, descomponiendo el proyecto en tareas

Mes	Enero				Febrero				Marzo				Abril				Mayo				Junio			
Semana	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Planificación del trabajo			X	X																				
Lectura del estado del arte			X	X	X	X																		
Elegir software para la implementación				X	X																			
Buscar y conseguir bases de datos				X	X	X																		
Trabajar con modelos del estado del arte (Estilo)					X	X	X	X	X															
Crear y probar el modelo (Estilo)						X	X	X	X															
Mejoras del modelo (Estilo)								X	X	X	X													
Trabajar con modelos del estado del arte (Artista)									X	X	X	X												
Crear y probar el modelo (Artista)									X	X	X	X												
Mejoras del modelo (Artista)											X	X	X	X										
Comparar resultados													X	X	X	X								
Redactar la memoria						X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
Preparar la presentación																				X	X	X	X	

Figura 2.2: Diagrama Gantt de la planificación del trabajo

	Horas de trabajo estimadas	Horas de trabajo reales
Gestión	7	10
Planificación de tareas	5	5
Decidir entorno de trabajo	2	5
Preparación	52	50
Leer artículos relacionados	20	25
Conseguir dataset con imágenes	2	4
Acomodar los datos	5	8
Probar distintas CNN	25	13
Implementación	170	155
Implementar predicción de estilo	50	44
Mejorar resultados con modelos (estilo)	10	20
Crear modelo desde cero	50	41
Implementar predicción de artista	50	30
Mejorar resultados con modelos (artista)	10	20
Documentación	80	107
Redactar memoria	60	83
Preparar presentación	20	24
Total	309	322

Tabla 2.1: Tabla de comparación de horas estimadas y reales para el desarrollo del TFG

3. CAPÍTULO

Estado del arte

El ámbito de la predicción de estilo o artista de obras de arte utilizando redes neuronales convolucionales no es uno de los ámbitos más populares del mencionado *Computer Vision*, ya que se trata de un tema muy concreto. Es por esto que no son muchos los artículos publicados en torno a este tema, pero si generalizamos, nos encontramos ante un problema de clasificación de imágenes mediante la utilización de redes neuronales, por lo que es un tema con bastante repercusión. En este capítulo se resumen y explican algunos de los trabajos más relevantes en el área de la predicción de estilo y artista.

3.1. Trabajos relacionados con la predicción de estilo

Entre los muchos de los métodos desarrollados para la predicción de estilo de obras de arte, tenemos a [Yang and Min, 2019], en el cual se usa la técnica llamada *Patch based* para predecir los medios que se utilizaron para pintar obras de arte. Básicamente, consiste en encontrar la zona de la imagen que más información contiene, y para ello, se enfoca la atención en distintas zonas (*patches* o módulos) de una imagen mediante una estructura *multi-columna*. Los medios a predecir se reducen a cuatro: *Oil, Pastel, Pencil y Watercolor*; y son tres los *datasets* que se han utilizado para validar el modelo, consiguiendo un *accuracy* del 85 %, 93 % y 85 % respectivamente. Es verdad que se consigue un gran acierto, pero el conjunto de clases a predecir es bastante pequeño (solamente 4), y las diferencias entre clases son claras. Aunque es un método que ayuda bastante a sacar información acerca de las imágenes, puede que solamente sea efectivo a la hora de buscar

texturas o formas pequeñas, pudiendo no ser el mejor caso para reconocer estilos o artistas, ya que el mismo estilo puede ser creado con distintos medios; por ejemplo, obras del barroco al *óleo* o al *fresco*, o que el mismo autor utilice distintos medios.

Por un lado, [Karayev et al., 2013] es uno de los precursores en cuanto a la predicción de estilo en imágenes utilizando redes neuronales convolucionales, no solo en cuanto a estilo artístico, sino que también en estilo para diferentes imágenes, como fotografías. En este caso se utilizan dos *datasets* distintos; uno para imágenes artísticas, y otro con fotografías y distintas imágenes. En el primer caso, las imágenes proceden de *Wikipaintings*, y contiene cerca de 85.000 muestras de imágenes, conformando 25 estilos pictóricos. Por otro lado, tenemos las fotografías pertenecientes a *Flickr*, un sitio web donde los usuarios pueden subir sus propias fotografías y vídeos. Este segundo grupo está formado por 80.000 imágenes repartidas en 20 estilos; como por ejemplo *HDR, Vintage, Noir* o *Romantic*. Para le detección de características se utilizan varios descriptores: como el histograma de color *L*a*b* (ya que muchos estilos del *dataset* de *Flickr* se basan en colores), *GIST* (el cual es un conocido descriptor que funciona bien a la hora de clasificar escenas y también con imágenes de baja resolución), o los descriptores de CNN *Decaf5* y *Decaf6*, los cuales están sacados de redes convolucionales profundas con pesos de *ImageNet*. En el mejor de los casos, se obtiene una precisión media de 36.8 % en el *dataset* de *Flickr* y del 47.3 % en *Wikipaintings*, utilizando en ambos casos una fusión o combinación de los descriptores.

En otros artículos como en [Lecoutre et al., 2017], se utilizan distintos modelos preentrenados de CNN como *ResNet-34*, *ResNet-50* y *AlexNet* para predecir estilos. Pero a diferencia de este caso, los estilos utilizados son 25 y se realiza el proceso conocido como *Transfer Learning*, donde parte de la red se vuelve a definir para ajustarse al reconocimiento de estilos y se reentrena la red, haciendo así que aumente la precisión de las predicciones. Otra de las técnicas que se utilizan es el *Bagging*, que consiste en promediar la salida de diferentes predicciones en varias variaciones de datos de entrada, y el *Data augmentation*, que realiza cambios en las imágenes (como rotaciones, zoom, traslaciones de ejes, o cambios de brillo) para crear así nuevas imágenes a partir de una, enriqueciendo el conjunto de imágenes, evitando así el *Overfitting*. A pesar de ser una buena idea para aumentar los datos de entrada, no se ha realizado *Data augmentation* en este proyecto, debido a la posibilidad de afectar al propio estilo de una obra, pudiendo modificarlo y haciendo así que se pierda precisión. Por ejemplo, el brillo pude ser determinante en diferentes estilos como el *Impressionism*, donde la luz juega un papel muy importante, y es posible que al disminuir el brillo de una imagen los detalles de la luz no sean tan notables, haciendo que el modelo no clasifique una obra de este estilo correctamente, o al revés.

Hay otros trabajos como [Bar et al., 2015], donde al igual que en [Karayev et al., 2013] se utilizan varios descriptores visuales para extraer información sobre las obras de arte, consiguiendo así saber más sobre los estilos artísticos. Algunos de esos descriptores son detecciones de bordes, histogramas de gradiente, histogramas de color, métodos estadísticos o métodos basados en diccionarios. Otra de las técnicas utilizadas en este trabajo es *PiCoDes* [Bergamo et al., 2011], que se utiliza para aprender una representación de código binario de las imágenes optimizadas en un subconjunto del *dataset* de *ImageNet*. Por último, se utiliza una arquitectura de CNN, y se combina toda la información obtenida para realizar la predicción. En ese caso se utilizan 27 estilos pictóricos, pero el número de imágenes por estilo no está equilibrado. En algunos casos, hay estilos que cuentan con más de 3000 imágenes, mientras que otros no llegan a 300. En cuanto a la precisión, hay que decir que en el mejor caso es del 56 %, utilizando *PiCoDes* con una dimensionalidad de 2048 y los descriptores de CNN *Decaf5* y *Decaf6*.

3.2. Trabajos relacionados con la predicción de artista

En cuanto a reconocimiento de artista, tenemos trabajos como [Jangtjik et al., 2016], donde se basan en el uso de redes neuronales convolucionales y una representación piramidal. En este artículo trabajan con 1300 imágenes pertenecientes a 13 artistas, contando con 100 imágenes por artista, todas ellas de un tamaño de 224x224 píxeles. Además, cada imagen se trabaja de tres formas distintas, cada una de ellas se utiliza en tres capas, subdividiendo las imágenes en 1, 4 y 16 partes en cada capa (representación piramidal), y se entrena un modelo CNN en cada una. Por otro lado, se usa un esquema de fusión ponderado, que combina los modelos en función de su probabilidad estimada, consiguiendo una precisión del 71.22 %. A pesar de haber conseguido una buena precisión, el conjunto de datos utilizado es bastante pequeño, ya que se cuenta con 100 imágenes por clase.

En [Viswanathan and Stanford, 2017] también se utilizan redes neuronales convolucionales, pero a diferencia del caso anterior, el dataset está equilibrado y se cuenta con más obras por artista. El dataset que se usa es el conjunto de datos que pertenece a *WikiArt*, utilizado en una competencia de *Kaggle*, que es el mismo que se ha utilizado en este trabajo. Los modelos tienen en total 57 artistas diferentes a predecir, todos ellos con al menos 300 obras, y al igual que se ha hecho en este trabajo, se han elegido aleatoriamente el mismo número de obras para cada artista (300 de cada artista), para tener así un conjunto de datos equilibrado. En este trabajo se utilizan distintas CNNs como una red CNN simple creada desde cero, y dos versiones de la red *ResNet-18*, una añadiendo una capa *fully conected*

para realizar *Transfer Learning*. Para todos los casos, las imágenes están normalizadas, centradas y reducidas a 224x224 píxeles, y además se define una probabilidad de 0.5 de que una imagen esté girada horizontalmente. El modelo de CNN creado desde cero es utilizado como base (*baseline*), para comparar así los resultados de las demás redes. Esta red está formada por convoluciones 3x3, *Max Poolings* de 2x2, capas *ReLU* y *batch normalization*, y por último capas *fully conected* para realizar la predicción. Por otro lado, la red *ResNet-18* es igual a la versión de 18 capas que aparece en la figura 4.9 del apartado 4.3.2, exceptuando el hecho de que se le ha añadido una capa *fully conected* para realizar las predicciones de artista. Por último contamos con la variación de *ResNet-18* a la cual se le ha añadido *Transfer Learning*. A esta última red también se le ha añadido una capa *fully conected* para calcular las puntuaciones de las predicciones de artista, en vez de las puntuaciones para las clases de *ImageNet*. La diferencia principal entre este modelo y el anterior consiste en que está red se entrena con los pesos previamente calculados del dataset *ImageNet*, por lo que cuenta con mayor facilidad para reconocer objetos, paisajes o personas, y por lo tanto para asociar estos con los estilos. Es por esto, que es esta última red la que obtiene mejores resultados, concretamente consigue un *accuracy* del 77.7 %, mientras que la otra versión que utiliza *ResNet-18* tiene un acierto del 51.6 % y el modelo CNN un 41.6 %. Como era de esperar, en este trabajo se consiguen mejores resultados que en el anterior [Jangtjik et al., 2016], ya que se utilizan muchas más imágenes por clase, además de aumentar considerablemente el conjunto de artistas.

Como ya hemos visto, hay muchas similitudes entre este trabajo y [Viswanathan and Stanford, 2017], pero la mayor diferencia consiste en el uso de diferentes redes. En este caso hay mayor variedad de redes (como una CNN simple, *ResNet-34, ResNet-50, VGG-16* y *DenseNet-121*), por lo que los resultados son de mayor variedad. En [Viswanathan and Stanford, 2017] en cambio, solo se usan dos versiones de *ResNet-18* (se usa la versión de *ResNet* con menos capas para ahorrar tiempo y memoria) y una CNN simple, por lo que los resultados son peores. Por otro lado, se utiliza *Data Augmentation*, aunque de una manera muy sutil, ya que solamente se aplican rotaciones horizontales, las cuales no afectan mucho a la imagen, pero modifican las imágenes originales.

4. CAPÍTULO

Redes Neuronales Convolucionales utilizadas

4.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales o CNN, son una de las mejores herramientas dentro del *Deep Learning* para la clasificación de imágenes. Esto se debe a que su comportamiento se inspira en el del cerebro humano, más concretamente en las neuronas de la corteza cerebral primaria. Este tipo de redes también están basadas en los *perceptrones multicapa*, los cuales son un tipo de red neuronal artificial, formada por distintas capas (capa de entrada, capa(s) oculta(s) y capa de salida) por lo que puede resolver problemas que no son linealmente separables. Podemos observar la estructura de una CNN en la imagen 4.1.

El funcionamiento principal de este tipo de redes se debe a la aplicación de filtros convolucionales, formados por grupos de *kernels*. Estos filtros pueden ser de distintos tamaños, como por ejemplo de 3x3, 5x5 o 7x7. Con los filtros se extrae la información a través de la convolución, la cual consiste en calcular el producto escalar entre la imagen de entrada y una matriz (el filtro). La entrada o *input* de la CNN es una imagen (en este caso una obra artística), compuesta por tres dimensiones si es a color, y dos si se trata de una imagen en blanco y negro.

El primer paso para entrenar una CNN es el pre-procesamiento de los datos. Como en nuestro caso tenemos todas las imágenes a color y redimensionadas a 256x256 píxeles, tendremos que separarlas en tres canales, cada uno para cada color de RGB, por lo que contaremos con 256x256x3 neuronas en la capa de entrada, es decir 196.608 neuronas.

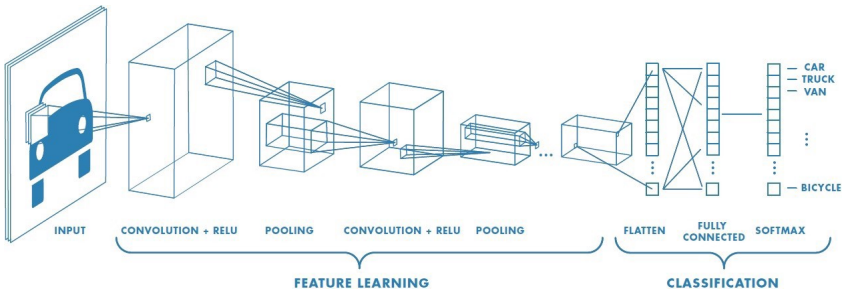

Figura 4.1: Estructura de una red neuronal convolucional. Imagen de https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

El segundo paso consiste en realizar la convolución, por lo que nos encontramos en un caso similar al de la imagen 4.2, donde se calcula el producto escalar de entre 9 píxeles (contiguos) de la imagen de entrada y el *kernel* de tamaño 3x3. El filtro se 'mueve' a través de toda la imagen, para poder así conseguir información de la imagen entera. Como contamos con imágenes separadas en tres canales RGB, se realizan tres convoluciones distintas para cada canal, y después se suman. Si por ejemplo tenemos 32 filtros por convolución, al realizar este proceso, obtendríamos 32 matrices de salida, obteniendo 32 rasgos o características diferentes de la misma imagen. A este conjunto de matrices se le llama *feature mapping*. En este caso, al tener imágenes de 256x256 y suponiendo que tenemos 32 filtros, la primera capa oculta tendría 256x256x3x32 = 6.291.456 neuronas. A diferencia de otros métodos donde los filtros se definen manualmente, las CNN tienen la habilidad de aprender y optimizar los valores para los filtros, haciéndolos más efectivos (ya que puede descartar filtros que no aportan mucha información, y crear otros nuevos) mediante *Backpropagation*.

A los resultado obtenidos tras la convolución entre la imagen y el filtro, aplicamos una función de activación, para clarificar los valores de la matriz de convolución. Una de las funciones de activación más comunes es *ReLU (Rectified Linear Unit)*, la cual se define de la siguiente manera: $f(x) = max(0,x)$. Gráficamente, la función tiene forma de rampa, ya que cuando x es menor o igual a 0, la función vale 0, y vale x a partir de 0. Con esto, evitamos los resultados negativos y los convertimos en 0.

En el siguiente paso tendremos en cuenta las características más relevantes obtenidas por la convolución, y volveremos a aplicarla para obtener mejores características o detallarlas mejor; pero para evitar aumentar en exceso la cantidad de neuronas de la red, se hace un sub-muestreo, que consiste en retener algunos de los valores, descartando los demás.

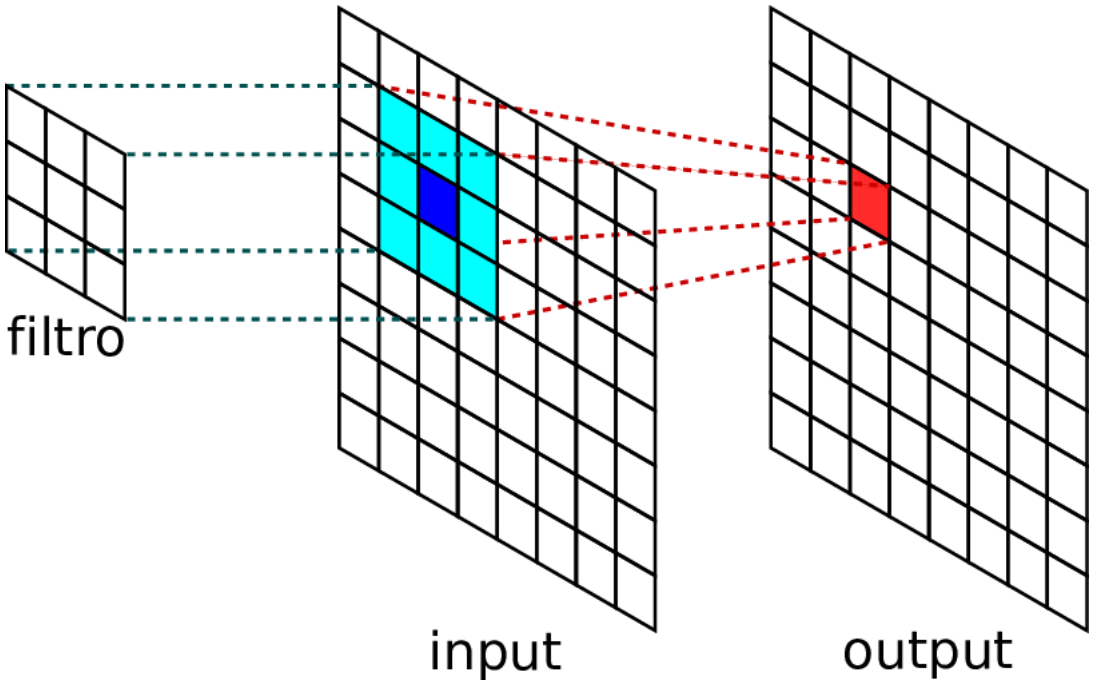

Figura 4.2: Ejemplo visual de una convolución. Imagen de [Redolfi, 2018]

Unos de los más utilizados son el *Max-Pooling* y el *Average-Pooling*, en este caso nos centraremos en el primero. Esta técnica consiste en agrupar píxeles, por ejemplo en grupos de 2x2, y quedarnos solamente con el píxel de mayor valor. Si contamos con 32 imágenes filtradas con un tamaño de 256x256, y aplicamos *Max-Pooling*, nos quedaremos con las 32 imágenes, pero de tamaño 128x128. Podemos ver un ejemplo en la imagen 4.3, donde la entrada tiene un tamaño de 4x4, se hace un *Max-Pooling* de 2x2 (cada color representa cada grupo) y la salida tiene un tamaño de 2x2, ya que de cada cuatro valores que tiene cada grupo, nos quedamos con el más alto. De esta forma, se reduce a la mitad la imagen en lo ancho y alto. En el caso de tener 6.291.456 neuronas tendríamos la mitad, 3.145.728.

Con esto habríamos hecho la primera convolución, pero si queremos que nuestro modelo sea más profundo y con mejor precisión, tendremos que repetir este proceso varias veces. A medida que avancemos, las imágenes serán más pequeñas y contarán con una menor resolución, por lo que para no disminuir drásticamente el número de neuronas, aumentaremos el número de filtros, como por ejemplo duplicando la cantidad de filtros por convolución. Este proceso se podrá hacer hasta que las imágenes filtradas no puedan ser procesadas por un *Max-Pooling*; si por ejemplo tenemos imágenes de 256x256, en cada convolución dividiremos en dos su tamaño, llegando a 128, 64, 32, 16, 8, y 4, haciendo

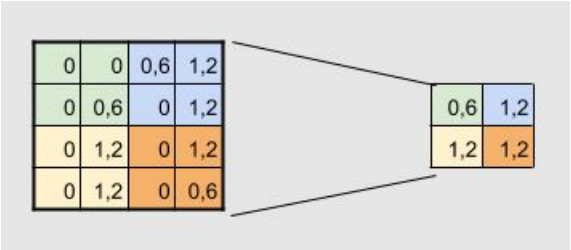

Figura 4.3: Proceso de *Max-Pooling*. Imagen de https://www.juanbarrios.com/redes-neurales-convolucionales/

como mucho 6 convoluciones.

Al finalizar las convoluciones, nos quedaremos con una última capa oculta, la cual tendremos que adaptar para que pueda ser compatible con la capa de salida, para hacer así una predicción. Básicamente hay que hacer que esa capa deje de ser tridimensional, y tenga una sola dimensión, es decir convertirla en un vector. A este proceso se le llama *Flattening*. Por último, aplicaremos una función *Softmax*, la cual se encarga de juntar la capa *Flatten* con la capa de salida, convirtiendo la salida de la red en estimaciones de las probabilidades de las clases a predecir. Así, la salida será un vector con una longitud igual al número de clases a predecir, y la suma de todos sus valores será igual a 1, indicando cada valor la probabilidad de que una imagen pertenezca a cierta clase. Como la predicción que se va a hacer está dentro de unos valores definidos, se dice que se ha hecho un aprendizaje supervisado.

En la figura 4.1 podemos ver todos los procesos y partes comentadas anteriormente, y además se puede observar que el ancho de las capas va aumentando a la vez que disminuyen en tamaño, esto se debe a que aumenta el número de filtros que se utilizan, y el tamaño de las imágenes disminuye debido a la convolución y al *Max Pooling*.

Uno de los problemas a evitar con las redes neuronales convolucionales, es limitar el número de capas a una cifra demasiado baja. Como ya hemos visto, la convolución hace que la imagen de la siguiente capa sea reducida, por lo que los detalles que se pueden obtener de la imagen original y la imagen reducida no son los mismos. De esta forma, si tenemos muy pocas capas, los detalles que se detectarán serán más visibles, como bordes o el color, mientras que si tenemos más capas, los detalles de las imágenes más pequeñas serán diferentes y nos aportarán más información. La estructura del modelo CNN que se ha creado desde cero para este trabajo se encuentra en la sección 4.5.

4.2. VGG

La red neuronal *VGG* (*Visual Geometry Group*) [Simonyan and Zisserman, 2015], es una de las redes más populares para la clasificación de imágenes. En el año 2014 quedó en segunda posición en la tarea de clasificar imágenes en el evento *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) con un error de clasificación del 7.325 %, por detrás de *GoogleNet*, que obtuvo un error del 6.656 %. Por otro lado, ganó en la categoría de localización con un error del 25.3231 %.

El propósito principal de la publicación de esta red, consistía en demostrar que aumentando la profundidad de una red (número de capas), se podía aumentar su eficacia, siempre hasta cierto punto. Además, uno de los factores que la hacen destacar, es el cambio del tamaño de los filtros para la convolución. Redes como *AlexNet* [Krizhevsky et al., 2012] (otra de las CNNs más utilizadas para la clasificación de imágenes), utilizan *kernels* (filtros) de tamaño 5x5, 7x7 o hasta 11x11. En este caso, se utilizan conjuntos de filtros de 3x3 en vez de filtos 5x5 o 7x7, ya que resulta menos costoso.

Resumiendo, la idea de *VGG* consiste en reducir el tamaño de los filtros de convolución, a la vez que se aumentan en cantidad. Por ejemplo, supongamos que tenemos un filtro de tamaño 7x7. Si lo sustituimos por tres filtros de tamaño 3x3, y suponiendo que tenemos C^2 canales de entrada y salida, el primer caso contaría con $49C^2$ parámetros (7x7 = 49). Mientras que en el segundo, serían $27C^2$, ya que 3x(3x3) = 27. De esta forma, no solo reducimos el número de parámetros de nuestra red (ya que $27C^2 < 49C^2$), sino que aumentamos su profundidad (debido a que utilizamos más capas al añadir más convoluciones). Al igual que con filtros de 7x7, podemos reemplazar un filtro de 5x5 con dos filtros de 3x3, o un filtro 11x11 con cuatro filtros de 3x3.

En la figura 4.4, podemos observar las diferentes 6 versiones planteadas de *VGG*, con 11, 11, 13, 16, 16 y 19 capas respectivamente. En todos los casos, las imágenes de entrada son de 224x224 píxeles. Para cada red, se pueden apreciar en letra negrita los cambios añadidos respecto a la versión anterior (la de su izquierda). La red *A* cuenta con 11 capas, mientras que *A-LRN* es igual que *A*, pero aplicándole normalización LRN (*local response normalisation*). Después tenemos la red *B*, la cual cuenta con 13 capas, añadiéndole 2 convoluciones al caso *A*. Las redes *C* y *D* tienen ambas 16 capas, pero las convoluciones que se añaden en *C* son de tamaño 1x1, mientras que las de *D* son de 3x3. Por último está *E*, la cual es igual a *D* pero añadiéndole 3 convoluciones.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 4.4: Diferentes versiones de *VGG* según el número de capas y sus tamaños. Imagen de [Simonyan and Zisserman, 2015]

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figura 4.5: Número de parámetros que contiene cada red (en millones). Imagen de [Simonyan and Zisserman, 2015]

4.2.1. VGG-16

La versión de *VGG* utilizada para este trabajo es *VGG-16*, ya que ambas versiones publicadas(*VGG-16* y *VGG-19*) son bastante parecidas en cuanto a rendimiento y tiempo de ejecucción. Esta versión es la red *D* de la figura 4.4, ya que a diferencia de la red *C*, que también cuenta con 16 capas, no utiliza convoluciones de tamaño 1x1. Como hemos visto antes, está compuesta por 13 capas convolucionales y 3 densas. Por otro lado, podemos ver en la imagen 4.5, que a pesar de haber bastante diferencia entre las capas, el número de parámetros (en millones) es bastante similar. Esta red cuenta con 138 millones de parámetros, mientras que la variante de 19 capas cuenta con 144 millones.

4.3. ResNet

Es lógico pensar que al añadir capas a una red neuronal, haciéndola más profunda, su eficacia se verá aumentada debido a su mayor complejidad. En realidad, las redes neuronales profundas pueden sufrir el problema de desaparición de gradiente o *vanishing gradient*. En estos casos, el valor del gradiente va acercándose a 0, haciendo que la red aprenda mucho más despacio o hasta deje de aprender (debido a la profundidad de la red), por lo que el error de entrenamiento no siempre disminuye a la hora de añadir capas a la red, sino que se satura, y después aumenta. Para evitar este problema, la red neuronal *ResNet*, propuesta en [He et al., 2016], utiliza bloques residuales y atajos (conocidos como *shortcuts*). El termino *ResNet* viene de *residual neural network* o red neuronal residual, el cual está basado en construcciones de células piramidales en la corteza cerebral.

Esta arquitectura fue la ganadora del *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) en el año 2015 [Russakovsky et al., 2015], donde se evalúan distintos algoritmos para le detección de objetos y clasificación de imágenes con un conjunto de datos muy grande. *ResNet* obtuvo una tasa de error del 3.57 %, obteniendo el mejor resultado ese año, situándose por debajo de la tasa de error media de un humano en la clasificación de imágenes, la cual se encuentra alrededor del 5 %. Además, a pesar de contar con muchas más capas (152), resultó ser una red bastante más simple que otras con menos capas (por ejemplo *VGG-19* cuenta con 19 capas y cuenta con más parámetros, teniendo 8 veces menos capas).

Este tipo de arquitectura usa bloques residuales, los cuales hacen que la entrada x de una capa, se sume a la salida de la misma. De esta forma, si la salida es $F(x)$, la nueva salida

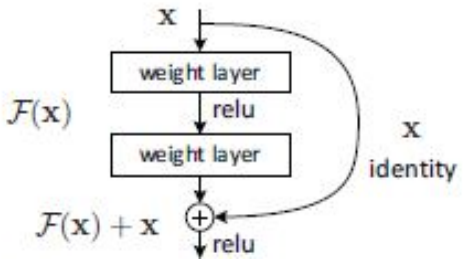

Figura 4.6: Bloque residual de la arquitectura *ResNet*. Imagen de [He et al., 2016]

será *F(x)+x*. Podemos observar la estructura de estos bloques en la imagen 4.6. Como se puede observar, la entrada de la red es *x*, y al aplicar funciones de activación como *ReLU*, obtenemos *F(x)*. Por otro lado, tenemos un acceso directo o atajo de *x* a la salida (llamado *identity*), donde se realiza la suma con *F(x)*, por lo que la salida, también conocida como *H(x)*, es igual a *F(x)+x*. Para poder realizar dicha suma, tanto *F(x)* como *x* tienen que tener la misma forma. Para eso, se multiplica a *x* por una matriz llamada *Ws*. En el caso de que no haya que utilizar la matriz *Ws*, debido a que *x* y *F(x)* son compatibles, al bloque residual se le conoce como bloque residual de identidad. Si en cambio, hay que hacer que *x* y *F(x)* sean compatibles, el bloque residual utilizado es conocido como bloque residual convolucional.

Las capas en las cuales la entrada va directamente a la salida mediante atajos se dice que son mapas de identidad. En estos casos no se aprenderá nada en esa capa, por lo cual, puede afectar negativamente a la precisión del modelo si en esas capas se consiguen detalles importantes para la predicción. En cambio, quizás mediante esas capas no se consiga detalle alguno, o no aporte nada, o incluso sea negativo para el aprendizaje, por lo que saltarse esa capa beneficia a nuestro modelo. Es por esto, que la ventaja principal de *ResNet* se basa en poder apartar las capas que no aportan nada al modelo, utilizando solo las que verdaderamente son necesarias para la clasificación.

4.3.1. ResNet-34

En la imagen 4.7 podemos observar la estructura de la red *ResNet-34*, que como indica su nombre, cuenta con 34 capas. La entrada de la red es una imagen de 224x224 píxeles. Las convoluciones se realizan con matrices de tamaño 3x3 excepto en el primer caso, que se realiza con un tamaño de 7x7. A pesar de ser más profunda que otras redes como *VGG-19* (que tiene 144 millones de parámetros), esta red cuenta con tan solo 21.8 millones de pa-

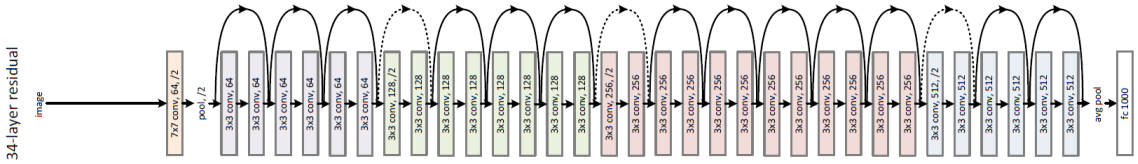

Figura 4.7: Estructura de la red *ResNet-34*. Imagen de [He et al., 2016]

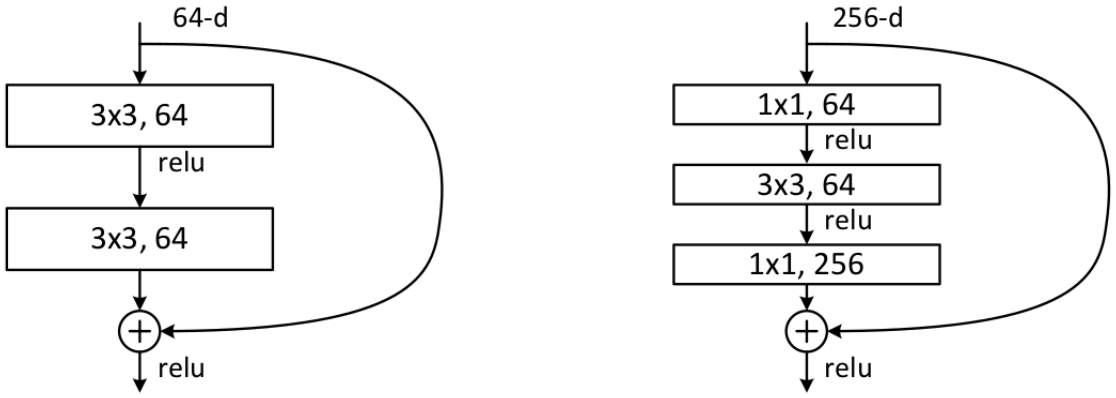

Figura 4.8: Comparación de las capas entre *ResNet-34* y *ResNet-50*. Imagen de [He et al., 2016]

rámetros. Además, se puede observar como las líneas curvas con flechas indican los saltos o atajos entre capas, saltándose siempre dos capas. Las curvas con líneas discontinuas indican un cambio en el tamaño de la imagen, debido a la aplicación de convoluciones y *Max-Pooling*.

4.3.2. ResNet-50

Esta otra versión de *ResNet* cuenta con 50 capas y un número de parámetros muy parecido al anterior caso: 23 millones. En la imagen 4.9, se pueden observar las diferencias entre las versiones de 18, 34, 50, 101 y 152 capas de *ResNet*. A diferencia de *ResNet-34*, en este caso, los atajos se realizan de 3 en 3 capas. Podemos ver esta diferencia gráficamente en la imagen 4.8. La elección de usar ambas dos versiones de *ResNet* (34 y 50 capas), se basa en que son dos de las versiones de *ResNet* más populares, y además hay diferencias notables entre ellas (como la diferencia de el número de capas que se evitan con los atajos).

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 4.9: Comparación de las diferentes versiones de *ResNet*. Imagen de [He et al., 2016]

4.4. DenseNet

Por otro lado, nos encontramos frente a la red *DenseNet* (*Dense Convolutional Network)* [Huang et al., 2017]. Al igual que hemos visto para *ResNet*, *DenseNet* también cuenta con atajos, pero a diferencia del caso anterior, todas las capas están conectadas entre sí mediante bloques densos o *Dense Blocks*. Esto se debe a que esta red está diseñada bajo la premisa de que cuanto más cortas sean las conexiones entre las capas cercanas a la entrada y las cercanas a la salida, la red podrá ser entrenada de una forma más profunda y eficiente.

Como hemos visto hasta ahora, la mayoría de las versiones de las CNNs, contaban con un número de capas no muy elevado. Y es que con el paso del tiempo, el número de capas que utilizan este tipo de redes ha ido en aumento, debido a la mejora de la tecnología y al desarrollo de las propias redes neuronales convolucionales. Como veremos más adelante, todas las versiones de esta red cuentan con más de 100 capas.

DenseNet mantiene conectadas entre sí todas las capas que trabajan con *feature maps* del mismo tamaño, de tal forma que cada capa obtiene la información de todas las anteriores, y transmite su propia información a todas las siguientes. Más concretamente, cada capa de esta red tiene como entrada los mapas de características o *feature maps* de las capas que la preceden, que como se ha explicado en el apartado 4.1, este conjunto contiene diferentes rasgos de la imagen, conseguidos a través de la convolución. Si suponemos que nuestra red cuenta con L capas, la capa l, tendrá como entrada los *feature maps* de

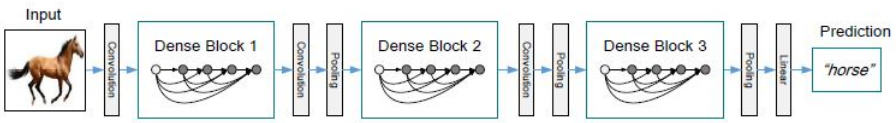

Figura 4.10: Ilustración gráfica de la arquitectura de la red *DenseNet*. Imagen de [Huang et al., 2017]

las primeras $l-1$ capas, y su salida irá a la entrada de las próximas $L-l$ capas. De esta forma, al contrario de *ResNet*, que suponiendo que cuenta con L capas, tendría L conexiones, *DenseNet* cuenta con $\frac{L(L+1)}{2}$ conexiones. Por esta razón, *DenseNet* es capaz de entrenar una cantidad de capas más alta, utilizando menos parámetros y sin sufrir el problema de desvanecimiento de gradiente.

Podemos ver un ejemplo visual del funcionamiento de este tipo de red en la imagen 4.10. Como en todos los casos, la entrada de la red es una imagen, en este caso la de un caballo. A continuación se le realiza una convolución de tamaño 7x7, y los mapas de características obtenidos se utilizan en el *Dense Block* o bloque denso número 1, donde todas las capas están conectadas entre sí. Dentro de este bloque, se aplican varias funciones como *batch normalization*, *ReLU* y distintas convoluciones, para calcular la transformación $H_l()$.

Como ya hemos visto, cada capa tiene la salida en función de los valores de salida de las capas que la preceden. Para calcular la salida de la capa l, conocida como x_l, se aplica la siguiente formula: $x_l = H_l([x_0, x_1, ..., x_{l-1}])$, donde $([x_0, x_1, ..., x_{l-1}])$ es la concatenación de los *feature maps* de las primeras *l-1* capas. Esta es una de las diferencias entre *DenseNet* y *ResNet*, ya que en la última, la salida de una capa se suma con la entrada de la siguiente, en vez de concatenarse.

Las siguientes dos fases de la red la componen una convolución de tamaño 1x1 y un *average pooling* de 2x2. Estas dos funciones hacen que el tamaño de los *feature maps* disminuya, haciendo que sea imposible el hecho de que todas las capas de la red estén conectadas entre sí, ya que el tamaño de los *feature maps* deben ser iguales. El conjunto de estas dos capas o fases, es conocido como *Transition Layers*.

De esta forma, se va repitiendo el proceso en el cual mediante las convoluciones y los *average poolings* se va disminuyendo el tamaño de los *feature maps*, y los bloques densos compuestos por las capas de la red procesan esa información y la acomodan. Así hasta llegar a la última capa, la cual aplica la función *softmax*, para al final hacer la predicción tal y como hemos visto en el apartado 4.1.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figura 4.11: Comparación de las diferentes versiones de *DenseNet*. Imagen de [Huang et al., 2017]

En la imagen 4.11 se pueden apreciar las distintas versiones de la arquitectura *DenseNet*. En todos los casos, la primera convolución es de tamaño 7x7 y se realiza un *max pooling* de 3x3 después. Además, los primeros dos bloques densos y todas las capas de transición (o *transition layers*) son iguales en todos los casos. La única diferencia consiste en el número de filtros que se aplican dentro de cada uno de los bloques densos, y esta es una de las ventajas respecto a las demás redes, ya que esta arquitectura cuenta con un número de filtros pequeño.

4.4.1. DenseNet-121

Como se representa en la imagen 4.11, hay cuatro versiones distintas de la red *DenseNet*. En este caso, se ha decidido utilizar la versión de 121 capas, ya que a pesar de ser la más pequeña de las cuatro, consigue buenos resultados, y al ser la menos profunda, no necesita tanto tiempo de entrenamiento. Por otro lado, al tener más de 100 capas, es bastante más profunda que todas las demás redes que se han decidido utilizar (*VGG-16*, *ResNet-34* y *ResNet-50*).

4.5. CNN simple

Por último tenemos la red neuronal convolucional simple, la cual ha sido creada desde cero. Como ya veremos en apartado 6, al ser la red más simple, esta será también la red

que peores resultados obtiene, pero el objetivo principal de este modelo no es superar los resultados de las demás redes, sino obtener unos resultados base, y ver como las demás redes (que parten de una idea similar a esta) mejoran esos resultados.

Como se puede ver en la imagen 4.12, la CNN está compuesta por 19 capas: 5 convoluciones, 5 funciones de activación *ReLU*, 5 *Max Pooling*, una capa *Flatten*, otra capa de *Dropout* y dos *Dense* (también conocidas como capas *fully conected*). Como se puede observar, el tamaño de entrada de las imágenes es de 256x256 píxeles, y ese tamaño se va reduciendo debido al *Max Pooling* que se da después de las convoluciones. Este proceso se repite hasta que el tamaño de las imágenes es de 8x8 píxeles. Por otro lado, tenemos 32 filtros por convolución, y a medida que avanzamos, este número se va duplicando, llegando hasta 512. Todas las convoluciones son de tamaño 3x3, y los *Max Pooling* en cambio, de 2x2. La capa *Flatten* se encarga de convertir la última capa oculta en un vector, para hacer que sea compatible con la capa de salida y hacer así la predicción. Mediante las capas *Dense*, se adecua el tamaño de las capas y se realizan cálculos para las predicciones. Es por esto que la última capa *Dense* tiene como parámetro el número de clases que constituye el modelo, ya que mediante la función de activación *softmax* se asignan los porcentajes a cada clase para la predicción. La capa de *Dropout* se utiliza para que la red 'olvide' o elimine cierta información utilizada para la predicción, ya que puede darse el caso de que la propia red aprenda a qué estilo pertenece cada obra, en vez de aprender cómo se representa un estilo y asignar obras a uno u otro. Este problema es conocido como *Overfitting*, y se puede detectar cuando el *accuracy* del entrenamiento sube, mientras que el de validación se estanca.

La red cuenta en total con 5.765.588 parámetros, por lo que es bastante más simple que los demás modelos que se han utilizado para este trabajo, y el desempeño que se obtiene por lo tanto es bastante peor. Para ambos casos (predicción de estilo y artista), se ha utilizado la misma red, al igual que el mismo número de *epochs* (10).

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 256, 256, 32)	896
activation_5 (Activation)	(None, 256, 256, 32)	0
max_pooling2d_5 (MaxPooling 2D)	(None, 128, 128, 32)	0
conv2d_6 (Conv2D)	(None, 128, 128, 64)	18496
activation_6 (Activation)	(None, 128, 128, 64)	0
max_pooling2d_6 (MaxPooling 2D)	(None, 64, 64, 64)	0
conv2d_7 (Conv2D)	(None, 64, 64, 128)	73856
activation_7 (Activation)	(None, 64, 64, 128)	0
max_pooling2d_7 (MaxPooling 2D)	(None, 32, 32, 128)	0
conv2d_8 (Conv2D)	(None, 32, 32, 256)	295168
activation_8 (Activation)	(None, 32, 32, 256)	0
max_pooling2d_8 (MaxPooling 2D)	(None, 16, 16, 256)	0
conv2d_9 (Conv2D)	(None, 16, 16, 512)	1180160
activation_9 (Activation)	(None, 16, 16, 512)	0
max_pooling2d_9 (MaxPooling 2D)	(None, 8, 8, 512)	0
flatten_1 (Flatten)	(None, 32768)	0
dropout_1 (Dropout)	(None, 32768)	0
dense_2 (Dense)	(None, 128)	4194432
dense_3 (Dense)	(None, 20)	2580

Total params: 5,765,588
Trainable params: 5,765,588

Figura 4.12: Estructura de la red neuronal convolucional simple utilizada para la predicción de estilo y artista.

5. CAPÍTULO

Datos utilizados

El conjunto de datos utilizado para este trabajo pertenece a *WikiArt*, y se utilizó para una competencia de *Kaggle*. Cuenta con 103.253 obras de arte diferentes, compuesta por 136 estilos artísticos y 2319 artistas. El conjunto de datos original tiene imágenes demasiado grandes, y debido a las limitaciones de almacenamiento y al excesivo tiempo de entrenamiento que conlleva trabajar con imágenes muy grandes, se utilizó otro *dataset* con las mismas imágenes, pero con un tamaño reducido. Más concretamente, las imágenes están escaladas para que siempre mantengan la forma, ya que el lado más pequeño de la imagen siempre tiene 256 píxeles, mientras que el otro varía para mantener la forma original de la imagen.

Como podemos ver en la figura 5.1, los 20 estilos más frecuentes están muy desequilibrados en el *dataset*; por ejemplo, hay estilos de arte como el *Impressionism*, el cual tiene más de 10.000 imágenes, mientras que otros como el *Art Informel* tienen alrededor de 1200 imágenes. Para que el conjunto de datos esté equilibrado, todos los estilos tendrán el mismo número de imágenes. Para solucionar esto, todos los estilos de arte tendrán la misma cantidad de imágenes que el estilo con el que menos cuenta, siendo esta de 1200. Para ello, se escogerán aleatoriamente 1200 obras por estilo. Se puede observar un ejemplo de cada estilo de arte en la figura 5.2. De esta forma, al contar con 20 estilos y 1200 imágenes por cada uno de ellos, las redes utilizarán en total 24.000 imágenes.

En el caso de los artistas, contamos con 2319, pero para tener el mismo número de estilos y artistas a predecir, tenemos que reducir bastante el conjunto. Para ello, nos quedaremos con los 20 artistas con más obras, todos ellos con más de 497. Este conjunto está formado

por 9987 obras, y como en el caso de los estilos hay 24.000, se ha pensado en hacer otro experimento, el cual se haría con un número de obras parecido al caso ya mencionado, para comparar la diferencia entre clasificar por estilo y por artista. En este supuesto caso, contaríamos con 54 artistas y 23.949 obras, y aunque el número de obras para la clasificación de artista y estilo sería bastante pareja, en el caso de los artistas el número de clases a predecir aumenta drásticamente a 54, por lo que no tiene mucho sentido hacer ese cambio, ya que lo que se aumenta en cantidad de imágenes (información) también se aumenta en número de clases, por lo que se sigue con una relación imágenes/clase muy parecida.

5.1. Preprocesamiento de los datos

Las imágenes son procesadas de manera previa al entrenamiento de los modelos. Para ello, se trabaja con el archivo *all_data_info.csv*. En él se encuentra todo tipo de información acerca de cada imagen, pero para acomodar los datos, solamente se ha utilizado el nombre de la imagen y el estilo (o artista) al que pertenece.

El primer paso consiste en agrupar todas las obras por estilo/artista, y contar cuantas obras hay de cada uno. Después se han ordenado las categorías de mayor a menor en cantidad de obras, ya que contamos con una tabla de tres columnas: el estilo/artista, el nombre de la imagen y la cantidad de obras por estilo/artista. Una vez ordenado, se filtra según la cantidad de imágenes que componen cada estilo/artista. Como ya hemos visto, nos quedaremos con 20 estilos y 20 artistas, cada uno de ellos compuesto por al menos 1200 y 497 imágenes respectivamente. Por último, nos quedaremos con 1200 imágenes aleatorias de cada estilo (497 para los artistas), haciendo que todos los estilos (y artistas) tengan el mismo número de imágenes y el dataset esté equilibrado. Una vez realizado este proceso, ya podremos empezar con el entrenamiento de los modelos.

Por otro lado, tenemos el caso de la predicción de estilo con el añadido de que varios estilos artísticos están fusionados en uno solo, pasando de 20 a 13 estilos a clasificar. En este otro caso, se ha hecho una copia del archivo *all_data_info.csv* y se ha cambiado el nombre de los estilos que se han fusionado, para después procesarlos como en el caso anterior: ordenar las imágenes según la cantidad de obras por estilo, e igualar el número de obras de cada estilo a la cantidad de obras que tiene el estilo con menos obras, escogiendo imágenes al azar.

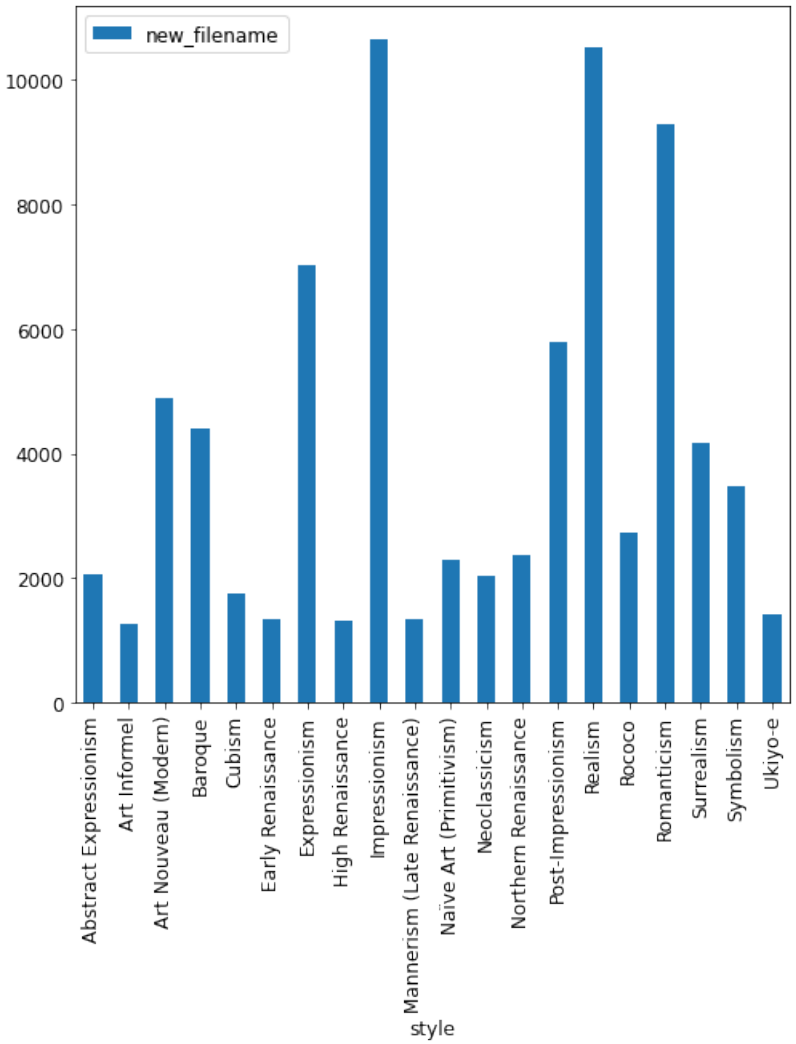

Figura 5.1: Histograma que muestra el número de imágenes de los estilos artísticos utilizados, antes de que se haga una selección de muestras para que todos los estilos tengan el mismo número de imágenes.

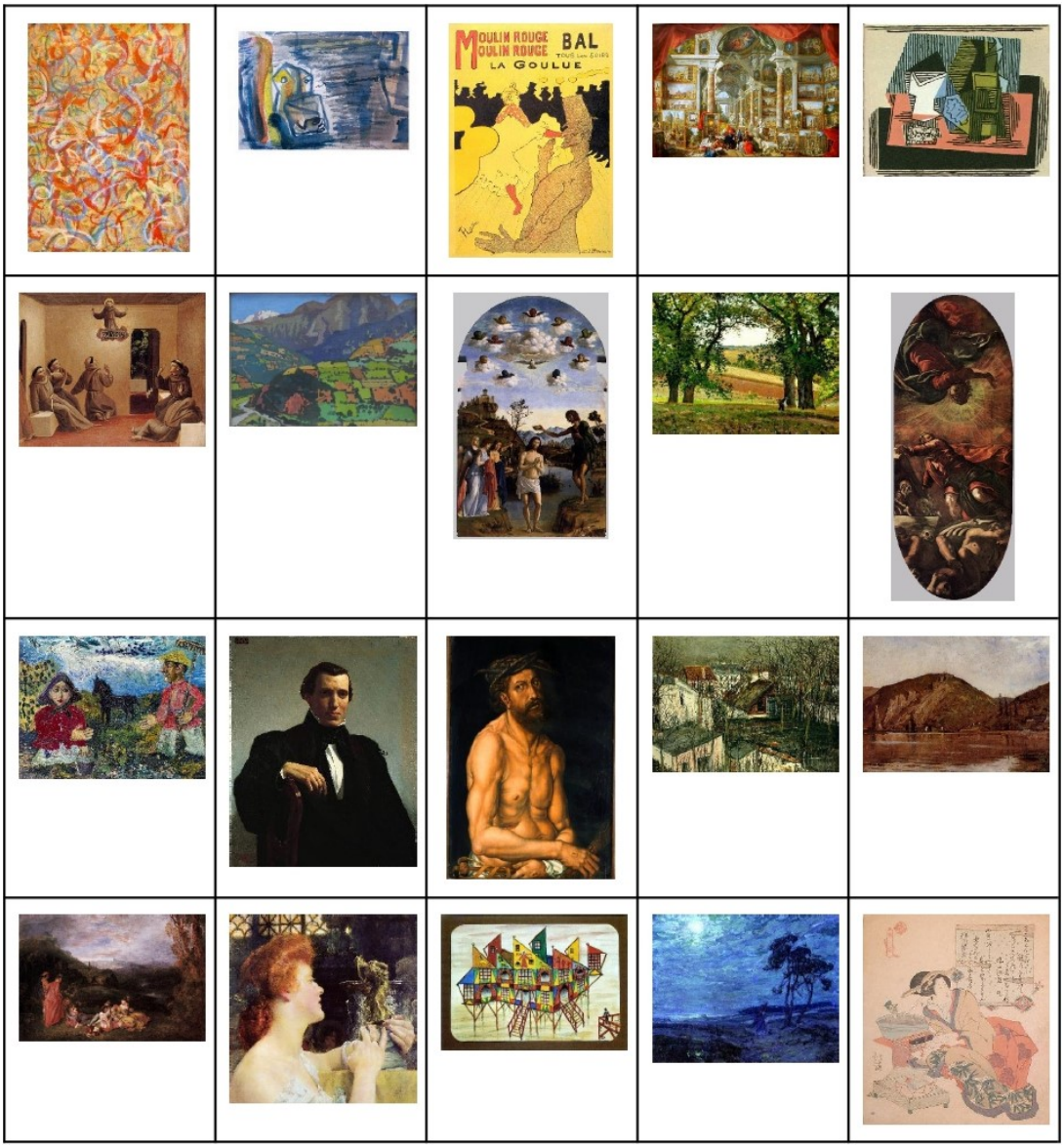

Figura 5.2: Imágenes de ejemplo para cada estilo artístico. De izquierda a derecha y de arriba a abajo: ”Abstract Expressionism, Art Informel, Art Nouveau (Modern), Baroque, Cubism, Early Renaissance, Expressionism, High Renaissance, Impressionism, Mannerism (Late Renaissance), Naïve Art (Primitivism), Neoclassicism, Northern Renaissance, Post-Impressionism, Realism, Rococo, Romanticism, Surrealism, Symbolism y Ukiyo-e”.

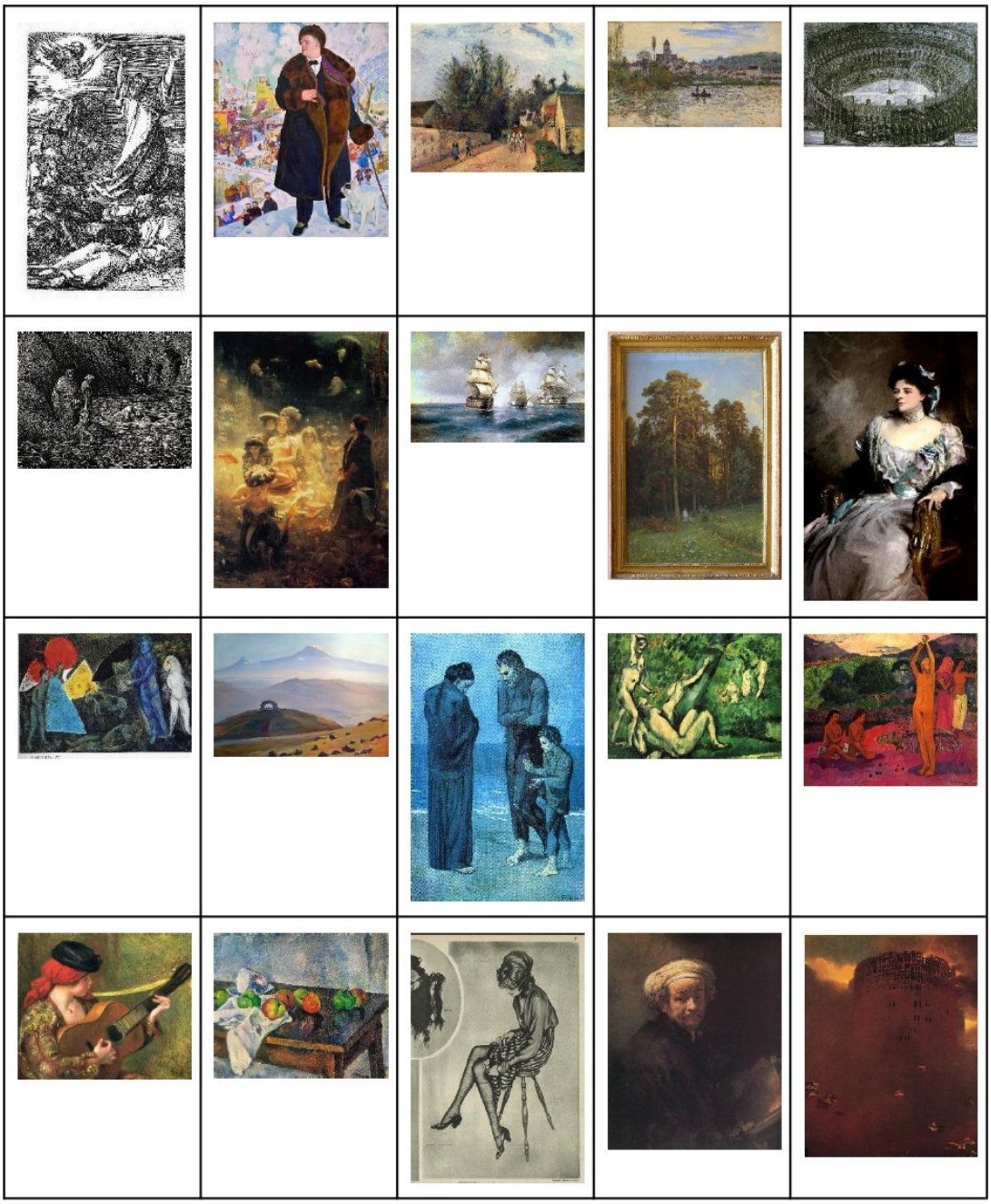

Figura 5.3: Imágenes de ejemplo para cada artista del dataset compuesto por 20 autores. De izquierda a derecha y de arriba a abajo: ”Albrecht Durer, Boris Kutsodiev, Camille Pissarro, Claude Monet, Giovanni Battista Piranesi, Gustave Dore, Ilya Repin, Ivan Aivazovsky, Ivan Shishkin,John Singer Sargent, Marc Chagall, Martiros Saryan, Pablo Picasso, Paul Cezanne, Paul Gauguin, Pierre-Auguste Renoir, Pyotr Konchalovsky, Raphael Kirchner, Rembrandt y Zdislav Beksinski.”

6. CAPÍTULO

Pruebas realizadas y resultados obtenidos

Se han realizado varios experimentos y variaciones a la hora de hacer predicciones, tanto de artista como de estilo. Dentro de estas variaciones, se han cambiado parámetros como el tamaño de las imágenes, el número de estilos y artistas, y hasta se ha probado juntar o fusionar ciertos estilos en uno solo (siempre y cuando esos estilos estén históricamente muy relacionados, tal y como se explica en el apartado 6.1.3).

6.1. Experimentos

A continuación se explican los diferentes experimentos realizados, detallando cuales son las diferencias entre ellos, los resultados obtenidos (representados a través de funciones de pérdida, *accuracy*, tiempo de ejecución, matrices de confusión y gráficos) y su explicación, al igual que algunos razonamientos y explicaciones de por qué se han hecho dichos experimentos. Se puede encontrar un resumen de los experimentos en la tabla 6.1.

6.1.1. Imágenes grandes

Como ya se ha mencionado en el apartado 2.2 de problemas e improvistos, las imágenes que fundamentan el dataset original son demasiado grandes y ocupan mucho espacio. Además, si contamos con que el dataset cuenta con 103.253 imágenes, el trabajo que se necesita para procesar los datos es muy costoso. A pesar de todo, se ha hecho una pequeña prueba con la red *ResNet-50* para predecir los estilos que cuentan con más de

Tipo de Predicción	Número de clases	Tamaño de las imágenes
Estilo	10	Grande
Estilo	33	Grande
Estilo	20	Reducido
Artista	20	Reducido
Estilos fusionados	13	Reducido
Estilo (mejora)	20	Reducido
Artista (mejora)	20	Reducido

Tabla 6.1: Tabla que muestra los distintos experimentos realizados con sus características.

500 imágenes, siendo 33 en total. Se pueden ver los resultados obtenidos en la tabla 6.2. Como era de esperar, el hecho de que las imágenes sean tan grandes hace que el tiempo que necesita la red por *epoch* sea demasiado alto (llegando a un total de más de 5 horas para cuatro *epochs*), pero a pesar de que haya muchos estilos, el acierto obtenido en 4 iteraciones es bastante bueno. Por otro lado, se ha probado reduciendo el número de estilos, y en vez de contar con los que tienen más de 500 imágenes, se han elegido los que tienen más de 2500, siendo 10 en total. Al contar con bastantes menos estilos, el acierto obtenido es mayor como era de esperar, pero el tiempo que necesita la red sigue siendo demasiado alto. Se pueden observar los resultados en la tabla 6.3. No se ha realizado la predicción de artista con imágenes grandes debido al excesivo tiempo que necesita la red para ser entrenada.

Además hay que tener en cuenta que estos experimentos se realizaron en la aplicación de *Jupyter Notebook* en un ordenador personal y mediante CPU. En cambio, las siguientes pruebas se han hecho mediante *Google Colab*, el cual contiene más y mejores recursos, a la vez que se utiliza GPU (la cual es más recomendada que la CPU para procesar imágenes), reduciendo mucho los tiempos. También hay que tener en cuenta que este otro entorno tiene limitaciones de tiempo, y además, según la cantidad de usuarios que haya simultáneamente, los recursos asignados varían, haciendo cambiar también a los resultados (sobre todo cambiando el tiempo de ejecución), por lo que no siempre *Google Colab* es la mejor opción.

6.1.2. Tamaño de las imágenes reducido

Para reducir el tiempo que necesita la red en entrenar, se ha utilizado el mismo dataset pero con las imágenes reducidas, el cual se encuentra en `https://github.com/somewacko/painter-by-numbers/releases/tag/data-v1.0`. Todas ellas están escaladas sin ser

Epoch	Accuracy	Tiempo
0	0.4024	1:17:45
1	0.4666	1:18:58
2	0.5040	1:19:23
3	0.5174	1:18:33

Tabla 6.2: Tabla que muestra la evolución del entrenamiento de la red *ResNet-50* en la predicción de 33 estilos con las imágenes de tamaño original.

Epoch	Accuracy	Tiempo
0	0.4705	50:39
1	0.5199	50:16
2	0.5499	50:17
3	0.5512	50:12

Tabla 6.3: Tabla que muestra el acierto y el tiempo que tiene la red *ResNet-50* en la predicción de 10 estilos con las imágenes de tamaño original.

deformadas, ya que una de sus dimensiones (la dimensión más pequeña) siempre mide 256 píxeles y la otra varía en función de la imagen, para que esta mantenga la proporción original. Este caso es el que se ha tenido en cuenta como caso principal, ya que se han realizado predicciones para estilo y artista.

Por un lado, tenemos las pruebas realizadas con las redes *VGG-16*, *ResNet-34*, *ResNet-50* y *DenseNet-121*, y por otro lado, el modelo CNN creado desde cero. Para saber a qué estilo pertenece cada imagen del dataset, contamos con el archivo *all_data_info.csv*, el cual contiene toda la información necesaria acerca de cada obra. Cada fila del archivo contiene información acerca de una imagen, más concretamente, el nombre del artista, el estilo de la obra, el año en el que fue creada, el título, el género, el nombre del archivo y más información, como el tamaño de la imagen en píxeles o el tamaño que ocupa la imagen en bytes.

Una vez se han acomodado los datos, el siguiente paso es entrenar cada modelo. Para ello se utiliza la función *vision_learner*, con el nombre del modelo como parámetro. Todos los modelos han sido entrenados durante 10 *epochs*, ya que como veremos más adelante, una vez pasadas esas iteraciones, varios modelos tienden a mantener el *accuracy* o a veces hasta disminuye. De esta forma, se mantienen todos en las mismas condiciones, y es una alternativa para comparar la eficiencia de los modelos de una manera más equitativa. En el caso de la red CNN creada desde cero también se han utilizado 10 *epochs*.

En todos los casos, se han tenido en cuenta detalles como el *accuracy*, el tiempo por

epoch, la pérdida en los casos de entrenamiento y validación y una matriz de confusión, para ver los errores más comunes que comete el modelo y buscar una explicación. Además todos los modelos se han entrenado bajo las mismas condiciones para compararlos de una forma más directa.

Reconocimiento de estilo

Como se ha visto, utilizaremos cinco modelos distintos para la predicción de estilo de obras de arte: *VGG-16, ResNet-34, ResNet-50, DenseNet-121* y una CNN simple creada desde cero. Además, tal y como se ha mencionado anteriormente, se han tenido en cuenta los 20 estilos que más imágenes tienen en el dataset: *Abstract Expressionism, Art Informel, Art Nouveau (Modern), Baroque, Cubism, Early Renaissance, Expressionism, High Renaissance, Impressionism, Mannerism (Late Renaissance), Naïve Art (Primitivism), Neoclassicism, Northern Renaissance, Post-Impressionism, Realism, Rococo, Romanticism, Surrealism, Symbolism* y *Ukiyo-e*.

Todas las imágenes tienen 256x256 píxeles, ya que ese es un buen tamaño de imagen para que lo procesen los modelos. Para eso, se recorta un trozo de ese tamaño desde el centro de la imagen, ya que en la mayoría de las obras, lo más característico e importante se encuentra en esa zona. A diferencia de la mayoría de los trabajos, no se realiza *Data Augmentation*, ya que esto consiste en añadir imágenes nuevas al dataset, a partir de modificar las originales con rotaciones o cambios de brillo por ejemplo. Aunque añadir más información al conjunto de datos puede ser beneficioso, ciertos cambios en la imagen pueden producir cambios de estilo, y eso puede hacer confundir a la red. El conjunto cuenta con 20 estilos, con 1200 imágenes por cada uno de ellos, por lo que en total son 24.000 imágenes.

A continuación se explican los resultados obtenidos por los modelos, y se comparan entre sí:

1. ***VGG-16***: Como ya se ha visto, esta red es una de las más simples utilizadas en este trabajo, ya que fue una de las primeras redes neuronales convolucionales utilizada en la clasificación de imágenes. En la tabla 6.4 se encuentran los resultados que se han obtenido con el entrenamiento de esta red. En ella se puede observar que el *accuracy* más alto se obtiene en la novena *epoch*, siendo este del 50.12 %, y que en la décima vuelve a disminuir. Por lo tanto, *VGG-16* es capaz de predecir correctamente el estilo de una obra en la mitad de los casos.

Por otro lado, en la imagen 6.3, se encuentra la matriz de confusión obtenida, la cual muestra en qué porcentaje ha confundido el modelo un estilo con otro, siendo las líneas verticales los estilos a los que pertenecen realmente las obras, y las horizontales las predicciones realizadas por el modelo. Por lo que la posición (x, y) de la matriz muestra el porcentaje de que una obra de estilo x haya sido predicha como estilo y (en realidad muestra el número de veces que se ha confundido un estilo, pero se ha normalizado entre 0 y 1). Por otro lado, la diagonal principal de la matriz muestra el porcentaje de acierto para cada estilo. En ella podemos ver que hay bastante confusión entre *Abstract Expressionism* y *Art Informel*, y por ejemplo estilos como *Expressionism*, *Realism* o *Symbolism* son los más equivocados a la hora de clasificarlos, mientras que el *Ukiyo-e* es el que mayor acierto de clasificación tiene por diferencia.

2. ***ResNet-34***: Esta otra red, surge de la idea errónea de que añadiendo más capas a una red neuronal esta aumentará su eficacia. Para solucionar esto se utilizan los atajos y bloques residuales. Esta versión de *ResNet* cuenta con 21.8 millones de parámetros, bastantes menos que la red anterior (*VGG-16* tiene cerca de 138 millones de parámetros). En la tabla 6.5 se pueden observar los resultados obtenidos por esta red. El *accuracy* obtenido es bastante similar al de *VGG-16*, consiguiendo en el mejor de los casos 50.72 % de acierto. Pero la mayor diferencia entre estas dos redes es el tiempo. Si bien *VGG-16* necesita alrededor de 7 minutos por *epoch*, *ResNet-34* necesita menos de la mitad, siendo bastante más efectiva en este aspecto.

   En la figura 6.4 tenemos a la matriz de confusión para la predicción de estilo. Al igual que en el caso anterior, los estilos *Art Informel* y *Abstract Expressionism* son confundidos en muchos casos, mientras que otros como *Cubism* y *Ukiyo-e* están muy bien catalogados, y el *Expressionism* es el que más error de clasificación tiene otra vez.

3. ***ResNet-50***: Por otro lado, tenemos la versión de 50 capas de *ResNet*, *ResNet-50*. Los resultados del entrenamiento para la clasificación de estilos se pueden encontrar en la tabla 6.6. En este caso, el *accuracy* aumenta respecto a la otra versión de *ResNet*, consiguiendo un 54.72 %, pero también aumenta en tiempo, por lo que según las necesidades (acierto o tiempo) elegiremos una u otra. Respecto a *VGG-16*, la supera en ambos aspectos, deduciendo que *ResNet-50* es mejor en la predicción de estilo.

   En cuanto a su matriz de confusión, la podemos encontrar en la imagen 6.5. Muy similar a las anteriores, *Ukiyo-e* sigue siendo el estilo que mejor se predice, mien-

tras que el peor es el *Expressionism*. Además, tanto *Abstract Expressionism* y *Art Informel* siguen siendo muy confundidos.

4. ***DenseNet-121***: En el caso de *DenseNet-121* nos encontramos en un caso con pros y contras bastante claros. Sus resultados se encuentran en la tabla 6.7, y como se puede observar es la red que mejor *accuracy* obtiene, con un 56.25 %, y hay que tener en cuenta que a diferencia de las demás redes, el accuracy de esta no se ha estancado en ningún momento, sino que iba en aumento, por lo que aumentando el número de *epochs* es posible que su eficacia sea aun mayor. Pero por otro lado, es la red que más tiempo ha necesitado para completar el entrenamiento, estando en varios casos cerca de los 8 minutos por *epoch*.

   En la figura 6.6 se encuentra su matriz de confusión, y los valores que se representan en ella vuelven a dar unos resultados muy parecidos, por lo que parece que todas las redes encuentran las mismas dificultades y facilidades a la hora de clasificar las obras, solo que en mayor o menor medida y con distintos tiempos.

5. ***CNN simple***: Por último, nos encontramos ante la red más simple de todo el trabajo. Esta CNN es la que peor *accuracy* obtiene, pero a la vez es la red más rápida de las utilizadas, ya que necesita menos de dos minutos por *epoch*. La tabla 6.8 muestra estos resultados, y como se puede observar, su acierto es bastante mejorable. Por otro lado, en la figura 6.1 podemos ver como progresa el *accuracy* del modelo en función de los *epochs*, viendo en azul el acierto del entrenamiento y en naranja el de test. Además, en la figura 6.2 encontramos el mismo gráfico pero esta vez se muestra el *loss* en vez del *accuracy*. Como se puede observar en ambas imágenes, la línea azul (la cual indica entrenamiento) tiende a seguir subiendo para el *accuracy* y a bajar en el *loss*, mientras que la línea de test (naranja) parece llegar a una especie de límite y se mantiene más o menos constante. A este fenómeno se le llama *overfitting* o sobreajuste, y se resume en que el modelo se está sobreentrenando, y de esta forma en la fase de entrenamiento se están aprendiendo unas características no muy eficaces. Para solucionar este problema, suelen utilizarse capas de *Dropout*, las cuales eliminan aleatoriamente un porcentaje de información acerca de las predicciones. Otra alternativa para minimizar este problema consiste en añadir más datos al dataset.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	3.0062	2.2130	0.3075	7:17
1	2.2699	1.8342	0.4037	6:54
2	1.9129	1.7102	0.4345	6:54
3	1.7666	1.6488	0.4531	6:54
4	1.6737	1.5774	0.4768	6:54
5	1.5744	1.5375	0.4852	6:54
6	1.5257	1.5193	0.4833	6:54
7	1.4488	1.4891	0.5008	6:54
8	1.4411	1.4834	0.5012	6:54
9	1.3402	1.4834	0.4989	6:55

Tabla 6.4: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *VGG-16* por *epoch* en la predicción de 20 estilos.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	3.0026	2.2123	0.3220	3:50
1	2.2783	1.8614	0.3968	3:22
2	1.9485	1.7272	0.4375	3:21
3	1.8179	1.6514	0.4541	3:22
4	1.6874	1.5778	0.4735	3:21
5	1.5720	1.5281	0.4947	3:21
6	1.5090	1.5043	0.4995	3:21
7	1.3687	1.4994	0.4997	3:21
8	1.3697	1.4910	0.5018	3:20
9	1.3476	1.4867	0.5072	3:20

Tabla 6.5: Tabla que muestra la función de pérdida en los conjuntos de entrenamiento y validación, el acierto y el tiempo que necesita la red *ResNet-34* por *epoch* en la predicción de 20 estilos.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	2.6595	2.0677	0.3691	5:04
1	2.1162	1.7589	0.4300	4:37
2	1.8674	1.6143	0.4679	4:37
3	1.6277	1.5581	0.4883	4:36
4	1.5437	1.4809	0.5102	4:35
5	1.4145	1.4327	0.5272	4:35
6	1.2603	1.3986	0.5345	4:35
7	1.1871	1.3746	0.5466	4:36
8	1.1023	1.3717	0.5472	4:35
9	1.0799	1.3654	0.5460	4:35

Tabla 6.6: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *ResNet-50* por *epoch* en la predicción de 20 estilos.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	2.6612	1.9315	0.4052	7:55
1	2.1716	1.7288	0.4385	7:38
2	2.0067	1.6247	0.4685	7:39
3	1.8577	1.5772	0.4787	7:52
4	1.7665	1.5104	0.5010	7:53
5	1.5894	1.4507	0.5181	7:49
6	1.4433	1.3881	0.5320	7:37
7	1.3294	1.3794	0.5468	7:37
8	1.2517	1.3605	0.5545	7:37
9	1.3169	1.3410	0.5625	7:37

Tabla 6.7: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *DenseNet-121* por *epoch* en la predicción de 20 estilos.

Epoch	Train_loss	Val_loss	Train_acc	Val_acc	Tiempo
0	2.9021	2.7532	0.0978	0.1645	1:11
1	2.6928	2.5890	0.1648	0.1992	1:08
2	2.5702	2.4934	0.2027	0.2322	1:07
3	2.4650	2.4467	0.2305	0.2435	1:07
4	2.3609	2.3368	0.2609	0.2778	1:07
5	2.2794	2.3281	0.2914	0.2639	1:13
6	2.1918	2.2861	0.3195	0.3010	1:07
7	2.1013	2.2573	0.3469	0.2930	1:07
8	1.9865	2.2625	0.3780	0.3001	1:07
9	1.8794	2.2713	0.4118	0.2908	1:07

Tabla 6.8: Tabla que muestra la pérdida y acierto de entrenamiento y validación, y el tiempo que necesita el modelo CNN creado desde cero por *epoch* en la predicción de 20 estilos.

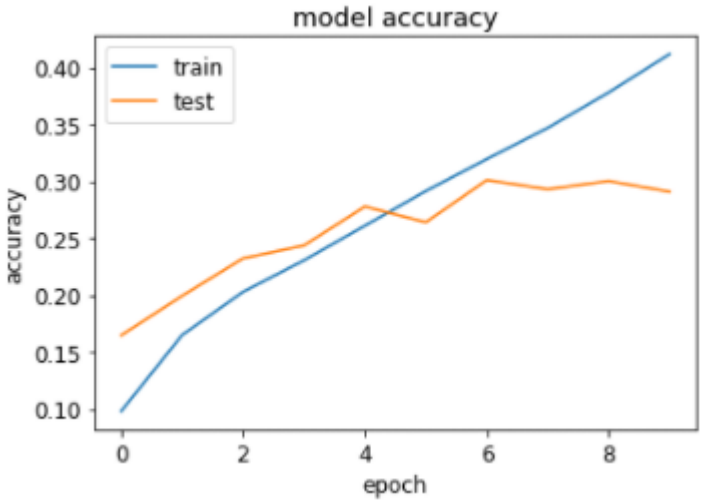

Figura 6.1: Gráfico que muestra la evolución de los *accuracy* de entrenamiento y validación por *epoch* para la predicción de estilos, utilizando la CNN creada desde cero.

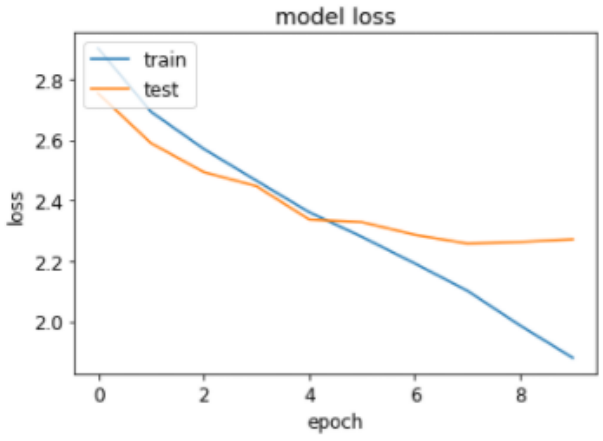

Figura 6.2: Gráfico que muestra la evolución de los *loss* (o pérdida) de entrenamiento y validación por *epoch* para la predicción de estilos, utilizando la CNN creada desde cero.

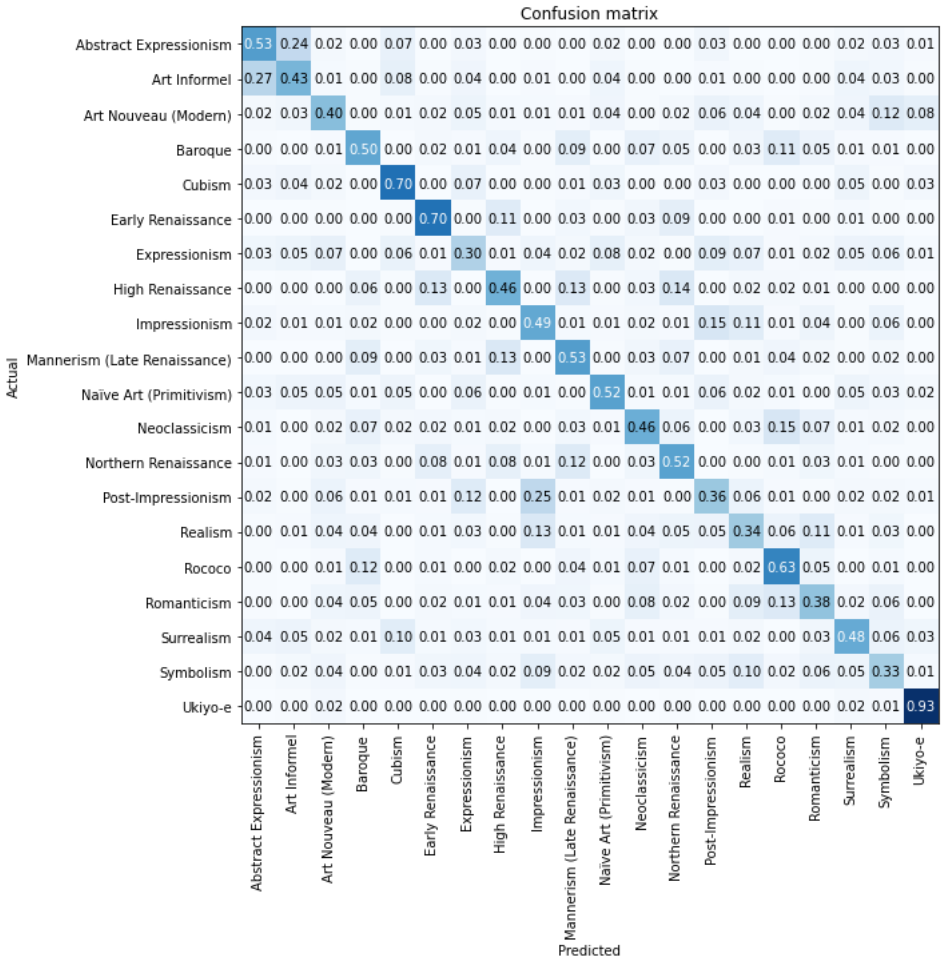

Figura 6.3: Matriz de confusión de las predicciones de estilo para *VGG-16*.

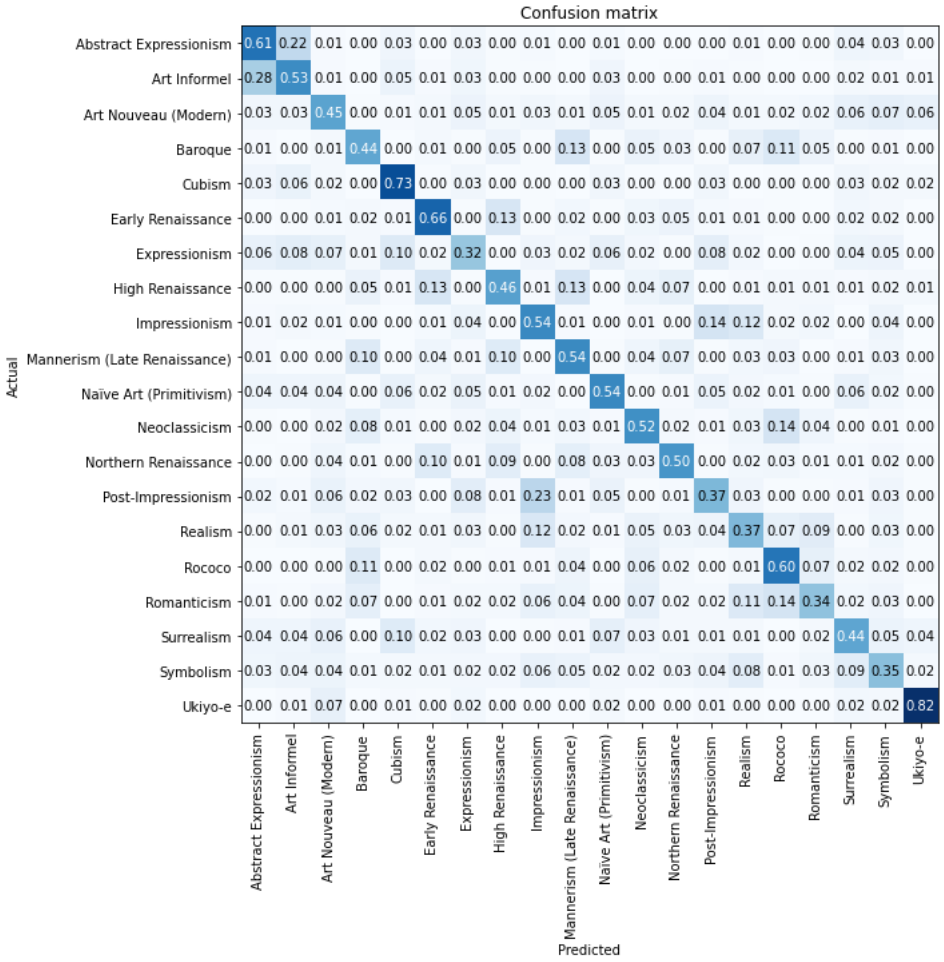

Figura 6.4: Matriz de confusión de las predicciones de estilo para *ResNet-34*.

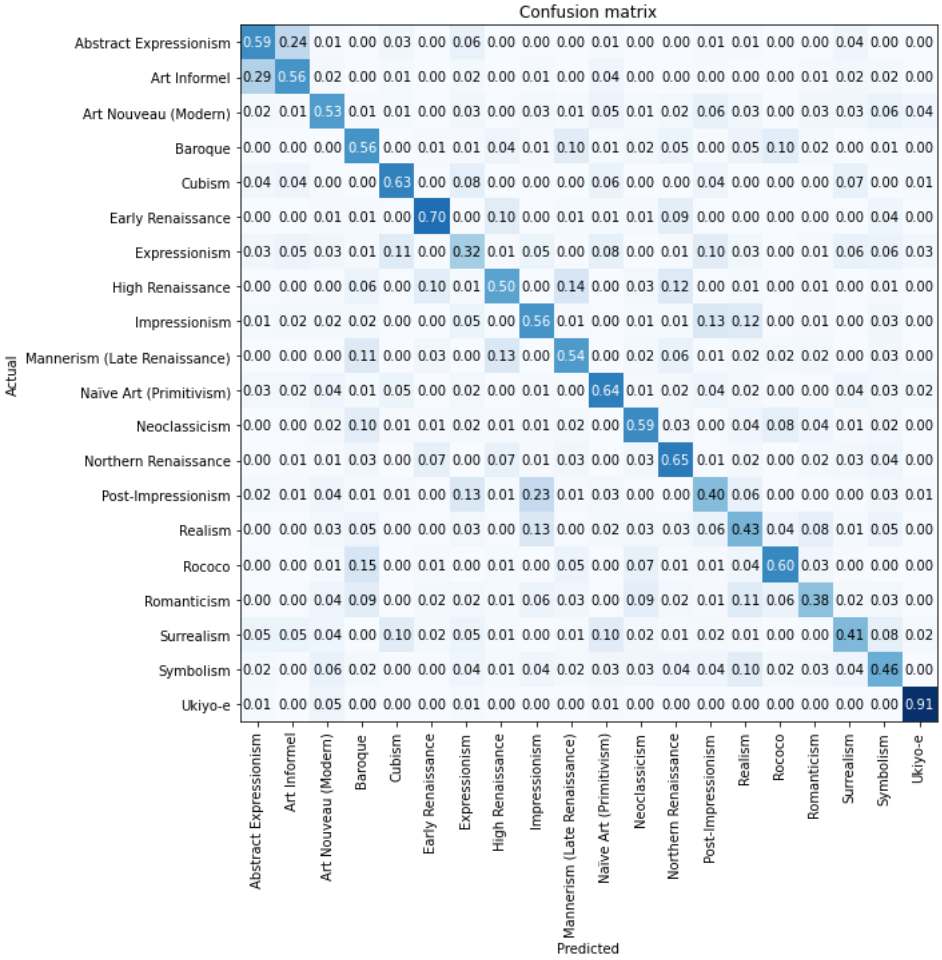

Figura 6.5: Matriz de confusión de las predicciones de estilo para *ResNet-50*.

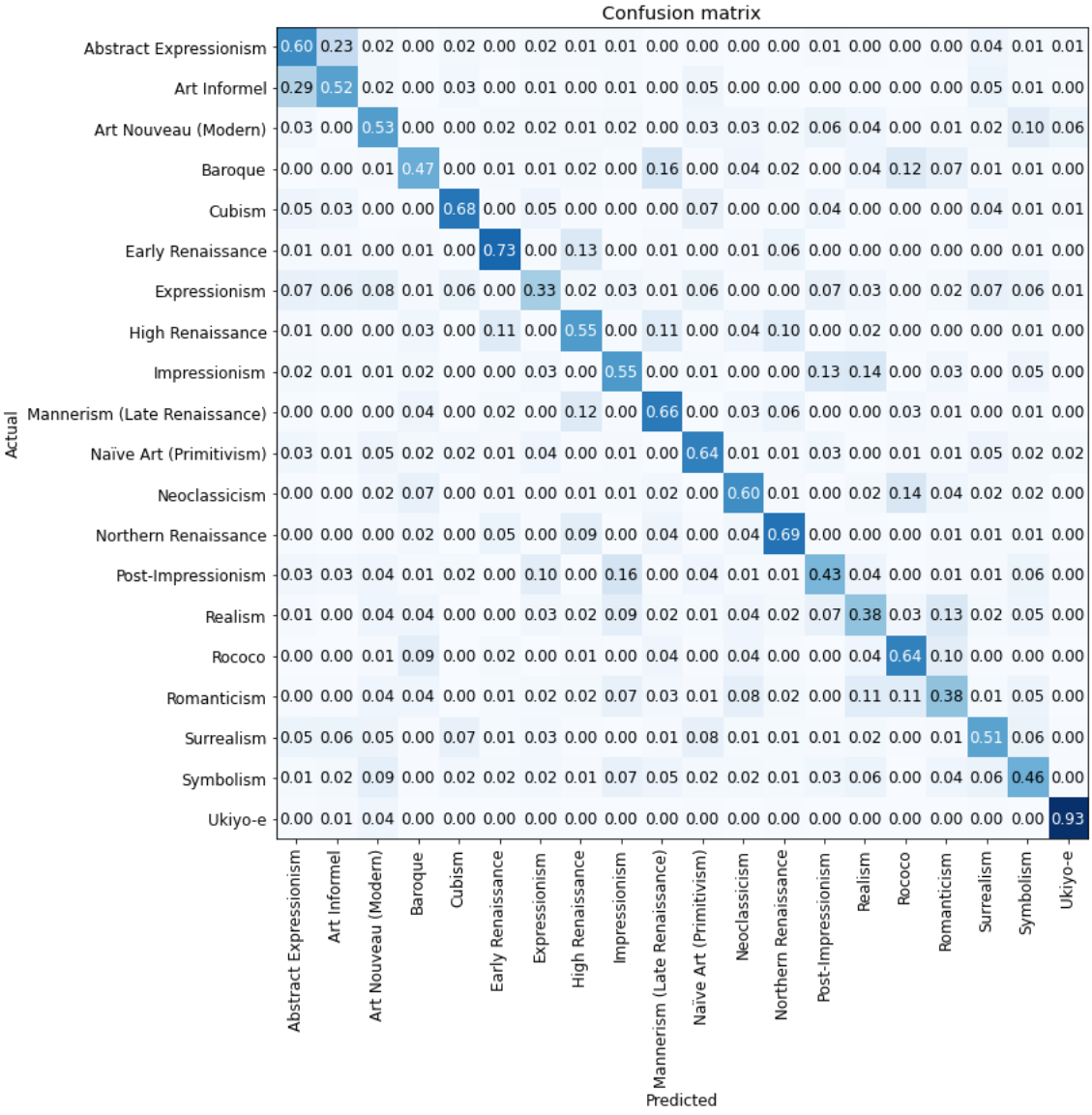

Figura 6.6: Matriz de confusión de las predicciones de estilo para *DenseNet-121*.

Reconocimiento de artista

En el caso de predicción de artista, también se ha probado utilizando a los 20 que más obras tienen en el conjunto de datos, formados por 9987 obras. Más concretamente, contamos con los siguientes artistas: "*Ivan Aivazovsky*, *Gustave Dore*, *Rembrandt*, *Claude Monet*, *Pierre-Auguste Renoir*, *Albrecht Durer*, *Ivan Shishkin*, *Giovanni Battista Piranesi*, *Raphael Kirchner*, *Paul Cezanne*, *John Singer Sargent*, *Zdislav Beksinski*, *Camille Pissarro*, *Boris Kutsodiev*, *Ilya Repin*, *Martiros Saryan*, *Pyotr Konchalovsky*, *Pablo Picasso*, *Marc Chagall* y *Paul Gauguin*". A continuación se muestran los resultados que se han conseguido para la predicción de artista con 20 casos:

1. ***VGG-16***: Como se ha visto anteriormente, *VGG-16* es una de las redes más simples utilizadas en es modelo, pero comparado con los resultados obtenidos en la predicción de estilo, la predicción de artista consigue unos resultados mucho mejores (ver tabla 6.9, consiguiendo como máximo un *accuracy* del 82.13 %. Además, al tener menos imágenes en el dataset, el tiempo por *epoch* disminuye bastante.

   Además, en la figura 6.9, encontramos la matriz de confusión para la predicción de artista (con 20 artistas). Como los resultados son muy buenos, la mayoría de los casos están concentrados en la diagonal principal de la matriz. Los artistas con mayor acierto son *Giovanni Battista Piranesi* y *Ivan Aivazovsky*, ambos con un 98 %, y el más confundido *Boris Kustodiev* con un 61 % de acierto, el cual se ha confundido en un 12 % de los casos con *Ilya Repin*, que fue maestro suyo .

2. ***ResNet-34***: En el caso de *ResNet-34* el *accuracy* es también bastante bueno, 80,92 % de acierto en el mejor de los casos. Aunque se encuentra un poco por debajo del acierto de *VGG-16*, al igual que en la predicción de artista, esta vez su tiempo por *epoch* es bastante bueno, llegando a la mitad del tiempo obtenido por la red *VGG-16*. Estos datos se encuentran en la tabla 6.10.

   Por otro lado, podemos encontrar la matriz de confusión para este caso en la imagen 6.10. En ella podemos ver que el artista con mayor confusión vuelve a ser *Boris Kustodiev*, con un 62 % de acierto, mientras que el artista con más acierto es *Giovanni Battista Piranesi* con un 100 % de acierto.

3. ***ResNet-50***: Esta otra versión de *ResNet*, vuelve a conseguir mejores resultados que su versión de 34 capas, tal y como se puede ver en la tabla 6.11. El *accuracy* más alto lo obtiene en la *epoch* número siete, con un 83,82 %. Ambas versiones de *ResNet* parecen atascarse en *accuracy* antes de acabar el entrenamiento, en *ResNet-34*

apenas sube un 0.5 % de acierto en las 4 últimas *epochs*, mientras que en *ResNet-50* el acierto máximo se consigue en la séptima *epoch* y despues disminuye.

En la matriz de confusión de la imagen 6.11 podemos observar que el artista más confundido es *Paul Gauguin*, que ha sido clasificado correctamente en un 64 % de los casos. Cabe destacar que en el 19 % de los casos, este ha sido confundido con *Camille Pissarro*. Por otro lado, artistas como *Ivan Aivazovsky*, *Gustave Dore* y *Giovanni Battista Piranesi* han obtenido un porcentaje de acierto de 97 %, 97 % y 100 % respectivamente.

4. ***DenseNet-121***: La red *DenseNet* vuelve a ser la red que mejor *accuracy* obtiene, con un 85.77 % en el mejor de los casos. Por otro lado, el tiempo que necesita por *epoch* es bastante menor que el que necesita en la predicción de artista, siendo el doble de rápida en este caso. Al igual que en el caso de clasificar artistas, los resultados obtenidos reflejados en la tabla indican que la red no ha llegado a un límite, indicando una posible mejora de los resultados al añadir más iteraciones.

   En la imagen 6.12 tenemos a la matriz de confusión obtenida tras el entrenamiento de la red *DenseNet-121*. En ella podemos ver que es *Gustave Dore* quien obtiene el mejor porcentaje de clasificación con un 98 %, y tras el está *Giovanni Battista Piranesi* con 97 % de acierto. En cambio, es *Ilya Repin* el artista con menos acierto con un 71 %.

5. ***CNN simple***: En cuanto a la red CNN creada desde cero, hay que decir que mejora bastante en *accuracy* al igual que las demás redes en la predicción de artista, llegando a tener como máximo un 61,70 % de acierto en el entrenamiento, y un 56,45 % en la validación, tal y como se puede apreciar en la tabla 6.13. Por otro lado, también es la red más rápida (consiguiendo menos de un minuto por *epoch*), tal y como lo era en la predicción de estilo. En la figura 6.7 se muestra el progreso del acierto en función del *epoch* de la ejecución, mostrando en naranja el acierto de test y en azul el de entrenamiento. En cuanto a la evolución del *loss*, el gráfico que muestra su evolución aparece en la imagen 6.8. En ambos casos se puede observar que las cuervas no llegan a un límite definido, aunque se puede apreciar una curvatura, la cual indica que aumentando el número de *epochs*, el *accuracy* del modelo aumentaría hasta llegar a cierto punto. Como ambas curvas tienen una forma muy parecida, se puede deducir que no hay *Overfitting*, que como ya se ha visto, indica que el modelo se aprende las predicciones en el entrenamiento, y su desempeño no se ve reflejado en el apartado de validación.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	2.3342	1.4158	0.5774	4:42
1	1.6097	0.9944	0.7102	4:34
2	1.3500	0.8568	0.7507	4:34
3	1.2047	0.7960	0.7608	4:34
4	1.0513	0.7392	0.7792	4:34
5	0.9245	0.6917	0.7934	4:34
6	0.7912	0.6361	0.8045	4:34
7	0.7017	0.6246	0.8161	4:34
8	0.6363	0.5996	0.8213	4:34
9	0.6296	0.5933	0.8203	4:34

Tabla 6.9: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *VGG-16* por *epoch* en la predicción de 20 artistas.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	2.2964	1.4363	0.5763	2:06
1	1.6121	0.9396	0.7149	2:01
2	1.3042	0.8883	0.7191	2:00
3	1.2623	0.8111	0.7413	2:02
4	1.0087	0.7144	0.7876	2:04
5	0.9112	0.6731	0.7982	2:02
6	0.7900	0.6433	0.8040	2:01
7	0.6671	0.6198	0.8071	2:00
8	0.6971	0.6252	0.8092	2:00
9	0.6452	0.6261	0.8092	2:00

Tabla 6.10: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *ResNet-34* por *epoch* en la predicción de 20 artistas.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	1.8577	1.2217	0.6480	3:06
1	1.4081	1.1311	0.6801	3:00
2	1.2364	0.8745	0.7397	2:56
3	0.9165	0.7382	0.7760	2:57
4	0.8708	0.6996	0.7913	2:56
5	0.7312	0.6140	0.8192	2:56
6	0.6217	0.5749	0.8382	3:00
7	0.5120	0.5733	0.8377	3:02
8	0.4150	0.5523	0.8345	3:01
9	0.3843	0.5466	0.8350	2:56

Tabla 6.11: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *ResNet-50* por *epoch* en la predicción de 20 artistas.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	1.8380	1.0734	0.6849	3:20
1	1.3219	0.8458	0.7407	3:02
2	1.0568	0.7562	0.7723	3:04
3	0.9705	0.6510	0.7968	2:59
4	0.7806	0.6199	0.8134	3:02
5	0.7186	0.5338	0.8329	3:07
6	0.5648	0.5055	0.8429	3:07
7	0.4684	0.4743	0.8461	2:59
8	0.3637	0.4605	0.8551	2:56
9	0.3821	0.4642	0.8577	2:56

Tabla 6.12: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *DenseNet-121* por *epoch* en la predicción de 20 artistas.

Epoch	Train_loss	Val_loss	Train_acc	Val_acc	Tiempo
0	2.6996	2.3727	0.1711	0.2959	1:19
1	2.2830	2.0467	0.3143	0.3843	0:40
2	2.0295	1.9416	0.3876	0.4160	0:40
3	1.8689	1.7726	0.4390	0.4675	0:40
4	1.7320	1.6667	0.4717	0.4919	0:39
5	1.6068	1.6045	0.5131	0.5081	0:39
6	1.5308	1.5408	0.5378	0.5207	0:39
7	1.4249	1.5459	0.5699	0.5162	0:39
8	1.3346	1.4105	0.5857	0.5645	0:39
9	1.2441	1.4153	0.6170	0.5641	0:39

Tabla 6.13: Tabla que muestra la pérdida y acierto de entrenamiento y validación, y el tiempo que necesita el modelo CNN creado desde cero por *epoch* en la predicción de 20 artistas.

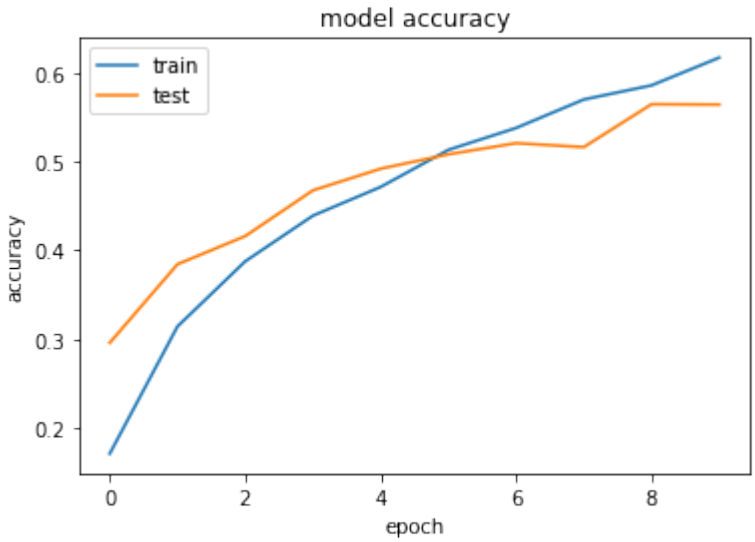

Figura 6.7: Gráfico que muestra la evolución de los *accuracy* de entrenamiento y validación por *epoch* para la predicción de artistas, utilizando la CNN creada desde cero.

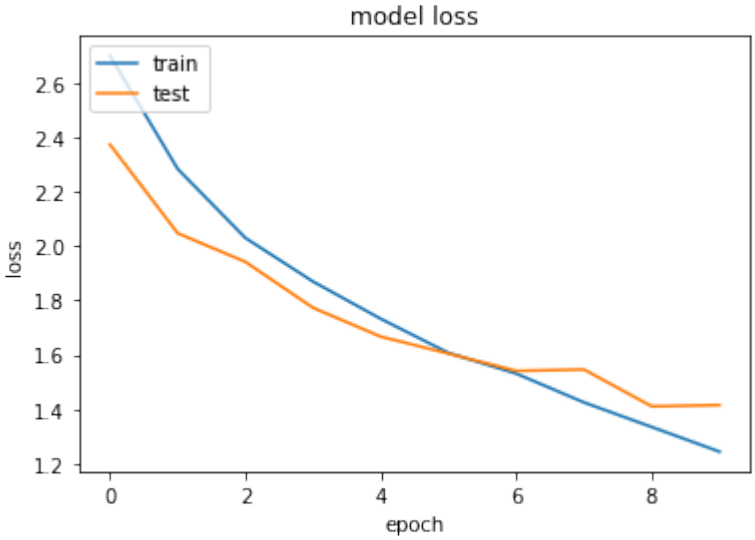

Figura 6.8: Gráfico que muestra la evolución de los *loss* (o pérdida) de entrenamiento y validación por *epoch* para la predicción de artistas, utilizando la CNN creada desde cero.

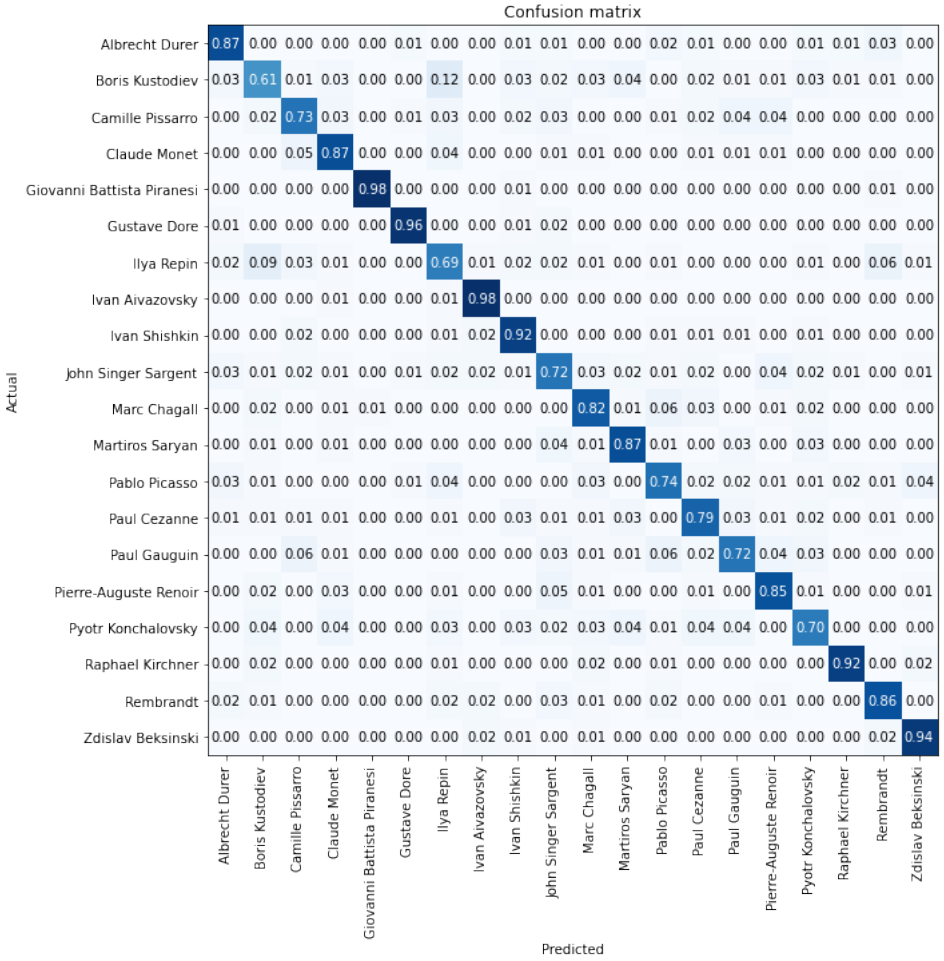

Figura 6.9: Matriz de confusión para la predicción de artista con 20 casos con la red *VGG-16*

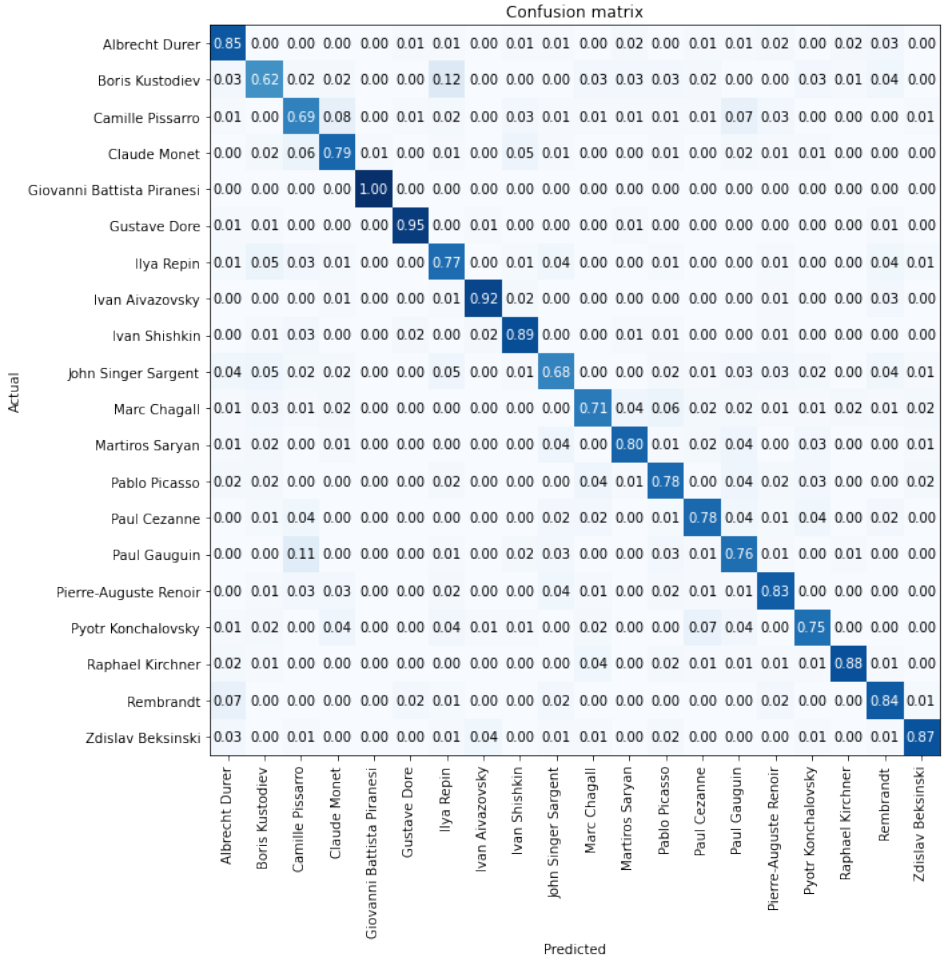

Figura 6.10: Matriz de confusión para la predicción de artista con 20 casos con la red *ResNet-34*

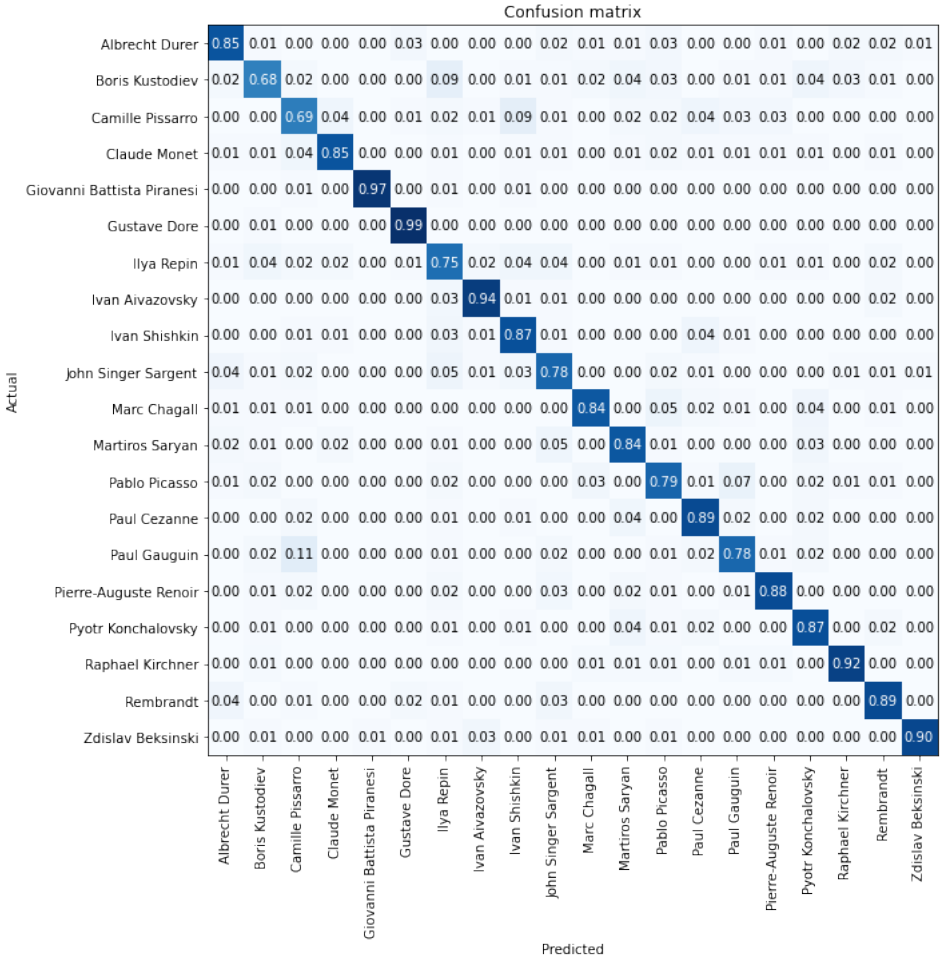

Figura 6.11: Matriz de confusión para la predicción de artista con 20 casos con la red *ResNet-50*

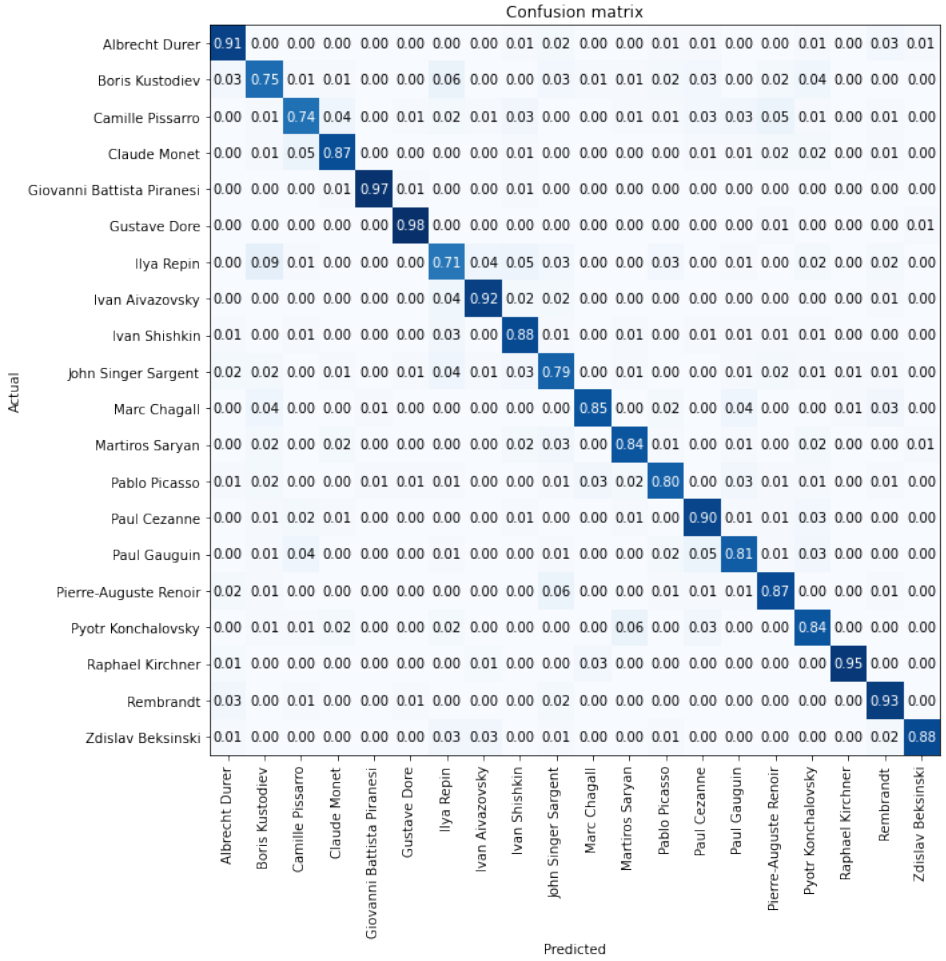

Figura 6.12: Matriz de confusión para la predicción de artista con 20 casos con la red *DenseNet-121*

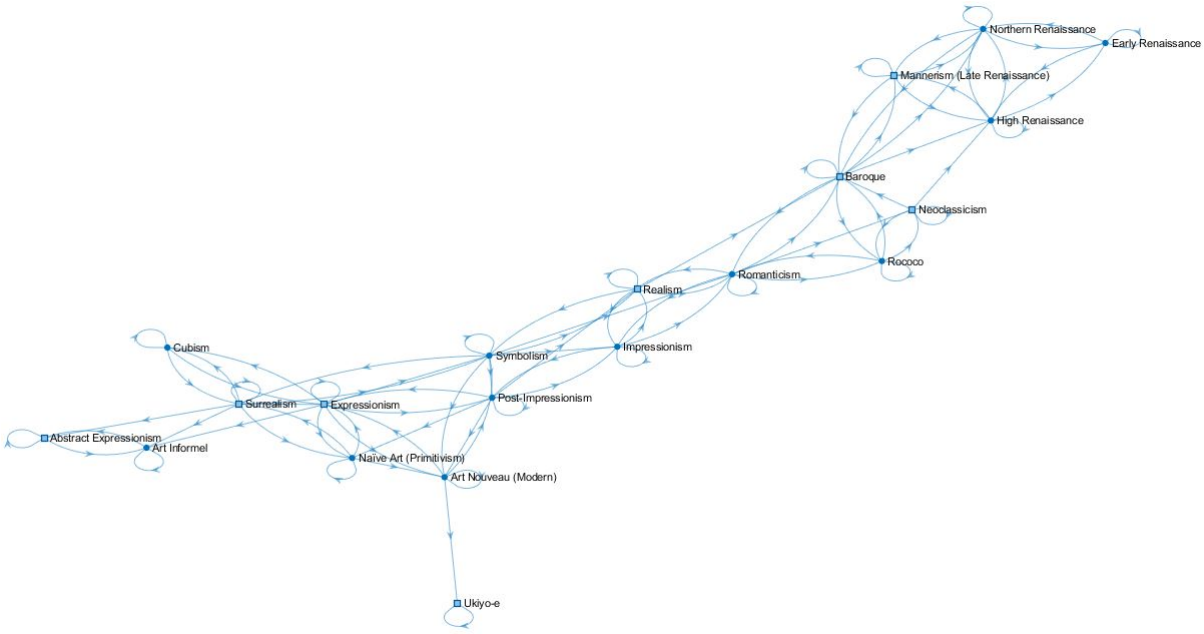

Figura 6.13: Grafo que muestra las relaciones entre los estilos artísticos

6.1.3. Estilos artísticos fusionados

Como se ha mencionado, hay muchos estilos artísticos que están históricamente relacionados (como se ve en la figura 6.13), ya que a veces un estilo deriva de otro, o hay estilos con diferentes variantes. En este caso, se muestra un experimento en el que se reduce el número de estilos, ya que algunos de los estilos relacionados se fusionan en uno, reduciendo el conjunto de 20 a 13 estilos.

Como se puede ver en el grafo 6.13, los estilos (representados en vértices) que están relacionados se encuentran cerca, y a simple vista se pueden apreciar ciertos grupos. Por ejemplo, en la parte superior derecha, se concentran las cuatro variantes del renacimiento, y también podemos ver en la parte inferior izquierda que el *Abstract Expressionism* y el *Art Informel* están estrechamente unidos y un poco separados del resto, tal y como se ha visto reflejado en las matrices de confusión para la predicción de estilo, ya que ambos eran muy confundidos. Por otro lado, también tenemos estilos bien clasificados, como es el caso del *Ukiyo-e*. Este estilo es de origen Japonés, y a simple vista un cuadro de este estilo es muy fácil de distinguir para cualquiera. Es por esto que en el grafo aparece muy apartado, demostrando que hay poca relación entre el *Ukiyo-e* y los demás estilos. Estas son las diferentes agrupaciones que se han hecho para disminuir la confusión entre los estilos y aumentar así el *accuracy*:

1. ***Renaissance (o Renacimiento)***: Este grupo está compuesto por las cuatro variantes del *Renaissance* que forman parte de los 20 estilos de arte en nuestro conjunto de datos: *Early Renaissance* o *Renacimiento Temprano* , *High Renaissance* o *Alto Renacimiento*, *Mannerism (Late Renaissance)* o *Manierismo (Renacimiento Tardío)* y *Northern Renaissance* o *Renacimiento del Norte*. El Renacimiento se originó en Italia, en el movimiento cultural del siglo XV conocido como *Quattrocento*, donde nació el Renacimiento temprano. Variaciones como el *High Renaissance* se desarrollaron en el comienzo del siglo XVI (*Cinquecento*), donde se consigue el equilibrio y la perfección. Años más tarde, el *Mannerism (Late Renaissance)* trae el inicio de la decadencia y ruptura de la forma equilibrada y perfecta del clasicismo. Finalmente, tenemos el *Northern Renaissance*, que fue la variación del *Renaissance* en el norte de Europa. Como se ha visto, estos cuatro estilos están bastante relacionados, tal y como lo muestra el grafo de las relaciones entre estilos, por lo que unirlos en un solo estilo es una buena idea.

2. ***Abstract-Expressionism-Informel***: En este caso, fusionamos los estilos *Abstract Expressionism* o *Expresionismo abstracto*, el *Expressionism* o *Expresionismo* y el *Art Informel* o *Informalismo*. Por un lado tenemos el *Expressionism*, que se originó a principios del siglo XX en Alemania. Mientras que por el otro lado, tenemos tanto el *Abstract Expressionism* y *Art Informel*, que se crearon después de la Segunda Guerra Mundial, pero el primero se originó en los EE.UU. mientras que el segundo fue en Francia, Italia y España. La razón de fusionar el expresionismo con los otros dos es que el *Abstract Expressionism* está relacionado con el *Expressionism*, y se desarrolló a la vez que el *Art Informel*. De esta forma, también hay que tener en cuenta que el *Expressionism* influyó en muchos estilos diferentes como el *cubismo* o el *surrealismo*.

3. ***Impressionism* (o *Impresionismo***): Aquí se han fusionado tanto el *Impressionism* como el *Post-Impressionism*. El primero fue originado por un grupo de artistas parisinos a finales del siglo XIX, mientras que el otro se desarrolló entre el final del sigo XIX y principios del XX. El *Impressionism*estaba más centrado en detalles como la luz y los colores, y el *Post-Impressionism* en cambio trató de reflejar fielmente la naturaleza con una visión muy subjetiva, pero al ser uno una nueva versión del otro, juntarlos en uno solo puede hacer aumentar el acierto de los modelos.

4. ***Baroque-Rococo***: Por último, tenemos tanto el *Baroque* o *Barroco* como el *Rococó*. El barroco se desarrolló entre el siglo XVII y principios del XVIII, principalmente

en Italia. Básicamente, la característica principal de este estilo de arte es el realismo con el uso de fuertes contrastes. El *Rococo* se originó en Francia en el siglo XVIII, y también fue conocido como *Barroco Tardío*, más influido por la naturaleza y la mitología. La fusión de estos estilos se debe a que el *Rococo* surge del *Baroque*, por lo que cuenta con muchos parecidos.

Al hacer estas fusiones entre los estilos, el número de imágenes por estilo sigue siendo el mismo (1200 por estilo), por lo que también se reduce el número de imágenes total, al igual que el número de estilos. De esta forma, pasamos de tener 20 estilos con 1200 imágenes por cada uno de ellos, teniendo 24.000 imágenes, a 13 estilos con 1200 imágenes, por lo que tenemos 15.600 imágenes. De esta forma, la red tardará menos tiempo en ser entrenada, y aumentará su acierto, ya que se han juntado estilos que en casos anteriores eran muy confundidos.

A continuación se explican los resultados obtenidos para cada modelo en la predicción de estilo con estilos fusionados:

1. ***VGG-16***: Tal y como se puede observar en la tabla 6.14, el *accuracy* obtenido al fusionar los estilos es bastante mayor que en el caso sin fusionarlos. En este caso el mejor acierto es de 63.01 %, mientras que en el caso anterior era de 50.12 %. Estamos ante una mejora muy significativa, aunque hay que tener en cuenta que se ha reducido bastante el número de estilos.

   En la imagen 6.16 encontramos la matriz de confusión que se ha obtenido al realizar el entrenamiento con este modelo. Estilos como el *Ukiyo-e* siguen con muy buena diferenciación del resto (con el mejor *accuracy* de todos), y todos los estilos que han sido fusionados en uno nuevo han obtenido un acierto mayor o igual al 65 %, siendo el *Renaissance* el mejor clasificado de estos, con un *accuracy* del 90 %.

2. ***ResNet-34***: En el caso de *ResNet-34*, el *accuracy* también ha aumentado, consiguiendo un 63.02 % respecto al 50.18 % obtenido en la predicción con 20 estilos.

   La matriz de confusión se puede ver en la imagen 6.17. En ella se ve que son tanto el *Renaissance* como el *Ukiyo-e* los estilos mejor clasificados, ambos con un acierto del 88 %. En cambio, dos de los peor clasificados o más confundidos, son el *Romanticism* y el *Realism*, el primero con un acierto del 26 % y el segundo 32 %.

3. ***ResNet-50***: Esta versión de *ResNet* formada por 50 capas, consigue un acierto mayor que su alternativa de 34 capas como era de esperar. Consiguiendo un *accuracy*

del 64.91 %, supera también a la precisión obtenida en la predicción de 20 estilos, la cual era del 54.72 %.

En cuanto a la clasificación de los estilos, podemos decir que vuelve a ser bastante buena, tal y como se puede ver en la imagen 6.18. Los mejor clasificados vuelven a ser *Ukiyo-e* y *Renaissance* con 86 % y 89 % de acierto respectivamente, mientras que los que mayor error han sufrido al ser clasificados son el *Symbolism* y el *Romanticism*.

4. ***DenseNet-121***: Esta red vuelve a ser la que mejor clasifica los estilos, consiguiendo un *accuracy* máximo del 66.68 %, el cual supera con creces el 56.25 % obtenido en la predicción de 20 estilos. Al igual que ha sucedido en los demás casos con *DenseNet-121*, el *accuracy* del modelo aumenta a medida que se avanza con las *epochs*, por lo que parece que aumentando el número de *epochs* también se aumentará su acierto, siempre hasta llegar a un límite.

   Por otro lado, nos encontramos con la matriz de confusión de *DenseNet-121* en la imagen 6.19. Tal y como muestra la matriz, los dos estilos mejor clasificados vuelven a ser tanto el*Renaissance* como el *Ukiyo-e*, con un acierto del 90 % y 91 % respectivamente. Los estilos que han sido fusionados también tienen un *accuracy* medianamente alto, todos ellos por encima del 68 %.

5. ***CNN simple***: Por último tenemos a la CNN simple creada desde cero. Al igual que en los demás casos, esta red es la que peor desempeño consigue en términos de acierto a la hora de clasificar. Aun y todo, se puede observar claramente que el *accuracy* es mejor que en el caso de la predicción de 20 estilos, el cual contaba con unos aciertos de 41.18 % y 30.10 % para entrenamiento y validación, mientras que ahora son del 38.25 % y 38.18 % (ver tabla 6.18). Aunque el *accuracy* de entrenamiento sea más bajo, en el caso de los 20 estilos se estaba dando *overfitting*, por lo que por mucho que se aumentara el número de *epochs* para el entrenamiento, el acierto de validación se estancaría o incluso bajaría. Mientras, en este caso, ambos valores de acierto son muy similares, por lo que aumentando el total de *epochs* quizá también mejoren los resultados. Esta idea se puede corroborar también viendo las figuras 6.14 y 6.15, donde vemos que no se ha llegado al límite, ni de *accuracy* ni de pérdida, por lo que es posible que se pueda seguir mejorando los *accuracy*, siempre hasta llegar a un límite, ya que a partir de cierto punto, el *accuracy* del modelo se estanca y no aumenta a pesar de aumentar los *epochs* de entrenamiento.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	2.1616	1.6284	0.5014	12:15
1	1.6579	1.3875	0.5497	11:42
2	1.5504	1.3584	0.5547	11:42
3	1.5091	1.2679	0.5797	11:39
4	1.3781	1.2257	0.5968	11:39
5	1.3031	1.1860	0.6033	11:39
6	1.2465	1.1513	0.6197	11:39
7	1.2935	1.1344	0.6264	11:39
8	1.1587	1.1289	0.6301	11:39
9	1.1669	1.1247	0.6279	11:39

Tabla 6.14: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *VGG-16* por *epoch* en la predicción de 13 estilos.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	2.2989	1.6802	0.4837	5:31
1	1.7419	1.4404	0.5291	5:17
2	1.6437	1.3713	0.5439	5:14
3	1.5096	1.2988	0.5760	5:17
4	1.4702	1.2421	0.5875	5:17
5	1.3457	1.1944	0.6083	5:07
6	1.3299	1.1657	0.6302	5:07
7	1.2360	1.1507	0.6214	5:07
8	1.2371	1.1622	0.6300	5:07
9	1.2126	1.1422	0.6268	5:07

Tabla 6.15: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *ResNet-34* por *epoch* en la predicción de 13 estilos.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	2.0634	1.5493	0.5208	7:54
1	1.6686	1.3583	0.5554	7:38
2	1.4625	1.3053	0.5647	7:35
3	1.3976	1.2134	0.5972	7:32
4	1.3605	1.1571	0.6179	7:40
5	1.2349	1.1092	0.6277	7:55
6	1.1302	1.0808	0.6402	7:41
7	1.0873	1.0575	0.6460	7:38
8	1.0184	1.0618	0.6491	7:35
9	1.0014	1.0447	0.6491	7:39

Tabla 6.16: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *ResNet-50* por *epoch* en la predicción de 13 estilos.

Epoch	Train_loss	Valid_loss	Accuracy	Tiempo
0	1.9845	1.4838	0.5395	7:59
1	1.7043	1.3366	0.5618	7:29
2	1.4396	1.2379	0.5825	7:36
3	1.4044	1.1741	0.6091	7:38
4	1.2089	1.1362	0.6237	7:39
5	1.1663	1.0853	0.6437	7:38
6	1.0722	1.0644	0.6485	7:38
7	0.9889	1.0220	0.6629	7:39
8	0.9760	1.0127	0.6658	7:39
9	0.9328	1.0095	0.6668	7:43

Tabla 6.17: Tabla que muestra la pérdida de entrenamiento y validación, el acierto y el tiempo que necesita la red *DenseNet-121* por *epoch* en la predicción de 13 estilos.

Epoch	Train_loss	Val_loss	Train_acc	Val_acc	Tiempo
0	2.3470	2.3113	0.2246	0.2399	4:31
1	2.2166	2.1559	0.2756	0.2954	3:54
2	2.1410	2.1097	0.3043	0.3098	3:49
3	2.0879	2.0666	0.3186	0.3198	3:50
4	2.0311	2.0534	0.3395	0.3424	3:57
5	2.0014	1.9980	0.3500	0.3514	3:53
6	1.9631	1.9584	0.3591	0.3715	3:52
7	1.9408	1.9258	0.3681	0.3736	3:51
8	1.9061	1.9322	0.3777	0.3818	3:56
9	1.8895	1.9007	0.3825	0.3799	3:57

Tabla 6.18: Tabla que muestra la pérdida y acierto de entrenamiento y validación, y el tiempo que necesita el modelo CNN creado desde cero por *epoch* en la predicción de 13 estilos.

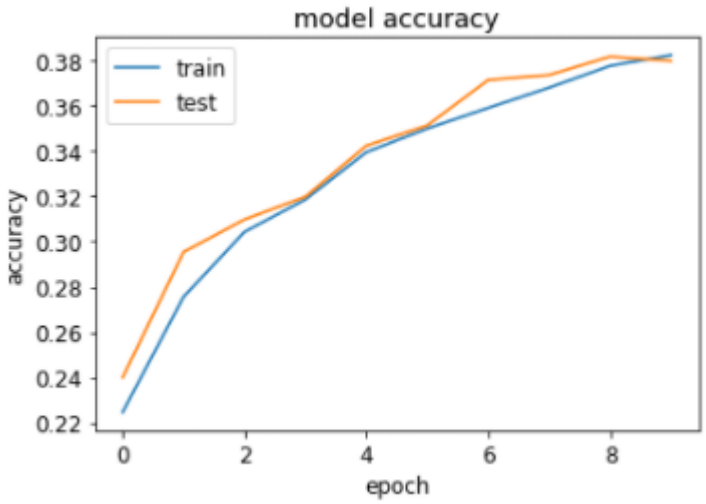

Figura 6.14: Gráfico que muestra la evolución del *accuracy* del entrenamiento y validación por *epoch* para la predicción de 13 estilos, utilizando la CNN creada desde cero.

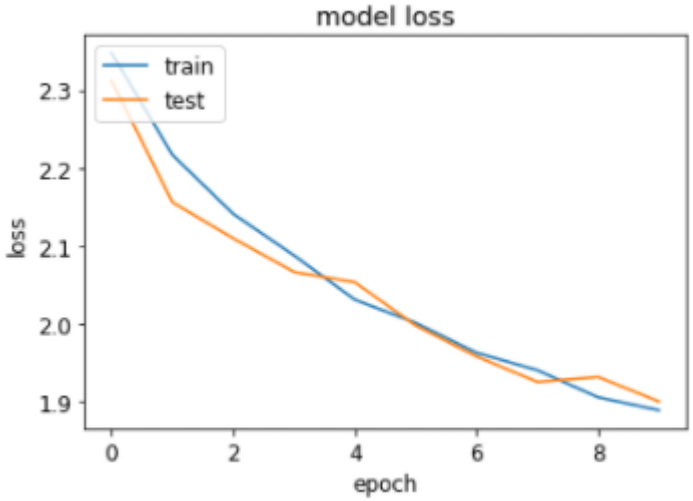

Figura 6.15: Gráfico que muestra la evolución del *loss* del entrenamiento y validación por *epoch* para la predicción de 13 estilos, utilizando la CNN creada desde cero.

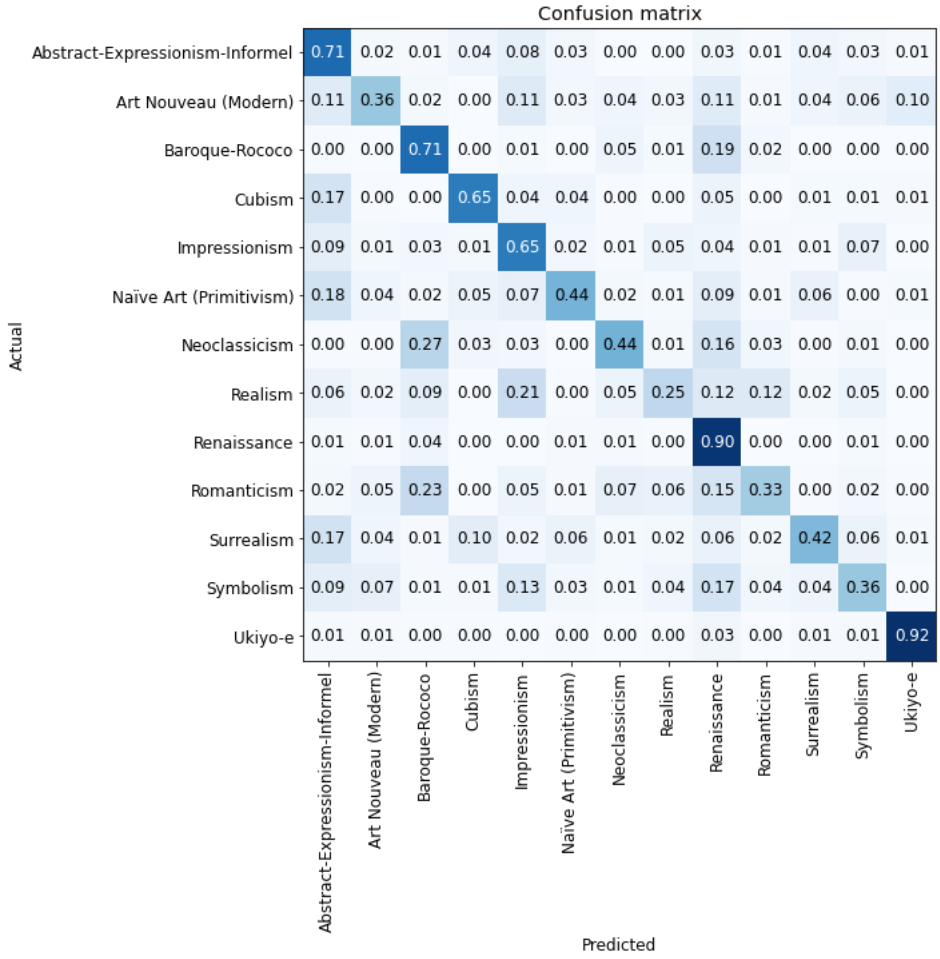

Figura 6.16: Matriz de confusión para la red *VGG-16* en el caso de la predicción de 13 estilos (con estilos fusionados).

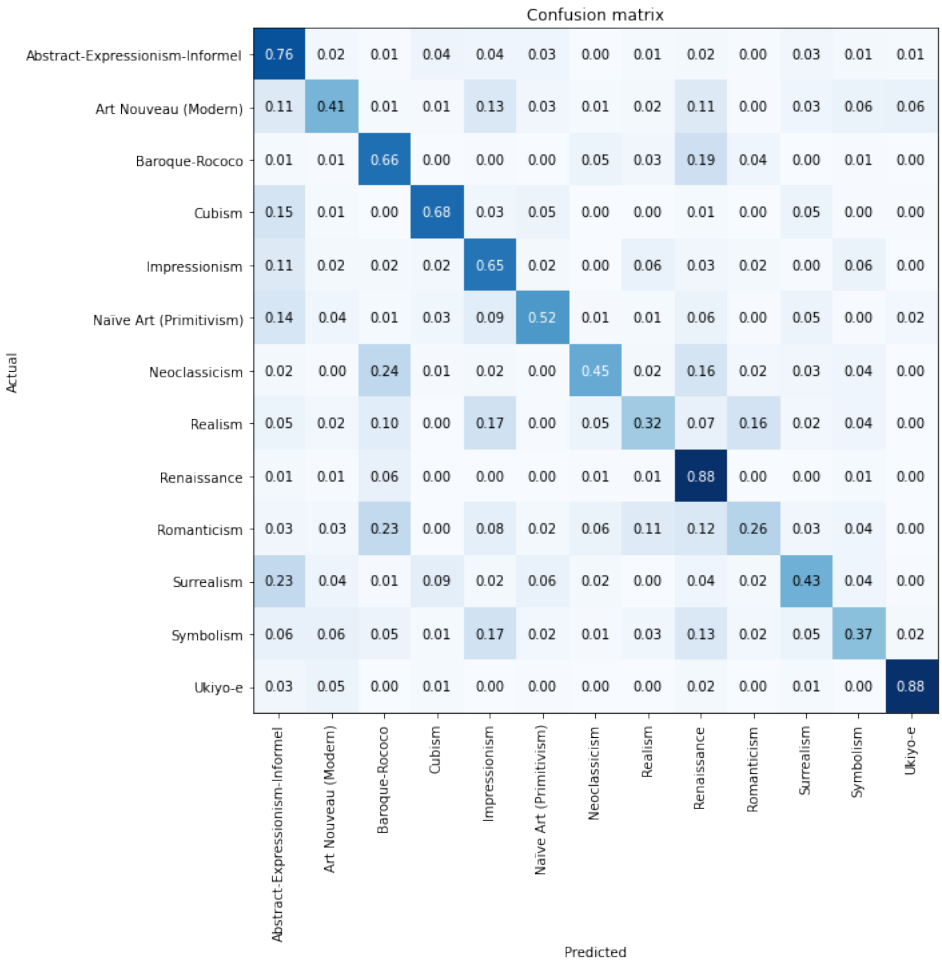

Figura 6.17: Matriz de confusión para la red *ResNet-34* en el caso de la predicción de 13 estilos (con estilos fusionados).

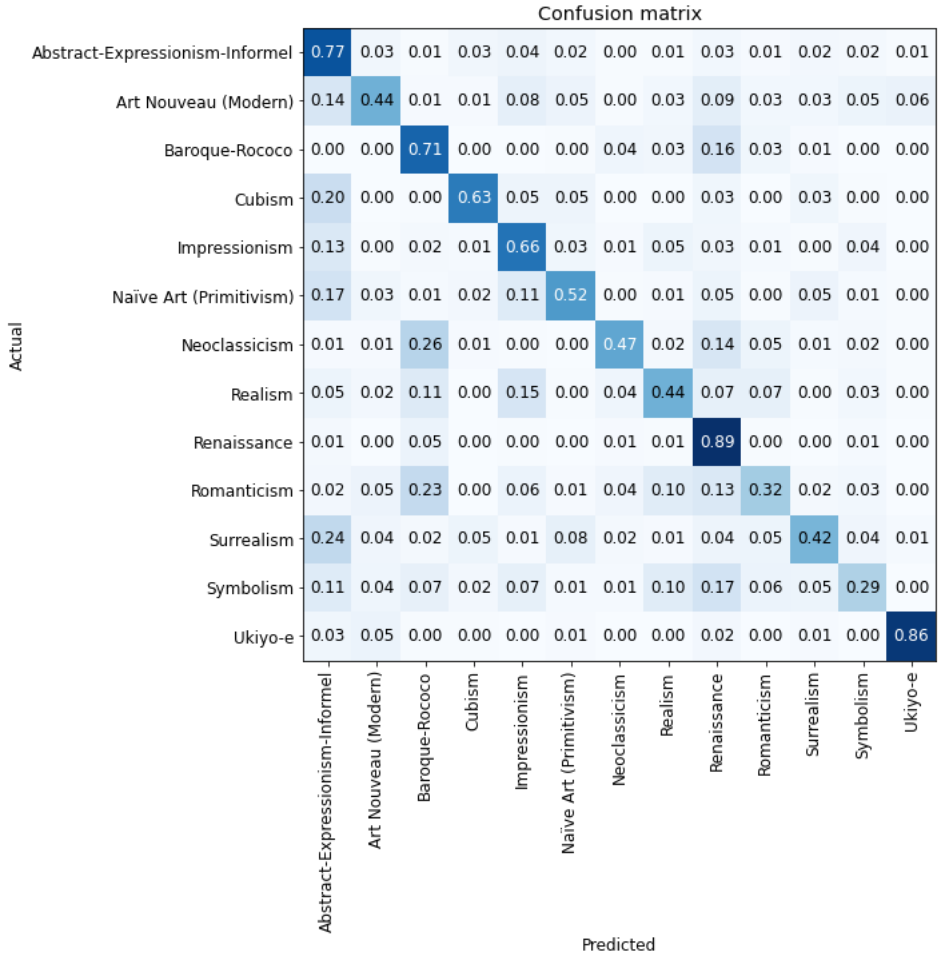

Figura 6.18: Matriz de confusión para la red *ResNet-50* en el caso de la predicción de 13 estilos (con estilos fusionados).

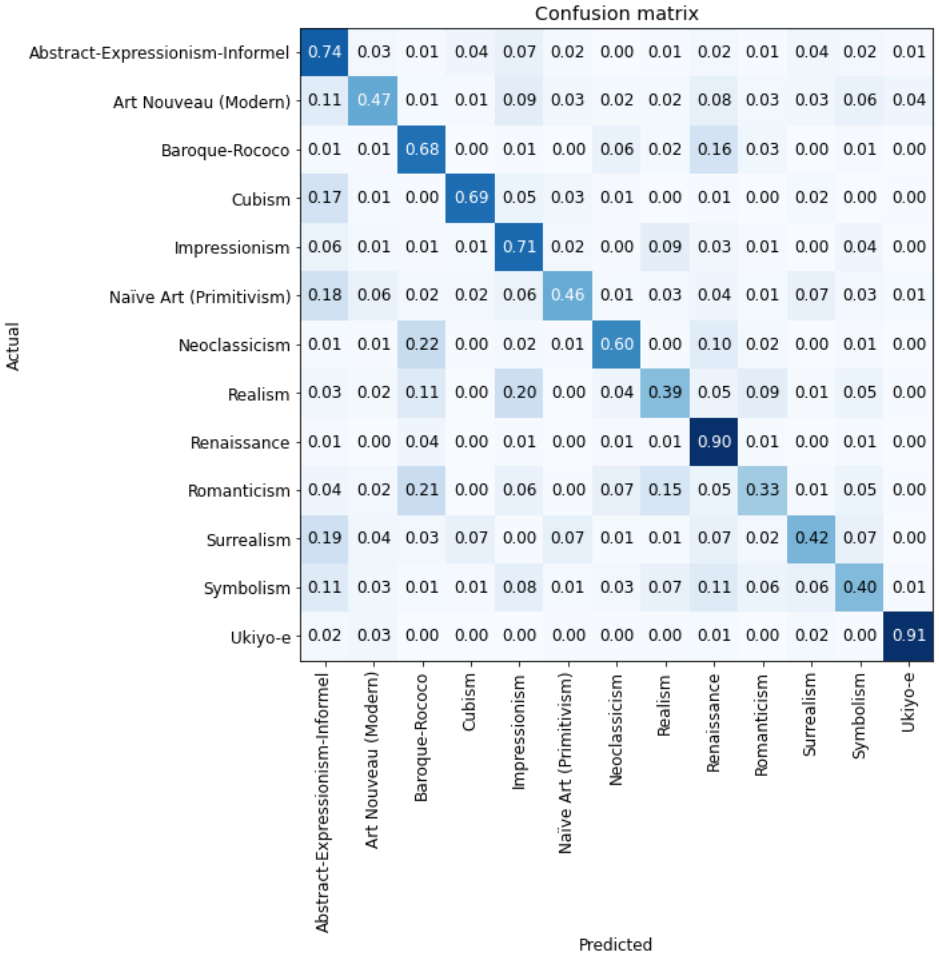

Figura 6.19: Matriz de confusión para la red *DenseNet-121* en el caso de la predicción de 13 estilos (con estilos fusionados).

Ya hemos visto que fusionando los estilos los resultados mejoran bastante, pasando en algunos casos como *VGG-16* y *ResNet-34* de al rededor de 50 % de *accuracy* a 63 %. Pero esta mejora se debe mayormente a que se ha reducido el número de clases, eliminando así ciertas relaciones entre clases, las cuales producían grandes confusiones y errores. En este otro caso, lo interesante es intentar mejorar esos resultados sin afectar ni modificar el número de clases e imágenes, es decir, intentar mejorar los resultados de los modelos para los casos de 20 estilos y 20 artistas.

Para mejorar los *accuracy* obtenidos, se ha decantado por una de las técnicas que se ha visto en la sección 3, donde en [Lecoutre et al., 2017] se vuelve a entrenar la red para aumentar su acierto. En este caso se hace algo parecido, ya que se descongelan los pesos de las capas de la red y se calcula un *learning rate* adecuado para el modelo a través de la función *lr_find*, para después volver a entrenarlo y mejorar así su rendimiento para la predicción. De esta forma, se actualizan los pesos de la red para que el entrenamiento de la misma sea más eficiente, ya que los nuevos pesos están dirigidos para la predicción de estilo o artista, y no para las clases predeterminadas de *ImageNet*.

A continuación se muestran las mejoras obtenidas para cada predicción:

En primer lugar tenemos la tabla 6.19, la cual muestra los *accuracy* que se han obtenido al acabar el entrenamiento, y los obtenidos reentrenando el modelo tras calcular el *learning rate* adecuado. Los resultados son para la predicción de 20 artistas y utilizando las redes *VGG-16*, *ResNet-34*, *ResNet-50* y *DenseNet-121*. Como se puede observar en la tabla, todos los resultados se han mejorado bastante, consiguiendo un máximo de 87.27 % de acierto, mejorando todos los *accuracy* entre un 1.5 % y 5 %.

En cuanto a la predicción de estilo, las mejoras conseguidas se pueden observar en la tabla 6.20. Al igual que en el caso de los artistas, las mejoras son bastante notables, ya que la mejora más pequeña es del 3 %, mientras que la más alta supera el 6 %. El acierto más alto pasa de 56.25 % a 59.41 %, por lo que teniendo en cuenta los nuevos resultados, se puede deducir que reentrenar los modelos con un *learning rate* adecuado es una buena opción para mejorar los resultados de los modelos.

Modelo	Accuracy (único entrenamiento)	Accuracy (con lr óptimo y reentrenando)
VGG-16	0.8213	0.8722
ResNet-34	0.8092	0.8576
ResNet-50	0.8382	0.8727
DenseNet-121	0.8577	0.8722

Tabla 6.19: Tabla que muestra la comparación entre los *accuracy* obtenidos para la predicción de artista, con un solo entrenamiento y tras volver a entrenar con un *learning rate* óptimo, para las redes *VGG-16*, *ResNet-34*, *ResNet-50* y *DenseNet-121*.

Modelo	Accuracy (único entrenamiento)	Accuracy (con lr óptimo y reentrenando)
VGG-16	0.5012	0.5652
ResNet-34	0.5072	0.5533
ResNet-50	0.5472	0.5791
DenseNet-121	0.5625	0.5941

Tabla 6.20: Tabla que muestra los *accuracy* obtenidos para la predicción de estilo, tanto con un solo entrenamiento, como volviendo a entrenar con un *learning rate* óptimo, para las redes *VGG-16*, *ResNet-34*, *ResNet-50* y *DenseNet-121*.

7. CAPÍTULO

Conclusiones

Como se ha visto en el apartado de *Pruebas realizadas y resultados obtenidos*, los resultados de la predicción de artista son bastante mejores que los resultados obtenidos para la predicción de estilos (en el caso de 20 estilos y artistas). A primera vista, parece que tiene poco sentido, ya que en el caso de los estilos hay 24.000 imágenes, y en el de los artistas casi 10.000, por lo que en el caso de los estilos al haber más información, debería haber mayor acierto. Pero hay que tener en cuenta, que hay obras que no están del todo bien definidas en un único estilo, es decir, hay algunas obras que pueden estar influenciadas por más de un estilo, ya que muchos estilos han sido definidos después de que las obras fuesen creadas. También sucede el caso donde hay variaciones de un mismo estilo, como pasa con el Renacimiento o *Renaissance*, donde los estilos *Early Renaissance*, *Mannerism (Late Renaissance)*, *High Renaissance* y *Northern Renaissance* son distintas variantes del mismo, por lo que confundir estos estilos no resulta muy extraño.

Otro de los puntos a tener en cuenta es la dificultad que se esconde a la hora de determinar a qué estilo pertenece una obra. Ya que esto dependerá de muchos factores, como la época, los colores, las formas o los objetos e ideas que se representan y que los expertos tienen que interpretar y catalogar. En cambio, catalogar una obra por artista es algo menos subjetivo, ya que normalmente el propio autor suele firmar la obra, por lo que catalogar una obra teniendo dicha información no tiene dificultad (en el caso de no saber quién es el autor sí que se dificulta su clasificación, lo cual la asemeja al caso de la del estilo). Por otro lado, si las obras de distintos artistas son confundidas, seguramente se debe a que uno de los artistas fue alumno del otro. De esta forma se puede comprender que la predicción de artista consiga mejores resultados que la predicción de estilo, ya que en esta última, la

facilidad para confundirse es mucho mayor.

En cuanto a los modelos, cabe destacar que el modelo que mejor desempeño ha obtenido en prácticamente todos los experimentos es *DenseNet-121*, que como se ha explicado en el apartado 4.4.1, es la red más profunda que se ha utilizado en este trabajo, por lo que tiene sentido que obtenga los mejores resultados en cuanto a *accuracy*. Además, otra de las posibles razones por la que *DenseNet-121* destaca tanto, es porque es la más reciente de las cuatro redes que se han probado (fue creada en 2017), por lo que está mejor optimizada, ya que *VGG-16* por ejemplo, la cual fue creada en 2014, cuenta con bastantes menos capas y en la mayoría de los entrenamientos ha necesitado más tiempo, consiguiendo menos acierto. Por otro lado tenemos a los modelos *ResNet-34* y *ResNet-50*, los cuales han conseguido muy buenos resultados, demostrando que son dos de los modelos que pertenecen a una de las mejores redes neuronales convolucionales para la clasificación de imágenes.

También está el modelo CNN creado desde cero, el cual ha conseguido los peores resultados de todos los modelos (tal y como era de esperar), pero dada la complejidad de la red, estos son bastante aceptables. Hay que tener en cuenta que se trata del modelo más simple que se ha utilizado en este TFG, ya que cuenta con menos de 5 millones de parámetros y 19 capas (respecto a los 138 millones de VGG-16, 21.8 y 23 millones de *ResNet-34* y *ResNet-50*, o los casi 10 millones de *DenseNet-121*), por lo que una de sus ventajas es el poco tiempo que se necesita para ser entrenado.

Es por esto que se dan por cumplidos los objetivos de este TFG, ya que se han analizado y utilizado distintas redes que conforman el estado del arte para la clasificación de imágenes, además de comparar los resultados obtenidos por las mismas entre ellas y un modelo CNN creado desde cero. También se han razonado los resultados de las predicciones, y se han encontrado las dependencias históricas que han causado las confusiones más importantes para los modelos.

8. CAPÍTULO

Trabajo futuro y posibles mejoras

Como se ha visto, los resultados que se obtienen son bastante aceptables en ambas predicciones, pero el conjunto de clases a predecir es bastante limitado. Hay que tener en cuenta que hay muchísimos más estilos y artistas que no se tienen en cuenta en el uso de estos modelos, por lo que si la imagen a predecir fuese de un estilo o artista que no aparece en el dataset, sería clasificado de forma incorrecta. Para solucionar este problema, es necesario aumentar el número de datos e imágenes, pero como ya hemos visto, uno de los problemas más relevantes es la falta de recursos y el excesivo tiempo que se necesita para procesar muchos datos. Es por eso que se necesitaría también un mejor equipo de trabajo, ya que plataformas como *Google Colab* tienen restricciones de tiempo de uso y recursos, a pesar de ser muy eficientes.

Por otro lado, las imágenes que se han utilizado para entrenar a los modelos son bastante más reducidas que sus versiones originales, ya que en su tamaño original eran muy pesadas de procesar y los tiempos aumentaban significativamente. Por eso, si se contara con un ordenador con mejores prestaciones que el utilizado para este trabajo, es muy posible que se consiguieran mejores resultados en las predicciones, ya que al utilizar imágenes más grandes se trabaja con mayor cantidad de información.

Una posible mejora o nueva función es crear una aplicación para dispositivos móviles, la cual mediante la cámara realice una fotografía y el modelo implementado en ella sea capaz de hacer una predicción (de estilo o artista, según quiera el usuario). De esta forma, cualquier persona podría averiguar a qué estilo o artista pertenece una obra, tan solo haciendo una fotografía. Para ello, hay distintas aplicaciones, como *TensorFlow Lite*. Con

esta herramienta, se puede entrenar un modelo con tan solo un conjunto de datos, formado por un grupo de carpetas, cada una de ellas con las imágenes y con el nombre de la clase a la que pertenecen las mismas. Dentro de esta herramienta, se puede trabajar con tres modelos ya creados: *EfficientNet-Lite*, *MobileNetV2* y *ResNet50*. Los pasos a realizar para entrenar el modelo son muy parecidos a los utilizados previamente, por lo que una vez entrenado el modelo, se descarga y se integra en el programa *Android Studio*, donde se desarrolla la aplicación. La interfaz de la aplicación sería bastante simple, tan solo un botón para hacer la foto, y tras hacerla, aparece una lista con las probabilidades de que la imagen pertenezca a los estilos según el modelo.

En la imagen 8.1, podemos encontrar la interfaz de la aplicación (diseñada solamente para la predicción de estilo), la cual muestra un menú muy simple con un botón para sacar las fotos, el cual abre la aplicación de la cámara al ser pulsado. Una vez se haya realizado la fotografía, la red calculará los porcentajes que representan la probabilidad de que la imagen pertenezca a un estilo artístico. En este caso, en la figura 8.2, podemos observar la foto hecha en la parte superior, la cual pertenece a la obra 'La gran ola de Kanagawa ' de *Katsushika Hokusai*, una de las obras más relevantes del estilo *Ukiyo-e*. Se puede ver que se decanta con una probabilidad mayor al 73 %, a que la imagen pertenece al estilo *Ukiyo-e*, el cual es bastante preciso. Hay que tener en cuenta que factores como la calidad de la fotografía afectan bastante a la precisión de la red, además de que los modelos utilizados son bastante más simples que los que se han utilizado anteriormente, por lo que los resultados no son tan buenos.

A pesar de las limitaciones que se dan en el caso de la aplicación, es una buena forma de implementar los modelos para el reconocimiento, ya que cualquier persona podría utilizarla sin tener muchos conocimientos. Es por esto, que la aplicación móvil parece una buena idea de trabajo futuro, aunque la misma aplicación necesitaría mejoras y más recursos.

Figura 8.1: Captura de pantalla que muestra el menú inicial de la aplicación para móvil de predicción de estilo.

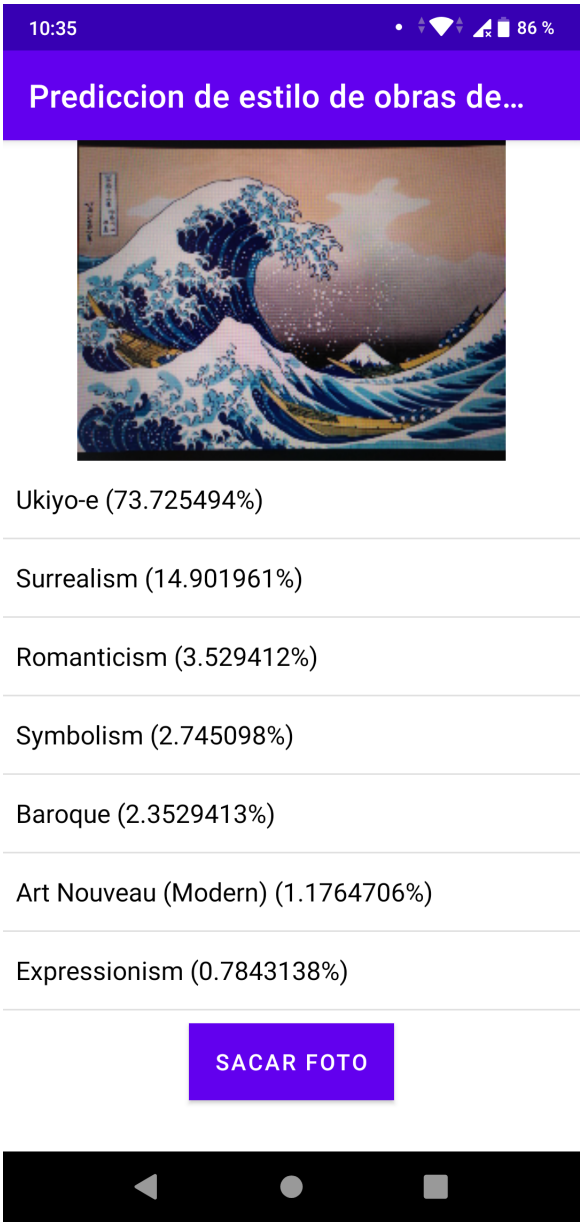

Figura 8.2: Captura de pantalla que muestra los resultados de la predicción de estilo para una fotografía del cuadro 'La gran ola de Kanagawa' perteneciente al estilo *Ukiyo-e*.

Bibliografía

[Bar et al., 2015] Bar, Y., Levy, N., and Wolf, L. (2015). Classification of artistic styles using binarized features derived from a deep neural network. *LNCS*, 8925:71–84.

[Bergamo et al., 2011] Bergamo, A., Torresani, L., and Fitzgibbon, A. (2011). Picodes: Learning a compact code for novel-category recognition. pages 2088–2096.

[Ciresan et al., 2011] Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. pages 1237–1242.

[Eva Cetinic, 2018] Eva Cetinic, Tomislav Lipic, S. G. (2018). Fine-tuning convolutional neural networks for fine art classification. pages 107–118.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition.

[Hershey et al., 2017] Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., Slaney, M., Weiss, R. J., and Wilson, K. (2017). Cnn architectures for large-scale audio classification. pages 131–135.

[Huang et al., 2017] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks.

[Jangtjik et al., 2016] Jangtjik, K. A., Yeh, M.-C., and Hua, K.-L. (2016). Artist-based classification via deep learning with multi-scale weighted pooling. page 635–639.

[Karayev et al., 2013] Karayev, S., Hertzmann, A., Winnemoeller, H., Agarwala, A., and Darrell, T. (2013). Recognizing image style. *CoRR*, abs/1311.3715.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. 25.

[Lecoutre et al., 2017] Lecoutre, A., Negrevergne, B., and Yger, F. (2017). Recognizing art style automatically in painting with deep learning. 77:327–342.

[Redolfi, 2018] Redolfi, J. (2018). Aplicación en agricultura de precisión de esquemas actuales de reconocimiento visual.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[Viswanathan and Stanford, 2017] Viswanathan, N. and Stanford (2017). Artist identification with convolutional neural networks.

[Wang et al., 2019] Wang, C., Cheng, M., Sohel, F., Bennamoun, M., and Li, J. (2019). Normalnet: A voxel-based cnn for 3d object classification and retrieval. *Neurocomputing*, 323:139–147.

[Yang and Min, 2019] Yang and Min (2019). A multi-column deep framework for recognizing artistic media. *Electronics*, 8:1277.