Universidad Zaragoza
1542

Trabajo de Fin de Grado

Reconocimiento de vocalistas mediante redes neuronales profundas

---

Autor: Andrés Fandos Villanueva

Director: Javier Civera Sancho

Grado en Ingeniería Electrónica y Automática
Curso 2019/2020

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe entregarse en la Secretaría de la EINA, dentro del plazo de depósito del TFG/TFM para su evaluación).

D./Dª. Andrés Fandos Villanueva ,en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza, Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado (Título del Trabajo)

Reconocimiento de vocalistas mediante redes neuronales profundas

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 26 de Junio de 2020

Fdo: Andrés Fandos Villanueva

RESUMEN

Los sistemas de reconocimiento de voz automático son una disciplina de la inteligencia artificial cuyo aprendizaje se puede realizar mediante técnicas de aprendizaje deductivo e inductivo. Ambas técnicas tienen premisas y conclusiones, pero son contradictorias entre sí. A continuación se detallan las características principales de cada una de estas técnicas:

- **Técnicas de aprendizaje deductivo**: utilizan hechos, información o conocimiento disponibles para deducir una conclusión válida, el enfoque que usan es de arriba a abajo, se mueven de una declaración generalizada a una conclusión válida y las conclusiones son ciertas. Los argumentos deductivos pueden ser válidos o inválidos, lo que significa que, si las premisas son verdaderas, la conclusión debe ser verdadera

- **Técnicas de aprendizaje inductivo**: implica hacer una generalización a partir de hechos específicos y observaciones, utilizan un enfoque ascendente, pasa de la observación específica a una generalización y las conclusiones son probabilísticas. El argumento inductivo puede ser fuerte o débil, lo que significa que la conclusión puede ser falsa incluso si las premisas son verdaderas.

Estos sistemas han experimentado un gran auge y desarrollo en los últimos años con el objetivo de permitir la comunicación hablada entre seres humanos y máquinas. Algunos ejemplos claros son *Alexa* de Amazon, *Siri* de Apple, el asistente de Google, *SmartThings* de Samsung, telefonía... Estos sistemas son capaces de procesar la señal de voz y reconocer la información que hay en dicha señal, convirtiéndola en texto (como en el reconocimiento de voz de windows) o en instrucciones para controlar una máquina (como en Alexa).

En este TFG se han desarrollado tres tipos de redes neuronales (técnicas de aprendizaje inductivo) para la clasificación de vocalistas, cuyas implementaciones conllevan varios desafíos. Es necesario de disponer de datos adecuados tanto para el entrenamiento de dichas redes como para comprobar los resultados una vez entrenadas. Por ello, se ha preparado un dataset de 400 pistas de audio de 30 segundos cada una, 100 para cada cantante, y en las que hay pistas *a capella* y pistas de canciones con todos instrumentos. Conseguir pistas *a capella* no es fácil, por lo que la mayoría del dataset está compuesto de pistas con todos instrumentos, lo que supone un reto extra para la resolución del problema. Asimismo, también será importante la elección de la arquitectura y el proceso de entrenamiento de las redes. Lo más complicado del problema es la extracción de características del audio. Primero es necesario dividir la señal en fragmentos (frames) para luego llevarlos al dominio de la frecuencia mediante ciertas transformaciones matemáticas, como la de Fourier o la del coseno discreto, además de otros filtros y procesos de normalización para obtener un vector de coeficientes representativo de la señal (los MFCC). Esto se explicará con más detalle a lo largo del documento.

El reconocimiento de voz ha sido estudiado en varios campos: distinción de voces de hombres y mujeres, distinción de géneros e instrumentos musicales, distinción de pájaros según el sonido que emiten al piar… Pero no se ha encontrado ningún trabajo específico en clasificar vocalistas. Los resultados experimentales obtenidos en este proyecto demuestran que este tipo de reconocimiento es factible y la precisión alcanzada puede ser considerablemente alta.

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVOS

En este trabajo vamos a utilizar los modelos denominados como redes neuronales artificiales para el reconocimiento de vocalistas en fragmentos de audio. Las redes neuronales artificiales son relativamente antiguas dentro del campo de la inteligencia artificial, puesto que fue en los años 40 y 50 cuando se empezaron a publicar los primeros artículos acerca de ellas. Sin embargo, no llegaron a obtener un desempeño aceptable en aquella época, ya que se necesitan una cantidad importante de recursos computacionales para entrenar y ejecutar una red neuronal. En los últimos años se han conseguido grandes avances gracias a la mejora de los computadores y al uso de GPUs para este tipo de algoritmos. Estas redes aprenden de la experiencia y generalizan de ejemplos previos a ejemplos nuevos mediante la abstracción de las características principales de una serie de datos.

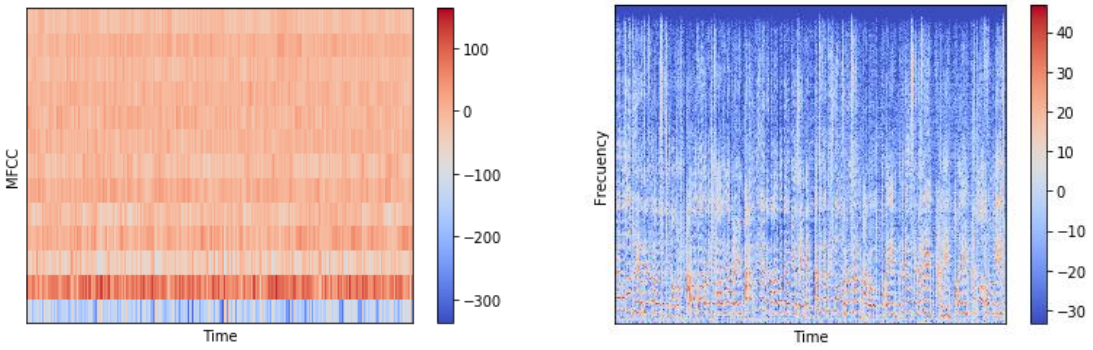
Algunos ejemplos de aplicación de redes neuronales los podemos encontrar en la red convolucional que creó Google para la opción *Street View* de Google Maps. Esta red alcanza una tasa de acierto del 96% a la hora de reconocer números de calle en las imágenes que toman sus coches. Se dice incluso, que las redes neuronales podrían acabar dominando uno de los juegos que se les resiste a los ordenadores: el juego de Go. En la Universidad de Edimburgo, unos investigadores han logrado usar redes convolucionales para detectar patrones en los tableros y tratar de sacar el mejor movimiento con una efectividad considerable.

En el presente trabajo, el problema a resolver consistirá en, dada una pista de audio de un vocalista o de una canción en la que cante dicho vocalista, crear una red capaz de averiguar quién está cantando. Para nosotros los humanos, es muy sencillo determinar quién está cantando cuando escuchamos una canción. De hecho, lo hacemos inconscientemente, y en caso de que no conozcamos a la persona que canta, basta con que nos digan quién es o buscarlo nosotros mismos con alguna app como Shazam o el asistente de Google y lo identificaremos para siempre en el futuro cuando lo volvamos a escuchar. Este proceso no es tan fácil para una red neuronal, ya que hay que transformar esa señal de audio en datos que la red “entienda”.

En los últimos años, la investigación en el reconocimiento de audio ha ganado impulso y ha sido usado en múltiples disciplinas como multimedia, monitorización bioacústica, detección de intrusos en áreas de vida silvestre, o sonidos ambientales. Algunas técnicas de aprendizaje automático como la *Mezcla Oculta y Gaussiana* (*Hidden and Gaussian Mixture*), *Perceptrón Multicapa* (*multi-layer perceptron*) y las *Redes Emergentes de Aprendizaje Profundo* (*Emerging Deep Learning Networks*) dieron como resultado una mejora del rendimiento de los sistemas de reconocimiento y clasificación de audio. Uno de los problemas dentro del reconocimiento de hablantes es la modificación de tono en la voz. En algunos casos, la persona a reconocer no puede controlar algunos cambios de su propia voz, por ejemplo, si la persona está enferma. La voz puede ser modificada por causas voluntarias e involuntarias. En el primer caso, esto ocurre cuando una persona, de manera consciente, hace alteraciones a su propia voz para no ser reconocido, como por ejemplo hablar más grave, cubrir la boca o taparse la nariz. Los cambios involuntarios se dan cuando una persona no puede controlar dichos cambios, por ejemplo, cuando tiene un resfriado, tos o está ronca.

El primer paso a dar para construir una red de reconocimiento de audio es el pre-procesado de las señales, el cual divide cada pista de audio en segmentos que se usarán en el siguiente paso para la extracción de sus características. Al dividir las pistas, aumenta el número de datos con el que entrenaremos dicha red a la vez que se reducirá su tamaño, para después extraer sus características y almacenarlas en vectores.

Hay dos maneras principales de procesar las señales de audio para convertirlos en datos "entendibles" para la red: la STFT (de sus siglas en inglés: *Short Time Fourier Transform*, o también llamado *Espectograma*) o los MFCC de la señal (de sus siglas en inglés: *Mel Frequency Cepstral Coefficient*). Los MFCC representan la amplitud del espectro del habla de manera compacta y son la técnica de extracción de características más usada en reconocimiento auditivo debido a su bajo costo computacional y su robustez (Figura 1.2a). En cambio, el espectrograma consiste en la representación gráfica del espectro de frecuencias del sonido, puede revelar rasgos como altas frecuencias o modulaciones de amplitud, que no pueden apreciarse incluso aunque estén dentro de los límites de frecuencia del oído humano (Figura 1.2b):



(a) MFCC (b) Espectograma

Figura 1.2 En la figura se muestra el MFCC y el espectograma de una pista de voz de Bruce Dickinson, vocalista de la banda de heavy metal Iron Maiden.

Una vez introducido el problema y los conceptos de una forma genérica, pasamos a definir los principales objetivos del proyecto:

- Elaboración de la base de datos (*dataset*) para el entrenamiento y evaluación de los algoritmos a desarrollar. Para ello, se han preparado un total de 100 pistas de audio, de 30 segundos de duración cada una, para cada uno de los 4 vocalistas que se han empleado en el presente proyecto, es decir, un total de 400 pistas.

- Implementación de un algoritmo capaz de procesar las señales de audio para conseguir transformar dichas señales en sus respectivos MFCCs.

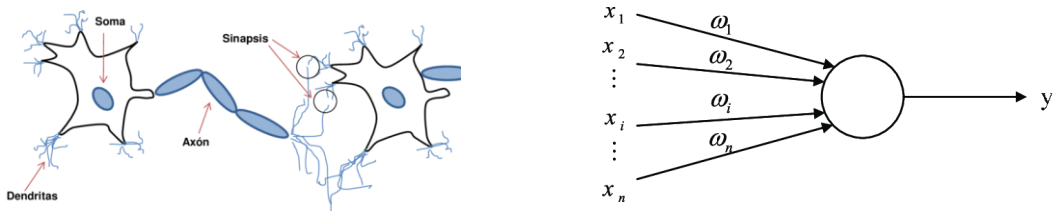
- Diseño e implementación de un algoritmo capaz de aprender a reconocer vocalistas mediante redes neuronales profundas. Para ello habrá que asegurarse de que se selecciona la arquitectura de red neuronal que tenga un mejor porcentaje de acierto.

A continuación, en el capítulo 2, se van a introducir los conceptos de los tres tipos principales de redes neuronales utilizados en el proyecto, necesarios para comprender su funcionamiento y, por tanto, el resto del trabajo. Más adelante, en el capítulo 3, se explicará la elaboración del dataset en profundidad junto a su pre-procesamiento antes de empezar a construir la red. En el capítulo 4 se explicarán y desarrollarán las distintas arquitecturas de redes utilizadas en el proyecto. En el capítulo 5 se mostrarán y explicarán los resultados obtenidos. El capítulo 6 constará de las conclusiones obtenidas del proyecto. Y, por último, en el anexo, se aportará información acerca de las herramientas utilizadas a lo largo del proyecto y el cronograma de las tareas desarrolladas.

2. REDES NEURONALES

2.1 Introducción

Las redes neuronales son una técnica de aprendizaje artificial inspirada en el mecanismo de aprendizaje biológico. El sistema nervioso humano contiene células, llamadas neuronas, que están conectadas entre sí mediante axones y dendritas, y a estas conexiones se les denomina sinapsis (Figura 2.1a). La fuerza de estas conexiones sinápticas a menudo cambia en respuesta a estímulos externos. Dicho cambio es el responsable del aprendizaje en el organismo. Este mecanismo es el que se trata de emular en las redes neuronales, las cuales contienen unidades de cómputo denominadas neuronas que están conectadas entre sí a través de pesos. Los pesos juegan el mismo papel que la fuerza de las conexiones sinápticas en los organismos biológicos. Cada entrada a la neurona se escala con un peso ω_n que afecta a la función calculada a esa unidad (Figura 2.1b). Una red neuronal artificial calcula una función de las entradas propagando los valores calculados desde las neuronas de entrada a las neuronas de salida y utilizando los pesos como parámetros intermedios.



(a) Neurona biológica

(b) Neurona artificial

Figura 2.1 Ejemplos de una neurona biológica y una artifical: (a) Forma y estructura de una neurona biológica. (b) Forma y estructura de una neurona artifical, donde las *x* se asemejarían a las dendritas, las *ω* serían los pesos, el círculo correspondería a la neurona y la *y* al axón.

El aprendizaje ocurre al cambiar los pesos que conectan las neuronas. Al igual que un organismo biológico necesita estímulos externos para aprender, el estímulo externo en redes neuronales artificiales es proporcionado por los datos de entrenamiento. Por ejemplo, los datos de entrenamiento pueden contener imágenes (entradas) y sus respectivas etiquetas (zanahoria, plátano, coche, bici...) como salida. Estos datos de entrenamiento alimentan a la red usando las representaciones de entrada para hacer predicciones sobre las etiquetas de salida y brindan retroalimentación a los pesos en la red neuronal, dependiendo de qué tan bien la salida predicha para una entrada particular (por ejemplo, la probabilidad de que sea una zanahoria) coincida con la etiqueta de salida anotada en los datos de entrenamiento. Los pesos entre neuronas son ajustados en respuesta al error en las predicciones. El objetivo de cambiar los pesos es modificar la función aproximada por la red, para que las predicciones sean más correctas en futuras iteraciones. Es por esto que los pesos son modificados cuidadosamente y de una forma justificada para reducir el error en la predicción. Al ajustar sucesivamente los pesos entre las neuronas, la función calculada por la red neuronal se refina con el tiempo para que proporcione

predicciones más precisas. Por tanto, si una red se entrena, por ejemplo, con muchas imágenes de diferentes coches, esta será capaz de reconocer qué es y qué no es un coche en una imagen similar a las de entrenamiento.

2.2 Arquitectura

2.2.1 Perceptrón

La red neuronal más simple es la denominada *perceptrón*. Esta red contiene una sola capa de entrada y un único nodo de salida cuyo valor solo puede ser 0 o 1 (ver Figura 2.1b). Para calcular la salida, el perceptrón realiza la siguiente operación matemática: $y = \sum_{i=1}^{n} \omega_i \cdot x_i$, donde *x* son las variables de entrada y ω sus respectivos pesos. Bien, pues dependiendo de si el resultado está por debajo o por encima de un umbral dado t, el resultado será 0 o 1 respectivamente. El perceptrón, por tanto, puede entenderse como un método de toma de decisiones.

$$y = \begin{cases} 0 \text{ si } \sum_{i=1}^{n} \omega_i \cdot x_i \leq t \\ 1 \text{ si } \sum_{i=1}^{n} \omega_i \cdot x_i > t \end{cases}$$

Pongamos un ejemplo muy sencillo para comprenderlo. Imaginemos que queremos ir a un festival de música y las variables binarias de entrada para decidir si vamos a ir o no son: el tiempo que va a hacer ($x_1 = 1$ si el tiempo va a ser bueno y $x_1 = 0$ si va a ser malo), si nuestra pareja quiere venir con nosotros o no ($x_2 = 1$ si es que sí y $x_2 = 0$ si es que no) y si hay buenos accesos al festival ($x_3 = 1$ si es que sí y $x_3 = 0$ si es que no). Supongamos que nuestro grupo favorito va a estar allí y queremos ir a toda costa, incluso si nuestra pareja no quiere venir o si no hay buenos accesos, aunque el tiempo sí que nos preocupa bastante. En tal caso, podríamos asignar un peso $\omega_1 = 6$ al tiempo, un peso $\omega_2 = 2$ a nuestra pareja y un peso $\omega_3 = 2$ a los accesos. Por último, el umbral que imponemos para tomar la decisión es 5:

El tiempo va a ser bueno → $x_1 = 1$
Nuestra pareja no quiere venir al final → $x_2 = 0$
Los accesos al festival no son buenos → $x_3 = 0$

$\sum_{i=1}^{n} \omega_i \cdot x_i = \omega_1 \cdot x_1 + \omega_2 \cdot x_2 + \omega_3 \cdot x_3 = 6 \cdot 1 + 2 \cdot 0 + 2 \cdot 0 = 6 \geq 5$ → Voy al festival ($y = 1$)

Por supuesto, el perceptrón no es un modelo completo que simule la toma de decisiones de un humano, lo que el ejemplo ilustra es cómo el perceptrón puede pesar diferentes datos de entrada para predecir el valor de salida de una función. De una manera similar, una red más compleja podría aproximar funciones más complejas.

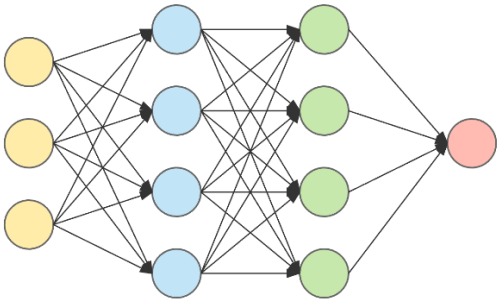


Figura 2.2 Red más compleja de perceptrones

En la red de la figura anterior, la primera columna de perceptrones son las variables de entrada. A esta primera columna se la llama *capa de entrada* (perceptrones amarillos). Esta capa está tomando 3 decisiones simples en función de las variables de entrada que tenga la red y sus respectivos pesos. Los perceptrones de las capas siguientes toman decisiones pesando el resultado de la capa anterior. De esta manera, cada perceptrón puede tomar decisiones en un nivel más complejo y abstracto que los perceptrones de la primera capa. Una red de perceptrones de muchas capas puede aproximar funciones más sofisticadas.

Para simplificar la condición $\sum_{i=1}^{n} \omega_i \cdot x_i$ vamos a hacer dos cambios. El primero es escribir esta misma expresión como resultado del producto $\omega \cdot x$ y el segundo es mover el umbral al otro lado de la inecuación y reemplazarlos por lo que se conoce como el *sesgo* del perceptrón:

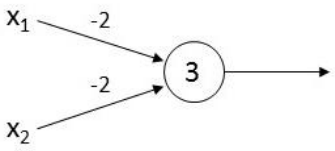
$$\omega \cdot x = \sum_{i=1}^{n} \omega_i \cdot x_i$$

$$b \equiv -t$$

Reescribiendo la inecuación anterior:

$$y = \begin{cases} 0 \text{ si } \omega \cdot x + b \leq 0 \\ 1 \text{ si } \omega \cdot x + b > 0 \end{cases}$$

Podemos pensar en el sesgo como una medida de lo fácil que es hacer que la salida del perceptrón sea 1. Si el sesgo es grande, será fácil para el perceptrón sacar un uno, mientras que, si este es negativo, será mucho más complicado. Imaginemos la siguiente situación:



Si el valor de las entradas x_1 y x_2 es 00, aplicando la fórmula de la inecuación, el perceptrón nos dará un 1 en la salida: $(-2)\cdot 0 + (-2)\cdot 0 + 3 = 3$, positivo → 1. Lo mismo ocurre cuando las entradas valen 01 y 10. Sin embargo, cuando las entradas valen 1 y 1, obtenemos 0 en la salida: $(-2)\cdot 1 + (-2)\cdot 1 + 3 = -1$, negativo. Nuestro perceptrón implementa una puerta NAND. Este ejemplo muestra que podemos usar los perceptrones para calcular funciones lógicas simples.

De hecho, se pueden usar redes de perceptrones para aproximar cualquier función. La razón es que la puerta NAND es universal para el cálculo, es decir, se puede construir cualquier cálculo a partir de puertas NAND, lo que demuestra que los perceptrones también son universales para el cálculo y se pueden usar para calcular cualquier función lógica. Esto evidencia que se pueden idear algoritmos de aprendizaje capaces de ajustar automáticamente los pesos y sesgos de una red de neuronas artificiales. Esta sintonización ocurre en respuesta a estímulos externos, sin intervención directa de un programador.

Estos algoritmos de aprendizaje nos permiten usar neuronas artificiales de una manera radicalmente diferente a las puertas lógicas convencionales. En vez de diseñar explícitamente un circuito de puertas NAND y otras puertas, nuestras redes neuronales simplemente pueden aprender a resolver problemas para los que a veces sería extremadamente difícil diseñar directamente un circuito convencional.

2.2.2 Neurona sigmoidea

La representación de esta neurona es similar a la del perceptrón (ver Figura 2.1b). La diferencia es que el valor de las entradas puede ser cualquier valor real y el valor de la salida pasa a ser $\sigma(\omega \cdot x + b)$, por lo que la salida tampoco será 0 o 1, sino que podrá tomar cualquier valor dentro de este intervalo. La función sigmoidea σ viene definida por:

$\sigma(z) = \frac{1}{1+e^{-z}}$, si lo escribimos de una forma más explícita: $\sigma(z) = \frac{1}{1+e^{-\sum_j \omega_j \cdot x_j - b}}$

A priori, esta neurona puede parecer muy diferente de un perceptrón, pero no lo es tanto. Imaginemos que z es un número muy grande, en tal caso e^{-z} tiende a 0, y, por tanto, la salida de la neurona tenderá a 1. Por el contrario, si z es un número muy negativo, e^{-z} tiende a ∞, y la salida de la neurona tenderá a 0. Solo cuando $\omega \cdot x + b$ tome valores pequeños habrá diferencia respecto al modelo del perceptrón.

Imaginemos que le damos como entrada a una red neuronal una imagen en la que sale el número 9 escrito por nosotros, y queremos que esta red nos diga si es ese número o no. Obviamente, sería más fácil si la salida solo pudiera ser “sí” o “no” como en un perceptrón, pero en la práctica podemos establecer una convención para tratar esto, por ejemplo, al decidir interpretar cualquier salida de más de 0.5 como "es un 9", y cualquier salida inferior a 0.5 como "no es un 9".

2.2.3 Arquitectura de las redes neuronales

Una vez descrito lo anterior, pasamos a definir la arquitectura general de las redes neuronales. Una red neuronal tiene la forma que se observa en la Figura 2.3, donde la capa de más a la izquierda se llama *capa de entrada* y sus neuronas son denominadas *neuronas de entrada*, la capa de más a la derecha se llama *capa de salida* y contiene las *neuronas de salida* y por último las capas intermedias denominadas *capas ocultas* porque las neuronas no son ni entradas ni salidas.

El diseño de las capas de entrada y salida de las redes neuronales suele ser sencillo. Por ejemplo, supongamos que estamos tratando de determinar si una imagen escrita a mano representa un "9" o no. Una forma natural de diseñar la red es codificar las intensidades de los píxeles de la imagen en neuronas de entrada. Si la imagen está en escala de grises y es de 28 por 28 píxeles, tendríamos un total de 784 píxeles (28x28), y, por tanto, 784 neuronas de entrada, con las intensidades escaladas apropiadamente entre 0 y 1. La capa de salida contendrá una única neurona de valores de salida inferiores a 0.5 si “no es un 9” o superiores a 0.5 si “sí es un 9”. Supongamos una red neuronal que tenga la estructura de la Figura 2.3:

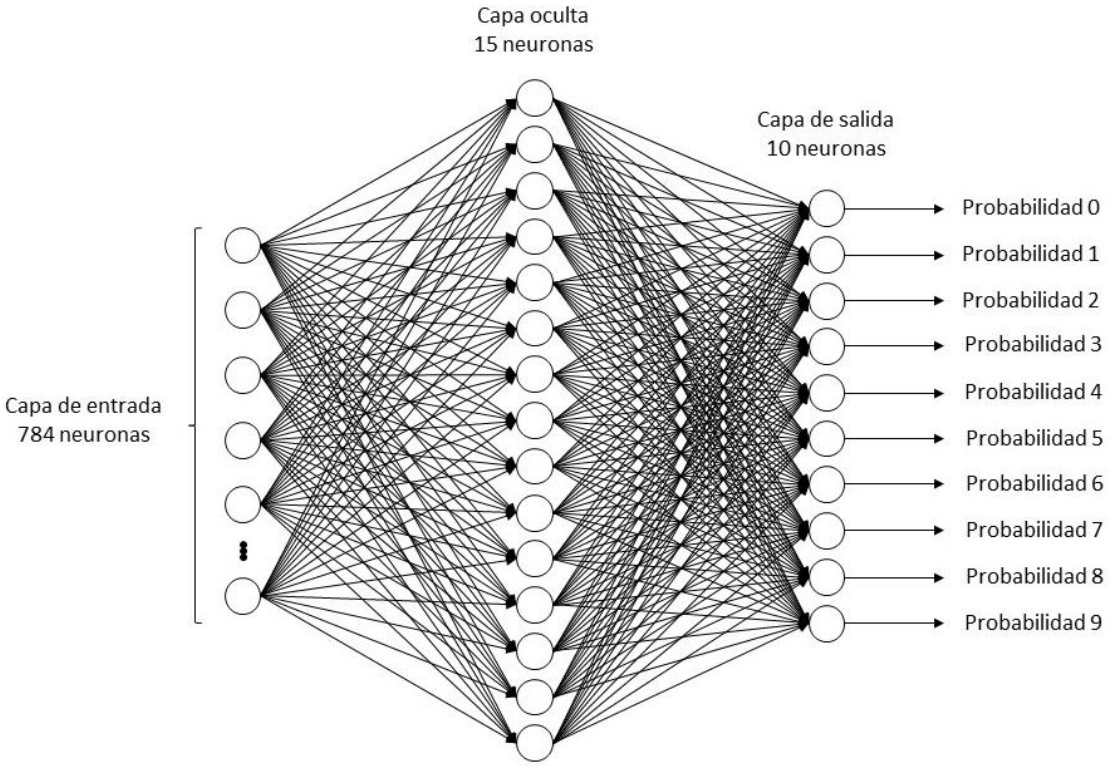


Figura 2.3 Arquitectura de las redes neuronales

Tenemos por un lado la capa de entrada, que serán imágenes de 28 por 28 píxeles, es decir, 28 x 28 = 784 neuronas, en las que aparecerán dígitos escritos a mano (por simplicidad se han omitido el resto de neuronas de esta primera capa que no aparecen en la imagen). Estos píxeles están en escala de grises y tendrán un valor 0.0 cuando sea blanco, 1.0 cuando sea negro, y un valor comprendido entre 0.0 y 1.0 para tonos grises dependiendo de su oscuridad. La segunda capa de la red es la capa oculta que consta de 15 neuronas. Por último, la capa de salida contiene 10 neuronas que se corresponden con los 10 dígitos 0 ,1, 2, 3, 4, 5, 6, 7, 8, y 9. Los valores de estas 10 neuronas de la capa de salida van a ser valores comprendidos entre 0 y 1 y nos van a indicar, según lo que crea la red, la probabilidad de que la imagen introducida sea un número u otro. Si, por ejemplo, la última neurona de la capa de salida nos da un valor de 1, significa que la red estará segura de que la imagen que le hemos introducido es el número 9, mientras que, si nos da un 0, significaría que la red está segura de que el dígito de la imagen no es un 9.

Este tipo de redes neuronales donde las salidas de una capa son entradas de la siguiente son denominadas *redes neuronales feedforward densamente conectadas*. Esto significa que no hay

bucles en la red, la información siempre se transmite, pero nunca se retroalimenta. Las redes en las que sí hay bucles se denominan *redes neuronales recurrentes*, y la idea en estos modelos es tener neuronas que se disparen durante un tiempo limitado, estas las veremos en profundidad más adelante.

Las redes neuronales son modelos de caja negra, los pesos y sesgos de las redes que hemos explicado se ajustan automáticamente, y eso significa que no tenemos una explicación inmediata de cómo las redes son capaces de hacer lo que hacen. Imaginemos que queremos determinar si una imagen muestra una cara de una persona o no. El planteamiento sería el mismo que en el reconocimiento de dígitos, los píxeles de la imagen serían las entradas a la red y en la salida tendríamos una única neurona que nos diría "sí" o "no".

Si queremos descifrar el problema nosotros diseñando una red manualmente y escogiendo pesos y sesgos apropiados, una heurística que podríamos usar es descomponer el problema en problemas más sencillos: ¿tiene la imagen un ojo en la parte superior derecha? ¿Y en la superior izquierda? ¿Y tiene una nariz en el medio? ¿Hay boca en la parte inferior? Si la respuesta a estas preguntas es "sí" o "probablemente sí", podremos concluir que probablemente la imagen será una cara. Por supuesto, esto es solo una heurística aproximada y sufre de muchas deficiencias, pues puede que la persona sea calva, o que solamente aparezca parte de la cara en la imagen, o que el ángulo del que se ha tomado la imagen ha hecho que algunas partes de la cara salgan oscuras y no se reconozcan con facilidad... Aun así, la heurística sugiere que, si podemos resolver los sub-problemas utilizando redes neuronales, entonces quizás podamos construir una red neuronal para la detección de rostros, combinando las redes para los sub-problemas.

Una posible arquitectura de red para este problema podría ser la de la Figura 2.4, donde los rectángulos denotan las redes de los sub-problemas. Es obvio que las sub-redes podrían también ser descompuestas en otras sub-redes que respondan cada vez preguntas más simples hasta llegar al nivel del píxel. El resultado sería una red que desglosa una pregunta compleja genérica en muchas preguntas simples y concretas. Esto lo hace la red a partir de una serie de muchas capas con capas iniciales que responden preguntas muy simples y específicas sobre la imagen de entrada, y capas posteriores que forman una jerarquía de conceptos cada vez más complejos y abstractos. Estas redes con este tipo de estructura multicapa (2 o más capas ocultas) se denominan ***Redes Neuronales Profundas*** (*Deep Neural Networks*).

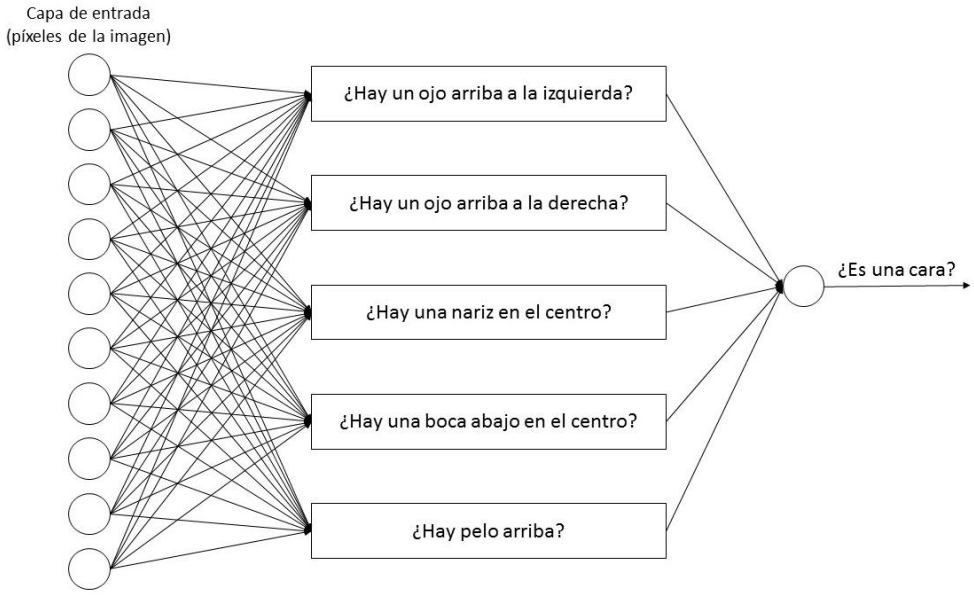
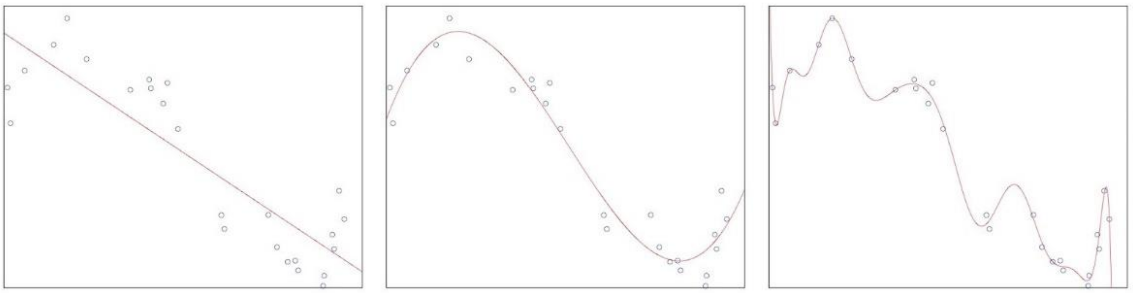


Figura 2.4 Arquitectura para una red neuronal de detección facial

A veces, al entrenar las redes neuronales, se puede producir infrajuste o sobreajuste. El primer caso ocurre cuando nuestro modelo sólo se ajusta a aprender los casos particulares que le enseñamos y es incapaz de reconocer nuevos datos de entrada, mientras que el segundo caso se produce cuando nuestro algoritmo está considerando como válidos sólo los datos idénticos a los de nuestro conjunto de entrenamiento y siendo incapaz de distinguir entradas buenas si se salen un poco de los rangos ya preestablecidos. En la siguiente imagen se muestra el ejemplo de la situación de ambos casos, y de la situación ideal:



Infrajuste Ajuste ideal Sobreajuste

Para solventar este problema, existe el algoritmo conocido como *Dropout*. Esta función es un método que desactiva un numero aleatorio de neuronas de una capa, y en cada iteración del entrenamiento de la red, las neuronas desactivadas no se toman en cuenta, lo que obliga a las neuronas cercanas a no depender tanto de las neuronas desactivadas. Dropout tiene un parámetro que indica la probabilidad de que las neuronas se queden activadas y este es elegido por el programador de la red, este parámetro toma valores de 0 a 1.

2.2.4 Hiperparámetros

Los hiperparámetros son parámetros que rigen todo el proceso de entrenamiento. Incluyen variables que determinan la estructura de la red (por ejemplo, número de capas ocultas) y las variables que determinan cómo se entrena la red (por ejemplo, la tasa de aprendizaje, también conocido como *learning rate*). Los hiperparámetros del modelo se establecen antes del entrenamiento (antes de optimizar los pesos y el sesgo) y son importantes ya que controlan directamente el comportamiento del algoritmo de entrenamiento, teniendo un impacto importante en el rendimiento del modelo en el entrenamiento. Las librerías de aprendizaje automático suelen tener la opción de usar valores por defecto, pero estos se pueden modificar para buscar una mejor optimización. Los modificados en este proyecto han sido los siguientes:

- **Épocas**: consiste en una iteración del entrenamiento de la red con todos los datos disponibles. Para cada época, el algoritmo de aprendizaje construye un modelo diferente, es decir, una red con un conjunto diferente de pesos. La forma de saber su valor es hacer que el modelo entrene durante la mayor cantidad de iteraciones siempre que el error de validación siga disminuyendo.

- **Batch Size**: define el número de datos de entrada a los que la red está expuesta antes de que los pesos se actualicen dentro de cada época. Es un parámetro relacionado con la eficiencia, asegurando que no se carguen demasiados patrones de entrada a la vez. Un batch size más grande permite mejoras computacionales que utilizan la multiplicación de matrices en los cálculos de entrenamiento. Sin embargo, se produce a expensas de necesitar más memoria para el proceso de entrenamiento. Un tamaño más pequeño induce más ruido en los cálculos de error, y puede ser más adecuado para evitar que el proceso de capacitación se detenga en los mínimos locales. Un valor razonable para este TFG, y para gran parte de la bibliografía analizada, podría ser 32.

- **Learning Rate**: Es el tamaño del paso que se realiza en cada iteración del entrenamiento, en la dirección del gradiente descendiente. Si es demasiado pequeño, al entrenamiento le costará mucho tiempo, cientos o miles de épocas, alcanzar un estado con errores pequeños. Por otro lado, si la tasa de aprendizaje es muy grande el algoritmo podría no converger. Un learning rate inicial razonable podría ser 0,001.

2.3 Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN por sus siglas en inglés *Convolutional Neural Networks*) son el tipo de red neuronal profunda más utilizada sobre todo en imágenes. A pesar de lo explicado anteriormente, no es habitual usar redes como las del capítulo anterior (redes neuronales *feedforward* densamente conectadas, Figura 2.3) para clasificar imágenes, ya que dicha arquitectura de red no tiene en cuenta la estructura espacial de las imágenes. Las CNN son las redes que más se usan hoy en día para el reconocimiento de imágenes porque utilizan una arquitectura especial que está adaptada a las características de los datos. El uso de esta arquitectura hace que las CNN sean rápidas de entrenar y esto, a su vez, nos ayuda a entrenar

redes profundas de muchas capas. Las CNN utilizan 3 ideas básicas: campos receptivos locales, pesos compartidos y agrupación (esta última llamada *Pooling*):

- **Campos receptivos locales**: en las capas completamente conectadas mostradas anteriormente, las entradas se representaron como una línea vertical de neuronas. En una red convolucional, será útil pensar en las entradas como un cuadrado de x por x neuronas, 28x28 si usamos el ejemplo anterior, cuyos valores corresponden a las intensidades de los 28x28 píxeles que estamos usando como entradas (Figura 2.5a). Como de costumbre, los píxeles de entrada irán conectados a una capa de neuronas ocultas, pero no se conectará cada píxel de entrada a cada neurona oculta. En su lugar, solo se harán conexiones en pequeñas regiones localizadas de la imagen de entrada (Figura 2.5b). Esta región es lo que se llama campo receptivo local para la neurona oculta, es una pequeña ventana sobre los píxeles de entrada. Cada conexión aprende un peso y la neurona oculta aprende un sesgo general también.

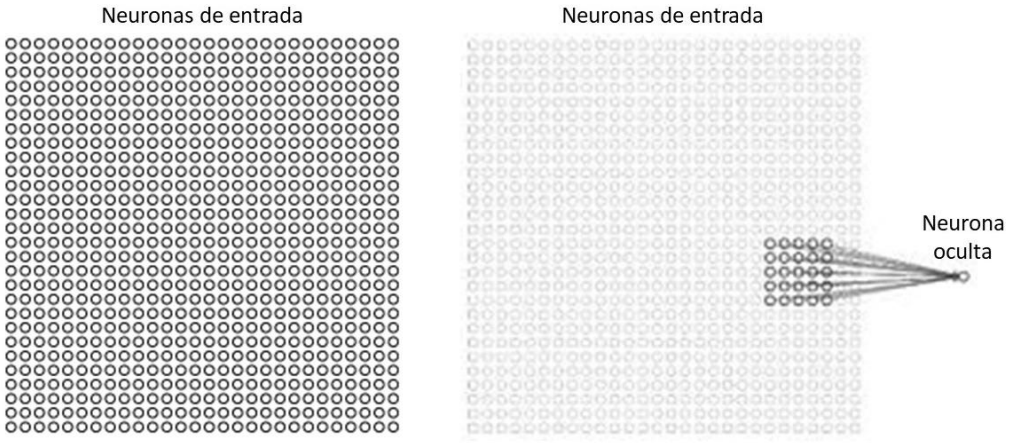


Figura 2.5 Apariencia de una capa convolucional: (a) Capa de entrada de 28x28 neuronas correspondientes a los 28x28 píxeles de la imagen de entrada. (b) Muestra de cómo se conexionan las neuronas en las CNN.

Se puede pensar que esta neurona oculta particular aprende a analizar su campo receptivo local particular. Luego deslizamos el campo receptivo local por toda la imagen de entrada. Para cada campo receptivo local, hay una neurona oculta diferente en la primera capa oculta. Para ilustrar esto concretamente, empezamos con un campo receptivo local en la esquina superior izquierda y lo vamos desplazando de pixel en pixel, de izquierda a derecha, y de arriba abajo:

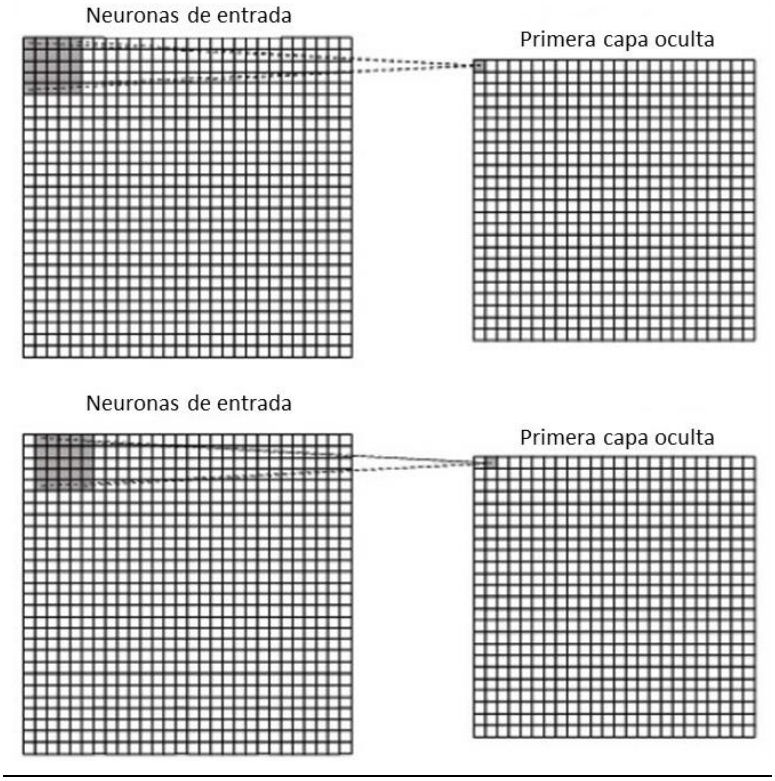


Figura 2.6 Representación del aprendizaje en las CNN

Este campo receptivo local se puede mover saltando la cantidad de píxeles que se quiera, no tiene por qué ser necesariamente de 1 en 1. En este caso, con el filtro 5x5, la capa oculta nos va a quedar de 24x24 neuronas.

- **Pesos y sesgo compartidos**: vamos a utilizar los mismos pesos y sesgos para cada una de las 24x24 neuronas ocultas, lo que significa que todas neuronas de la primera capa oculta detectan exactamente la misma característica, solo que en diferentes localizaciones de la imagen de entrada. Supongamos que los pesos y el sesgo son tales que la neurona oculta puede detectar, por ejemplo, un borde vertical en un campo receptivo local particular. Es probable que esa habilidad también sea útil en otros lugares de la imagen. Por tanto, es útil aplicar el mismo detector de características en todas partes de la imagen. Dicho de otra forma, las redes convolucionales están bien adaptadas a la invariancia de traslación de las imágenes. Si mueves la imagen de un gato un poco, sigue siendo una imagen de un gato. Por esta razón llamamos *mapa de características* al mapa que va de la capa de entrada a la capa oculta, y los pesos y sesgos compartidos son lo que definen dicho mapa.

  Se suele decir que los pesos y sesgos compartidos definen un *kernel* o *filtro*. La estructura de red descrita hasta ahora puede detectar un solo tipo de característica localizada. Para hacer el reconocimiento de imágenes, necesitaremos más de un mapa de características. Así pues, una capa convolucional completa consta de varios mapas de características diferentes:

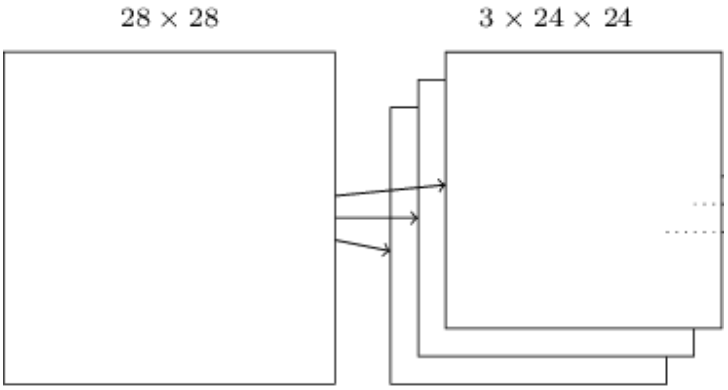


Figura 2.7 Mapas de características de una CNN

En el ejemplo que se muestra, hay 3 mapas de características. Cada mapa de características está definido por un conjunto de pesos compartidos y un solo sesgo compartido. El resultado es que la red puede detectar 3 tipos diferentes de características, con cada característica siendo detectable en toda la imagen (se han mostrado solo 3 por simplificar, en realidad habría muchísimos mas).

Veamos un ejemplo: en la Figura 2.8 se muestran 20 imágenes correspondientes a 20 mapas de características (o *kernels* o *filtros*) diferentes. Cada mapa es representado por un bloque de imagen de dimensiones 5x5, correspondiente a los pesos 5x5 en el campo receptivo local. Los bloques más blancos significan un peso más pequeño (típicamente, más negativo), por lo que el mapa de características responde menos a los píxeles de entrada correspondientes. Los bloques más oscuros significan un peso mayor, por lo que el mapa de características respode más a los píxeles de entrada correspondientes.

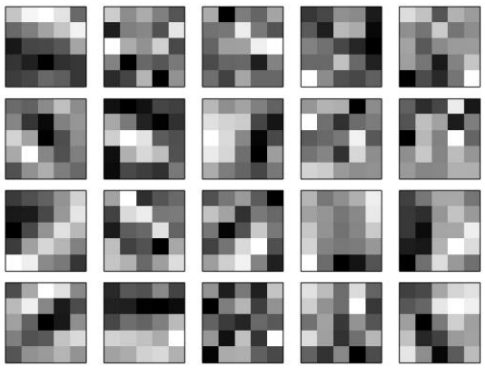
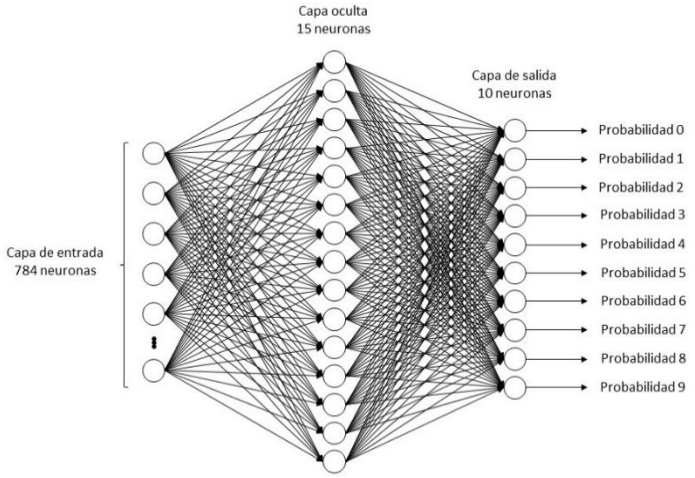


Figura 2.8 Ejemplos de mapas de características

En términos generales, las imágenes de la Figura 2.8 muestran el tipo de características a las que las CNN responden. Está claro que aquí hay una estructura espacial más allá de lo que esperaríamos al azar. Muchas de las características tienen sub-regiones claras de luz y oscuridad. Eso muestra que nuestra red realmente está aprendiendo patrones relacionadas con la estructura espacial. Sin embargo, más allá de eso, es difícil entender qué están aprendiendo.

Una enorme ventaja de los pesos y sesgos compartidos es que reduce considerablemente el número de parámetros involucrados en la red. Para cada mapa de caraterísticas necesitamos 5x5 = 25 pesos compartidos más un único sesgo compartido, por lo que cada mapa de características requiere 26 parámetros. Si tenemos 20 mapas de características, eso es un total de 20 x 26 = 520 parámetros definiendo la capa convolucional. En cambio, si tuviéramos una red *feedforward* densamente conectada como la de la imagen con 15 neuronas ocultas:



necesitaríamos un total de 784 neuronas x 15 pesos más 15 sesgos extra = 11.775 parámetros. El uso de la capa convolucional reducirá el número de parámetros que necesita para obtener el mismo rendimiento que el modelo completamente conectado *feedforward*. Esto se traduce en un entrenamiento más rápido.

- **Agrupación (pooling)**: las CNN también tienen capas de agrupación, y estas, son usadas comúnmente después de las capas convolucionales. Lo que hacen estas capas es simplificar la información en la salida de dichas capas. Una capa de agrupación toma cada salida del mapa de características de la capa convolucional y prepara un mapa de características condensado. Por ejemplo, cada neurona de la capa de agrupación puede salir de una región de, por ejemplo, 2x2 neuronas en la capa convolucional anterior. Este proceso se conoce como Max-Pooling. En la Figura 2.9 se puede ver cómo funciona este proceso. El resultado será una capa de 12x12 neuronas

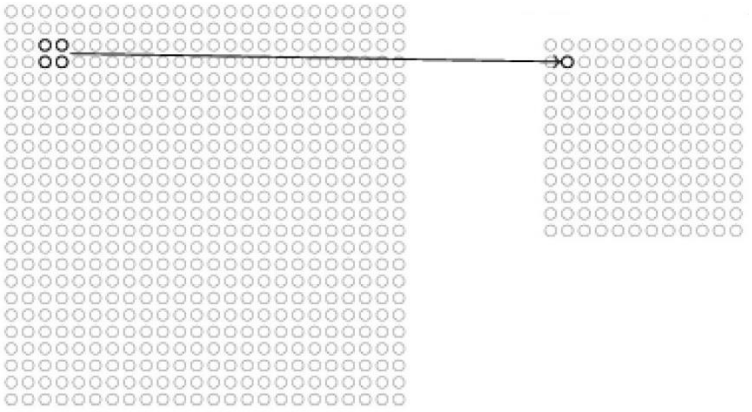
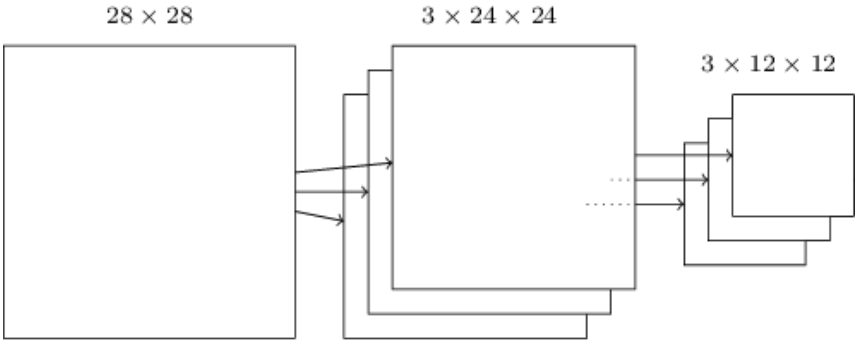


Figura 2.9 Muestra de aplicar Max-Pooling a una capa convolucional

Al aplicar esta técnica, nos quedará la red de la siguiente manera:



Podemos pensar en esta técnica como una forma para que la red pregunte si una característica determinada se encuentra en algún lugar de una región de la imagen. La intuición es que una vez que se ha encontrado una característica, su ubicación exacta no es tan importante como su ubicación aproximada en relación con otras características. Una gran ventaja es que hay muchas menos características agrupadas, por lo que esto ayuda a reducir la cantidad de parámetros necesarios en las capas posteriores.

Ahora que hemos explicado las 3 ideas básicas en las que consisten las CNN, podemos juntarlas todas para formar una red neuronal convolucional completa. Es similar a la arquitectura que estábamos viendo, pero tiene la adición de una capa de 10 neuronas de salida, que corresponde a los 10 valores posibles para los dígitos (Figura 2.10). La última capa es una capa *feedforward* densamente conectada, y sirve para conectar todas las neuronas desde la capa Max-Pooling a las 10 neuronas de la capa de salida que se corresponden con los dígitos (se ha utilizado una única flecha para conectar la Max-Pooling a la feedforward por simplicidad, pero la idea sería que todas neuronas de la Max-Pooling estarían conectadas con todas las de la capa *feedforward*). La transformación de una capa convolucional a una feedforward se realiza mediante la función *Flatten()*.

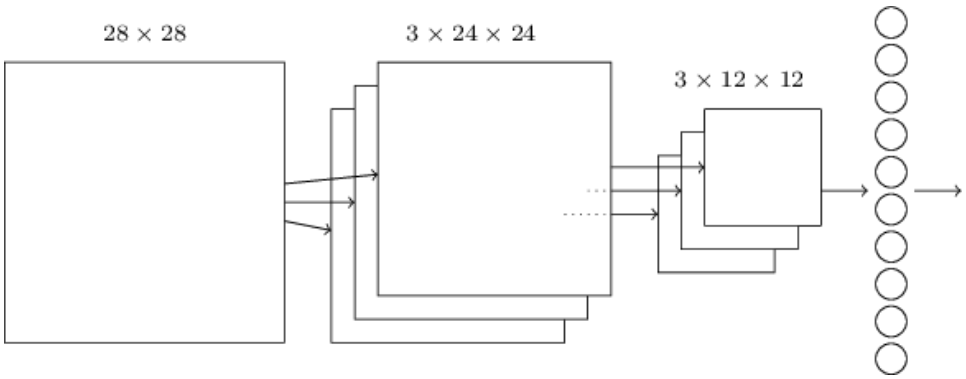


Figura 2.10 Ejemplo de una red neuronal convolucional completa

2.4 Redes Neuronales Recurrentes (RNN)

Las redes neuronales recurrentes (RNN por sus siglas en inglés *Recurrent Neural Networks*) se caracterizan por la presencia de nodos auto conectados y se usan en muchos modelos y aplicaciones de procesamiento temporal. Estos nodos permiten a la red memorizar y realizar un seguimiento de las entradas anteriores para predecir lo que viene después, lo que hace posible almacenar y acceder a información durante largos períodos de tiempo.

A un nivel alto, podríamos definir el funcionamiento de este tipo de redes de la siguiente forma: entra un dato de entrada a la red, la neurona hace un procesamiento del dato y nos da como resultado el estado (que memoriza el estado de la neurona en ese instante de tiempo) y la salida de la neurona. El estado va a ser reutilizado en el siguiente paso para tener información acerca del contexto en el que nos encontramos, en el siguiente paso tenemos el nuevo dato que entra a la red y es procesado de la misma forma, y, además, el estado anterior. Por tanto, la idea básica aquí es que estamos introduciendo información a la red secuencialmente, paso a paso, y obtenemos la salida y el estado actual de una neurona en función del estado anterior y la entrada actual. Así pues, finalmente solo nos quedaríamos con la salida del último paso, descartando los resultados anteriores.

La arquitectura básica de una RNN se caracteriza por una conectividad recurrente formando bucles de retroalimentación. Las conexiones de retroalimentación se ejecutan entre las unidades de la capa oculta del modelo, permitiendo que el patrón de activación en esta capa en cada paso influya en su comportamiento en el siguiente paso. Esta arquitectura se muestra en la Figura 2.11, y en ella tenemos una secuencia de vectores como entrada $\{x_1, x_2, \ldots, x_T\}$, una secuencia de estados ocultos $\{s_1, s_2, \ldots, s_T\}$ (o también $\{h_1, h_2, \ldots, h_T\}$) y una secuencia de salida $\{o_1, o_2, \ldots, o_T\}$. Los pesos de conexión del sistema se establecen mediante la operación de un procedimiento de aprendizaje que ajusta los pesos gradualmente a lo largo de muchas iteraciones para reducir el error de salida durante la ejecución de la tarea de recuperación en serie.

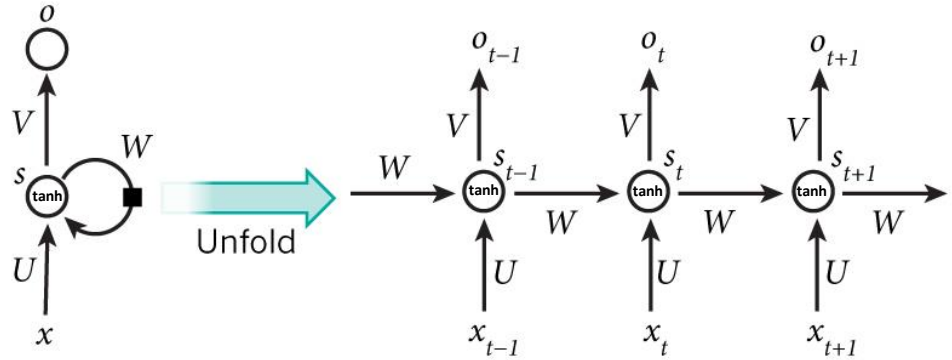


Figura 2.11 Arquitectura básica de una RNN

Una capa recurrente aproxima una función recursiva f, que toma como entrada un vector de entrada x_t y el estado oculto anterior h_{t-1} y devuelve el nuevo estado oculto y la salida como:

$$s_t = \tanh(U \cdot x_t + W \cdot s_{t-1})$$

$$o_t = \text{softmax}(V \cdot s_t)$$

donde U, W y V son matrices de pesos que se comparten en todos los pasos que se usan para calcular el estado de salida, la función de activación tanh es la función tangente hiperbólica y *softmax* sirve para pasar de la última capa recurrente a una capa feedforward densamente conectada para obtener así la salida. Sin embargo, se sabe que esta función recursiva tiene el defecto de que dificulta el aprendizaje para las dependencias a largo plazo. Para superar este problema, se introdujo un tipo de RNN con memoria a largo plazo llamada LSTM (por sus siglas en inglés *Long Short-Term Memory*). Estas utilizan una función más complicada que aprende patrones a largo plazo con mayor facilidad y recuerdan información durante largos períodos de tiempos.

Una celda LSTM (Figura 2.12) consta de cuatro subunidades: puerta de entrada, puerta de salida, puerta de olvido y el vector de estado secundario c_t, y los cuadros amarillos que contiene en la parte inferior son capas feedforward con las funciones de activación sigmoideas y tangente hiperbólica.

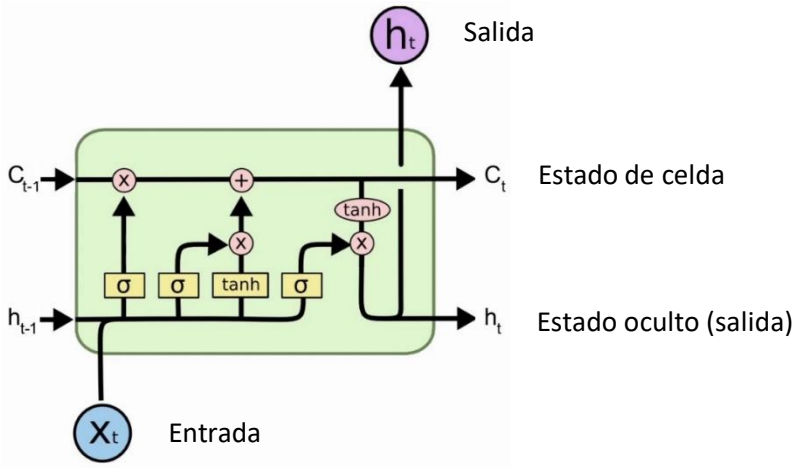


Figura 2.12 Estructura de una celda de una RNN-LSTM

El primer paso de una RNN-LSTM es decidir qué información vamos a desechar. Esta decisión la toma una capa sigmoidea llamada "puerta de olvido" (Figura 2.13). Esta capa tiene como entradas h_{t-1} y x_t, y al aplicar la función sigmoidea genera un número entre 0 y 1 para cada número en el estado de celda C_{t-1}. Un 1 representa "mantener completamente esto", mientras que un 0 representa "deshacerse completamente de esto".

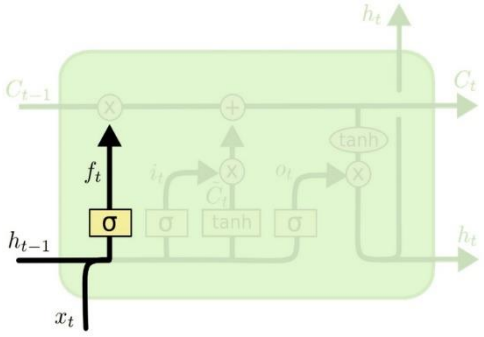


Figura 2.13 Puerta de olvido de una LSTM

El resultado de esta rama se calcula mediante la siguiente función:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Se concatena el vector de entrada con el vector del estado anterior y aplicamos la función sigmoidea. b_f es un sesgo como en redes anteriores y W_f es la matriz de pesos para esta capa de la puerta de olvido

Para saber qué va a olvidar la red, un ejemplo muy simple es el siguiente:

Estado de la celda anterior C_{t-1}: (1,2,4)
Resultado de la capa sigmoidea f_t: (1,0,1)

¿Qué recordará y qué olvidará la red? → $C_{t-1} \cdot f_t = [1,2,4] \cdot [1,0,1] = [1,0,4]$

La red recordará el 1 y el 4 porque al tener pesos de 1 en las posiciones primera y tercera del vector f_t considera que es información importante, mientras que el 2 decidirá olvidarlo porque al tener 0 en su posición, considera que no es importante.

El siguiente paso es decidir qué nueva información vamos a almacenar en el estado de la celda. Esto tiene dos partes, primero una capa sigmoidea llamada "puerta de entrada" (Figura 2.14) decidirá qué valores actualizaremos, y a continuación, una capa tanh creará un vector de nuevos valores candidatos, $\hat{C}_t$, que podrían agregarse al estado. En el siguiente paso, combinaremos estos dos para crear una actualización del estado.

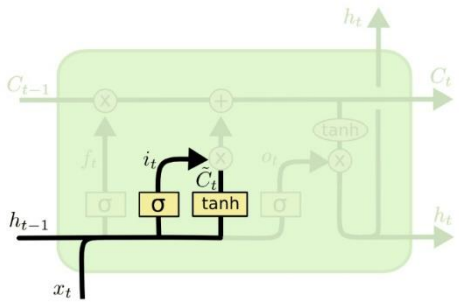


Figura 2.14 Puerta de entrada de una LSTM

El resultado de esta rama se calcula con las operaciones:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_C)$$

La forma de calcular lo que va a entrar a la LSTM es exactamente igual que en el caso anterior para saber qué recordar y qué olvidar.

$$\hat{C}_t \cdot i_t$$

El paso que sigue, va a consistir en actualizar el antiguo estado de la celda, C_{t-1}, en el nuevo estado de la celda C_t. Los pasos anteriores ya decidieron qué hacer, ahora solo hay que hacerlo. Para ello simplemente hay que multiplicar el estado anterior por f_t, olvidando las cosas que se decidieron olvidar antes y luego sumando $i_t \cdot \hat{C}_t$ (tal como muestra la Figura 2.15).

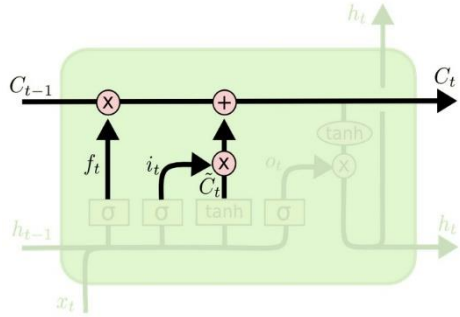


Figura 2.15 Actualización estado de celda LSTM

El resultado del nuevo estado se calcula de la siguiente manera:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$$

Simplemente cogemos los dos resultados anteriores y los sumamos, tal como se ve en la imagen. Así pues, ya tenemos el estado actual actualizado.

Finalmente, tenemos que decidir qué vamos a generar. Esta salida se basará en nuestro estado de celda, pero será una versión filtrada por la capa sigmoidea de la rama en la que nos

encontramos. Primero, ejecutamos esta capa sigmoidea que decide qué partes del estado de la celda vamos a generar. Luego, colocamos el estado de la celda en tanh para que los valores estén entre -1 y 1 y lo multiplicamos por la salida de la puerta sigmoidea, de modo que solo generemos las partes que decidimos. Esta capa se llama “puerta de salida” (Figura 2.16).

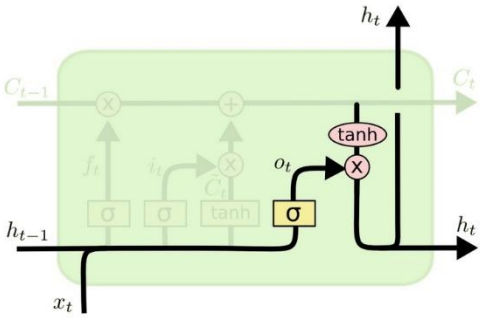


Figura 2.16 Puerta de salida de una LSTM

El resultado de la salida se calcula como:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$

o_t se calcula de la misma manera que i_t, y h_t es el nuevo estado oculto que será la salida de la celda actual, así como el estado h_{t-1} para la siguiente celda.

De esta manera, quedan concluidas explicaciones de los dos tipos principales de redes neuronales que se han utilizado en el proyecto. En el capítulo 3 a continuación, se explicará el proceso para la elaboración del dataset y el pre-procesamiento del audio que hay que llevar a cabo antes de implementar las redes explicadas.

3. DATASET Y PREPROCESADO

En el presente capítulo se van a explicar los MFCCs (qué son y como extraerlos), los pasos para la recogida de caraterísticas y el pre-procesamiento previo al entrenamiento de la red. En primer lugar, hay que crear el dataset, para después realizar el pre-procesamiento de las pistas antes de extraer los MFCCs.

3.1 MFCC

Los MFCCs (Coeficientes Cepstrales en las Frecuencia de Mel) han sido las características dominantes utilizadas para el reconocimiento de voz durante algún tiempo ya que tratan de imitar la forma en que los humanos percibimos el sonido, y esto es debido a su capacidad para representar el espectro de amplitud de la voz en una forma compacta. La característica fundamental de los MFCCs es que son capaces de diferenciar los timbres, es decir, que si un violín y una trompeta están emitiendo la misma nota (misma frecuencia), el MFCC reconoce cual es el violín y cual la trompeta por sus timbres, o lo que es lo mismo, por la cantidad de armónicos que tenga cada onda y las intensidades de los mismos. Los MFCCs son coeficientes que dependerán de la frecuencia y la intensidad en cada instante de tiempo de la señal, y estos coeficientes son los que utilizará la red para aprender.

El primer paso para la extracción del MFCC es dividir la señal en frames, o ventanas, en intervalos fijos de tiempo. El objetivo es modelar pequeñas secciones de la señal que son estadísticamente estacionarias y generar un vector de características cepstrales para cada frame.

El segundo paso es hacer la transformada discreta de Fourier de cada frame para pasar al dominio de la frecuencia, y acto seguido se toma el logaritmo del espectro de amplitud. Se descarta la información de la fase porque varios estudios perceptuales han demostrado que la amplitud del espectro es mucho más importante que la fase. Tomamos este logaritmo porque el ruido percibido de una señal es aproximadamente logarítmico.

El siguiente paso es suavizar el espectro y enfatizar frecuencias perceptivamente significativas. Esto se logra mediante la recopilación de componentes espectrales en intervalos de frecuencia. Aunque uno esperaría que estos intervalos estuvieran igualmente espaciados, lo cierto es que no tiene por qué ser así, ya que, para el habla, las frecuencias más bajas son perceptualmente más importantes que las frecuencias más altas. Por lo tanto, el espacio del intervalo sigue la llamada “escala de frecuencia de Mel” (Figura 3.1). La escala de Mel se basa en un mapeo entre la frecuencia real y el tono percibido. Esto se debe a que, aparentemente, el sistema auditivo humano no percibe el tono de manera lineal. Este mapeo es aproximadamente lineal por debajo de 1 kHz y logarítmico por encima (Figura 3.2).

El último paso es aplicar la transformada discreta del coseno a los componentes de los vectores espectrales de Mel calculados para cada frame. Estos componentes están altamente correlacionados, por lo que, para reducir el número de parámetros en el sistema y

decorrelacionar sus componentes, se aplica esta transformada a los vectores. El valor que se obtiene al aplicar esta transformada es el MFCC.

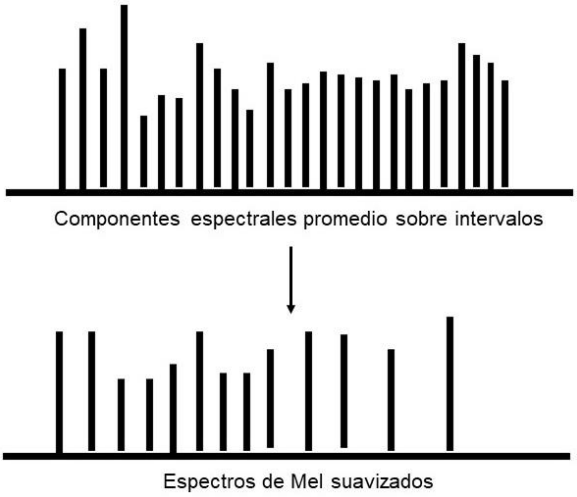


Figura 3.1 Ejemplo de suavizado de Mel

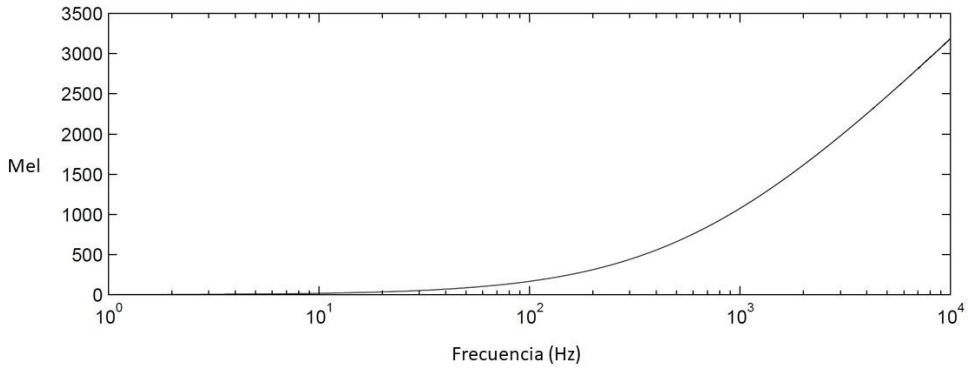


Figura 3.2 Ejemplo de suavizado de Mel

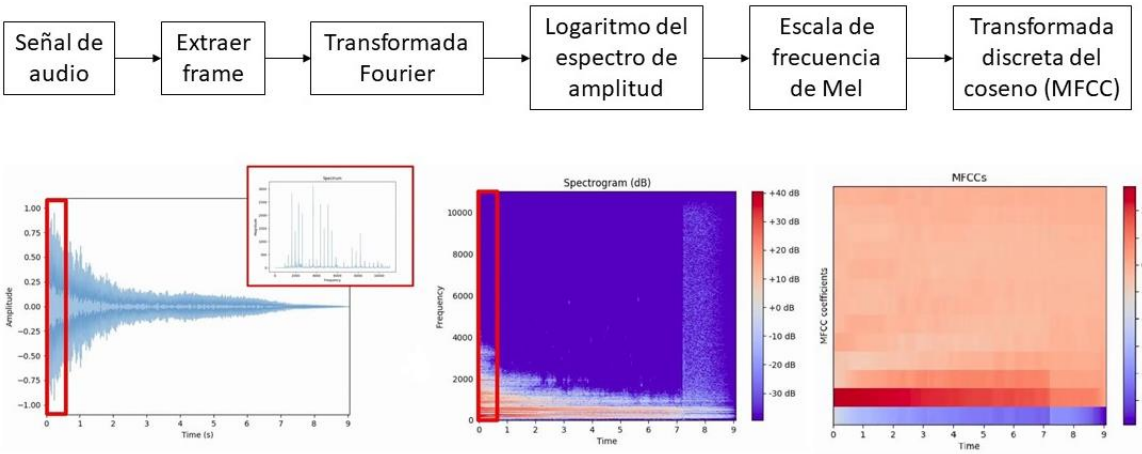


Figura 3.3 Ejemplo de extracción de un MFCC

3.2 Preparación del dataset

Para la elaboración del dataset, se ha tomado como referencia el realizado por George Tzanetakis durante los años 2000 y 2001 para su artículo [16]. Este dataset consta de 1.000 pistas de 30 segundos cada una, y se dividen en 10 géneros musicales, por lo que hay 100 pistas para cada género. En nuestro caso, hemos decidido clasificar a 4 vocalistas: Bruce Dickinson (vocalista de Iron Maiden), Freddie Mercury (vocalista de Queen), James Hetfield (vocalsita de Metallica) y Michael Jackson. El dataset consta de un total de 400 pistas de 30 segundos, 100 pistas para cada vocalista.

El motivo de que la duración de las pistas sea de 30 segundos es simplemente por temas de Copyright. Al tener las pistas esta duración y usarlas con fines académicos, se entra en el conocido "*Fair Use*" o "*Uso Justo*", que es una doctrina legal sobre el copyright que permite un uso limitado del material con derechos de autor sin la necesidad de requerir permiso a los titulares de tal derecho. Con vistas a publicar el presente proyecto en webs dedicadas a redes neuronales para que cualquiera pueda usarla libremente, decidí hacerlo de esta manera.

Para crear el dataset se ha utilizado el software de edición de audio Cubase Elements 8. El proceso es muy simple: se importa una pista de audio, se recorta para que dure 30 segundos y se concatenan fragmentos donde había voz para no dejar partes instrumentales. Una vez montada la pista, se exporta de nuevo en formato ".wav" (por ser este formato de fácil tratamiento en Python), a una frecuencia de muestreo de 48kHz y una profundidad de 32 bits (método de exportación por defecto de Cubase). Finalmente, esta pista se mueve al directorio del que Python extraerá los archivos de audio para procesarlos. Una vez creado el dataset, empezamos la programación para el pre-procesamiento.

3.3 Pre-procesamiento de audio

El primer paso antes de crear la red, tal y como se ha comentado en la introducción, es procesar el audio para poder así extraer los MFCCs de las pistas. Lo primero que vamos a hacer para que el proceso de extracción de MFCCs sea más rápido es, con una función diseñada por nosotros previa al código de creación y entrenamiento de la red, cambiar la frecuencia de muestreo de todas las pistas de 48kHz a 16kHz. De esta forma, tal y como se ha explicado en el apartado 3.1, se eliminan sonidos de altas frecuencias que no nos afectan para el entrenamiento de la red, consiguiendo así un procesamiento mucho más rápido al reducir considerablemente el peso de los archivos. Esto mismo se hará también con las pistas que utilicemos para probar la red una vez entrenada, ya que conviene que todos archivos que se usen tengan las mismas características. Esas pistas serán procesadas inicialmente por esta misma función, para después empezar con la extracción de MFCCs.

Para extraer los MFCCs, en nuestro caso se han utilizado 2048 frames, y se han obtenido 13 MFCCs de cada uno de ellos. La representación de extracción del MFCC se muestra en la Figura 3.3. Si observamos esta Figura, veremos claramente el proceso descrito anteriormente para extraer el MFCC: cogemos un intervalo de tiempo (frame), calculamos su transformada de

Fourier y lo representamos con colores en función de la frecuencia y la intensidad de la onda en ese instante de tiempo, y así sucesivamente con todos los frames. El aspecto del espectograma y el MFCC es similar, ya que se extraen de la misma manera, solo que, en el MFCC, después de haber llegado hasta el logaritmo del espectro de amplitud, se aplica el filtro de frecuencias de Mel para suavizar el espectro y enfatizar frecuencias perceptivamente significativas, y finalmente se aplica la transformada discreta del coseno para obtener los coeficientes. Así pues, en el MFCC se observan claramente las 13 filas (o MFCCs) que nosotros habíamos especificado que queríamos por cada frame, y, aunque no se aprecia tan bien, tenemos un total de 2048 columnas (frames). En definitiva, lo que tenemos es una matriz 13x2048, donde cada posición de la matriz tiene un valor, el MFCC. En la escala de colores se ve qué valores se han asignado a cada color.

Una vez extraídos los MFCCs, se almacenan en un vector con su etiqueta correspondiente (en nuestro caso la etiqueta es el vocalista al que pertenece ese MFCC), y con eso se entrenará la red. Esta aprenderá qué armónicos tienen un mayor peso, y, de esta forma, reconocerá qué vocalista es el que canta.

4. ARQUITECTURAS UTILIZADAS

A continuación, se van a mostrar y a explicar los tres tipos de arquitecturas utilizadas para la resolución del problema.

4.1 Red densamente conectada

La primera arquitectura con la que se ha tratado de resolver el problema ha sido una red neuronal feedforward densamente conectada, explicada en el apartado 2.2.3 de este documento. La estructura tiene la siguiente forma:

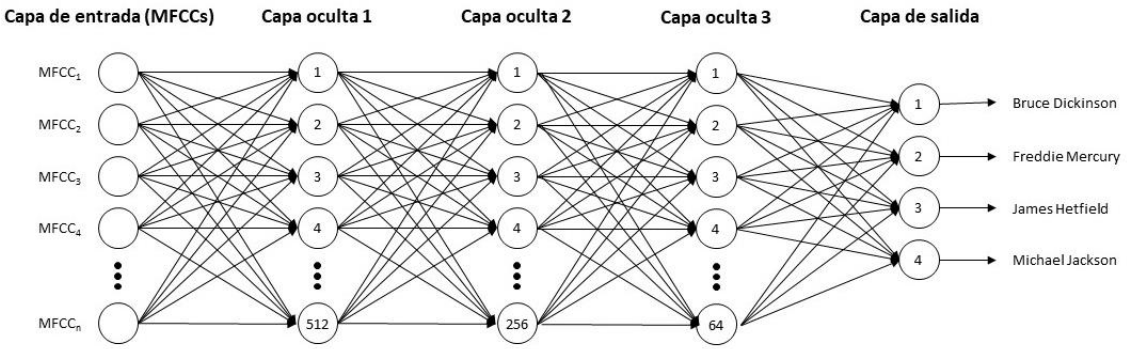


Figura 4.1 Representación de la red feedforward densamente conectada utilizada

La primera capa de entrada consta básicamente de todos los MFCCs que habíamos obtenido en el pre-procesamiento del audio, que, si recordamos, estaban almacenados es un vector. Por ello, lo primero que ha habido que hacer es utilizar la función Flatten() para convertir ese vector de MFCCs en una capa de entrada como la que muestra la figura. Seguidamente, hemos incluido tres capas ocultas de 512, 256 y 64 neuronas respectivamente, para finalmente, llegar a la capa de salida con las 4 neuronas que se corresponden con los 4 cantantes utilizados y que nos darán la probabilidad de que cada uno de los vocalistas esté cantando en la pista de audio que le introduzcamos. Respecto a los hiperparámetros, se ha usado un valor de 1000 para el batch size, un valor de 3000 para las épocas y un valor de 0,001 para el learning rate. Además, después de cada capa oculta se ha utilizado la función Dropout para reducir el sobreajuste, con un valor de 0,3. Por último, de las 100 pistas de audio que tenemos de cada cantante, se ha utilizado el 80% de ellas para entrenar la red, y el 20% restante para validar su porcentaje de acierto.

4.2 Red convolucional

La siguiente red utilizada ha sido una convolucional. En nuestro caso, se han usado filtros de 3x3 a lo largo de los MFCCs para el aprendizaje (Figura 4.2), y un total de 4 capas convolucionales de 16, 32, 64 y 128 mapas de características cada una, respectivamente. Después del último mapa, se ha añadido una capa de MaxPooling de 2x2. A continuación, se ha usado la función

Flatten() para aplanar la matriz y dejarla con la forma de una capa feedforward densamente conectada.

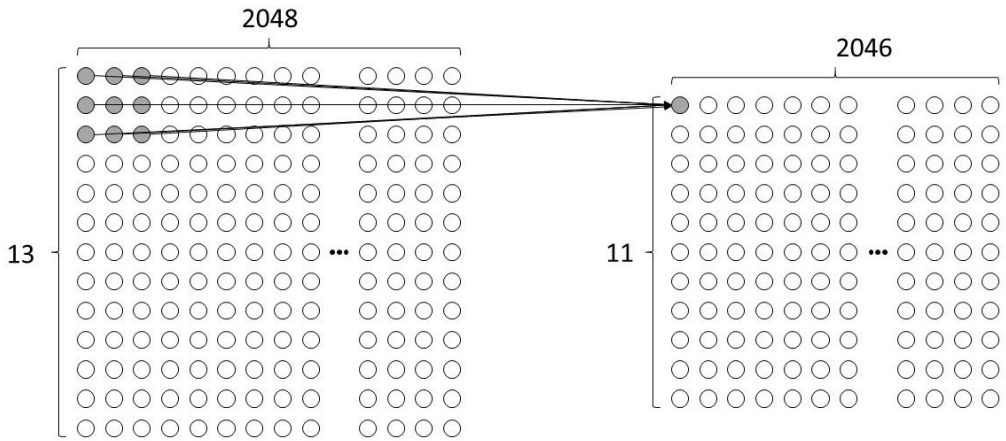


Figura 4.2 Representación del aprendizaje de la CNN utilizada

A continuación, se han utilizado 2 capas más feedforward densamente conectadas de 128 y 64 neuronas respectivamente, y finalmente la última capa de 4 neuronas para que nos den la probabilidad de cada vocalista. La representación de la red quedaría de la siguiente forma (no se han dibujado todas las conexiones en las capas densas por simplificar):

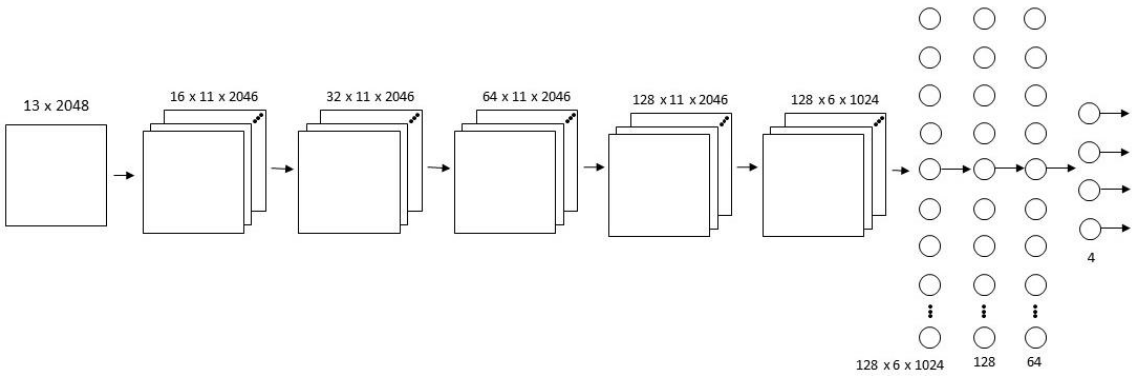


Figura 4.3 Representación de la arquitectura de la CNN utilizada

En los hiperparámetros, se ha usado un valor de 100 para el batch size, un valor de 50 para las épocas y un valor de 0,001 para el learning rate. Después de las capas convolucionales y la MaxPooling se ha utilizado la función Dropout para reducir el sobreajuste con un valor de 0,5. A continuación, se ha usado la función Flatten() para pasar de la última capa convolucional a una capa feedforward densamente conectada, y después de cada una de estas capas, se ha vuelto a utilizar la función Dropout con un valor esta vez de 0,5. Para el entrenamiento y validación de la red se ha usado el mismo porcentaje de pistas que antes.

4.3 Red recurrente

La última red utilizada ha sido una recurrente. La estructura de esta red se puede observar en la Figura 4.4, y consta de dos capas ocultas LSTM, cuatro capas feedforward densamente conectadas y finalmente la capa de salida.

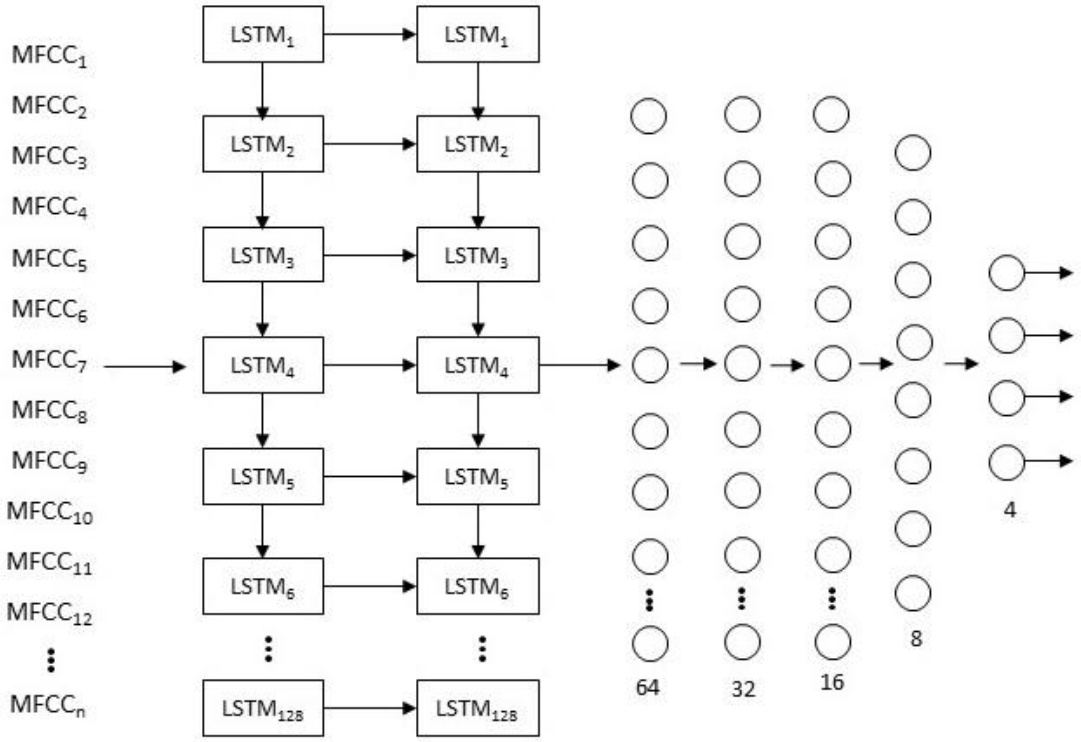


Figura 4.4 Representación de la arquitectura de la RNN utilizada

En cuanto a los hiperparámetros, se ha usado un valor de 100 para el batch size, un valor de 50 para las épocas y un valor de 0,001 para el learning rate. Las capas LSTM constan de 128 celdas cada una, y se ha utilizado la función Dropout en la última para reducir el sobreajuste con un valor de 0,3. Respecto a las capas feedforward, estas tienen 64, 32, 16 y 8 neuronas, tal y como se observa en la Figura anterior, y también se ha utilizado la función Dropout después de cada una de ellas para evitar el sobreajuste. Finalmente, al igual que en los 2 casos anteriores, tenemos la capa de salida con las 4 neuronas y se han usado los mismos porcentajes de pistas para el entrenamiento y validación.

5. DATOS EXPERIMENTALES

En el presente capítulo se van a mostrar y comentar los resultados que se han obtenido en el experimento con los distintos tipos de redes utilizadas, explicadas en el capítulo anterior. Las dos características más importantes a la hora de interpretar el resultado del entrenamiento de una red son la *tasa de acierto* (accuracy) y la función *de coste* (loss).

La tasa de acierto es una métrica para medir el rendimiento de un modelo y normalmente se expresa como un porcentaje, por lo que nos interesa que sea cuanto más alto mejor. La tasa de acierto es el número total de predicciones donde el valor pronosticado es igual al valor verdadero y generalmente se dibuja y monitorea durante la fase de entrenamiento, aunque el valor a menudo se asocia con la tasa de acierto general o final del modelo. La tasa de acierto es más fácil de interpretar que la pérdida.

La función de coste tiene en cuenta las probabilidades o la incertidumbre de una predicción en función de cuánto varía la predicción del valor real. Esto nos da una visión más matizada de qué tan bien está funcionando el modelo. A diferencia de la tasa de acierto, la pérdida no es un porcentaje, sino la suma de los errores cometidos para cada muestra en los conjuntos de entrenamiento o validación. La pérdida a menudo se usa en el proceso de entrenamiento para encontrar los mejores valores de parámetros para el modelo (por ejemplo, pesos en la red neuronal). Durante el proceso de capacitación, el objetivo es minimizar este valor. Para ello se utilizan el algoritmo del gradiente descendiente, que busca el punto de menor valor de la función de coste para optimizarla, y la técnica de retropropagación, que calcula el vector de gradientes dentro de la complejidad de la arquitectura de la red neuronal. No se va a entrar en detalle de este algoritmo y este método ya que no es necesario para entender el funcionamiento general de las redes neuronales, y su explicación necesitaría de apartados enteros dedicados exclusivamente a ellos. Además, para este proyecto se ha usado el algoritmo y el método disponible en las librerías de Python, con lo que es algo que no se ha diseñado ni modificado.

Para comprobar el acierto de nuestra red se han utilizado 16 pistas de audio, 4 de cada cantante. Una vez la red ya ha sido entrenada, se van a usar dichas para poner a prueba nuestra red y ver si es capaz de decirnos quién es el vocalista. Además, una de las pistas de Freddie Mercury no es Freddie, sino Marc Martel, un canadiense que se hizo famoso en Youtube por la increíble similitud entre su voz y la de Freddie, por lo que se empleó su pista para ver la tasa de acierto de la red.

5.1 Red densamente conectada

En la red feedforward densamente coenctada se han obtenido los siguientes resultados:

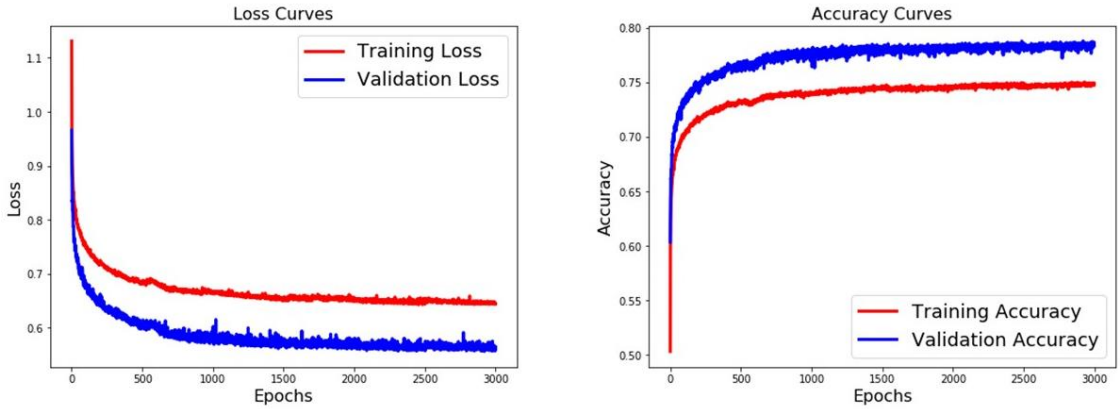


Figura 5.1 Curvas de pérdida y tasa de acierto de la red feedforward densamente conectada

Para los datos de entrenamiento tenemos una pérdida de 0.644 y una tasa de acierto del 74,9%, mientras que para los datos de validación tenemos una pérdida de 0,56 y una tasa de acierto del 78,35%. A pesar de ser una simple red feedforward densamente conectada, los resultados son aparentemente buenos, pero a la hora de comprobar el acierto de la red con las 16 pistas de audio vamos a ver que nuestra red no está muy segura de los resultados que nos da. Estos resultados han sido los siguientes y están expresados en tanto por 1:

	BRUCE DICKINSON	FREDDIE MERCURY	JAMES HETFIELD	MICHAEL JACKSON
BRUCE DICKINSON ***Bohemian rhapsody***	**0.528**	**0.191**	**0.023**	**0.258**
BRUCE DICKINSON ***I've got the fire***	**0.332**	**0.223**	**0.325**	**0.130**
BRUCE DICKINSON ***I can't see my feelings***	**0.552**	**0.090**	**0.169**	**0.190**
BRUCE DICKINSON ***No prayer for the dying***	**0.576**	**0.193**	**0.055**	**0.176**
FREDDIE MERCURY ***Don't try so hard***	**0.050**	**0.588**	**0.055**	**0.306**
MARC MARTEL ***Bohemian rhapsody***	**0.124**	**0.499**	**0.225**	**0.152**
FREDDIE MERCURY ***Ogre battle***	**0.153**	**0.656**	**0.066**	**0.125**
FREDDIE MERCURY ***White Man***	**0.094**	**0.651**	**0.168**	**0.086**
JAMES HETFIELD ***Escape***	**0.096**	**0.117**	**0.774**	**0.012**
JAMES HETFIELD ***The memory remains***	**0.089**	**0.154**	**0.710**	**0.048**
JAMES HETFIELD ***Hero of the day***	**0.121**	**0.223**	**0.468**	**0.188**

JAMES HETFIELD *Holier than Thou*	**0.015**	**0.067**	**0.895**	**0.023**
MICHAEL JACKSON *Bad*	**0.104**	**0.272**	**0.179**	**0.444**
MICHAEL JACKSON *Wanna be startin' somethin'*	**0.044**	**0.138**	**0.080**	**0.738**
MICHAEL JACKSON *Xscape*	**0.032**	**0.161**	**0.157**	**0.650**
MICHAEL JACKSON *You are there*	**0.259**	**0.262**	**0.026**	**0.453**

Tabla 5.1 Resultados experimentales de la red feedforward densamente conectada

Se puede ver como efectivamente, las probabilidades más altas corresponden al vocalista que está cantando, incluso en el caso de Marc Martel vemos que la probabilidad máxima es la de Freddie Mercury consiguiendo en cierto modo engañar a la red. Sin embargo, estas probabilidades apenas superan el 50% de seguridad salvo en los casos de *Escape* y *Holier Than Thou* de James Hetfield, por lo que, a pesar de que en todas pistas acierta en que la mayor probabilidad coincide con el vocalista correspondiente, no es una red muy adecuada.

5.2 Red convolucional

Los resultados con este tipo de red han sido notablemente mejores que los anteriores, aunque en las probabilidades veremos que, en algunos casos concretos, la red sigue sin estar muy segura, incluso en un caso se confunde de vocalista.

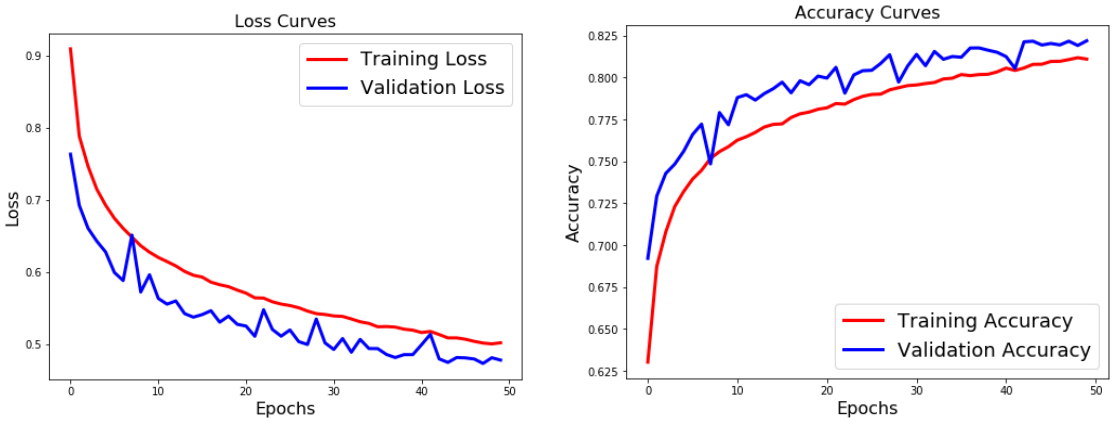


Figura 5.2 Curvas de pérdida y tasa de acierto de la red neuronal convolucional

En este caso, para los datos de entrenamiento tenemos una pérdida de 0,5 y una tasa de acierto de 81,1%. Y para los datos de validación, una pérdida de 0,48 y una tasa de acierto de 82,2%. Los resultados son mejores que los anteriores y estas han sido las probabilidades:

	BRUCE DICKINSON	FREDDIE MERCURY	JAMES HETFIELD	MICHAEL JACKSON
BRUCE DICKINSON *Bohemian rhapsody*	0.547	0.211	0.034	0.208
BRUCE DICKINSON *I've got the fire*	0.195	0.270	0.439	0.096
BRUCE DICKINSON *I can't see my feelings*	0.523	0.114	0.209	0.154
BRUCE DICKINSON *No prayer for the dying*	0.580	0.207	0.070	0.143
FREDDIE MERCURY Don't try so hard	0.042	0.621	0.070	0.267
MARC MARTEL *Bohemian rhapsody*	0.106	0.551	0.246	0.096
FREDDIE MERCURY *Ogre battle*	0.124	0.711	0.065	0.101
FREDDIE MERCURY *White Man*	0.056	0.744	0.145	0.056
JAMES HETFIELD *Escape*	0.029	0.081	0.886	0.004
JAMES HETFIELD *The memory remains*	0.071	0.176	0.735	0.018
JAMES HETFIELD *Hero of the day*	0.104	0.233	0.507	0.156
JAMES HETFIELD *Holier than Thou*	0.009	0.069	0.909	0.013
MICHAEL JACKSON *Bad*	0.057	0.258	0.161	0.525
MICHAEL JACKSON *Wanna be startin' somethin'*	0.026	0.100	0.052	0.822
MICHAEL JACKSON *Xscape*	0.028	0.171	0.133	0.667
MICHAEL JACKSON *You are there*	0.278	0.279	0.036	0.406

Tabla 5.2 Resultados experimentales de la CNN

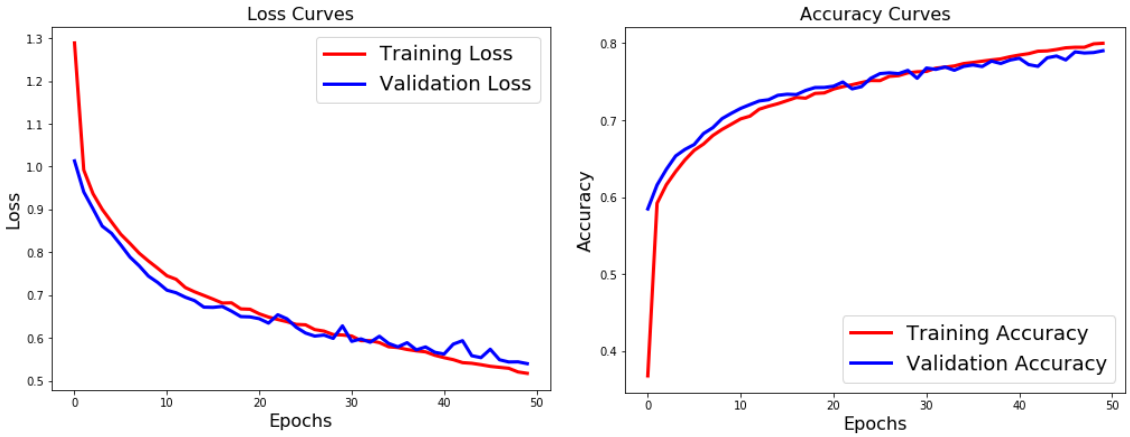
Como se puede apreciar, los resultados son algo mejores que los de la red feedforward densamente conectada. Esta vez ya llegamos a probabilidades del 60% y 70% e incluso del 80% y 90% en algún caso. Aun así, hay veces que la red sigue sin estar demasiado segura de quién está cantando.

Los casos de Bruce Dickinson son los más confusos para la red en comparación a los demás vocalistas, incluso llega a equivocarse en el caso de la pista *I've got the fire*, que piensa que es James Hetfield en vez de Bruce, pero a pesar de equivocarse, tampoco está muy segura de que el vocalista que está cantando sea James, ya que la probabilidad que nos da es un 43,9%. Por tanto, se puede concluir que la red neuronal convolucional es mejor que la feedforward densamente conectada para la clasificación de vocalistas, ya que nos da resultados más fiables.

5.3 Red recurrente

Por último, los resultados de esta red han sido muy similares a los de la convolucional, han estado levemente por debajo en cuanto a la tasa de acierto. Aun así, veremos en las gráficas y la tabla que las probabilidades son en general parecidas:

Figura 5.3 Curvas de pérdida y tasa de acierto de la red neuronal recurrente



En esta última red, tenemos para los datos de entrenamiento una pérdida de 0,517 y una tasa de acierto del 80%, y para los datos de validación tenemos una pérdida de 0,54 y una tasa de acierto del 79%. A continuación, se muestran las probabilidades obtenidas:

	BRUCE DICKINSON	FREDDIE MERCURY	JAMES HETFIELD	MICHAEL JACKSON
BRUCE DICKINSON *Bohemian rhapsody*	**0.492**	**0.229**	**0.034**	**0.245**
BRUCE DICKINSON *I've got the fire*	**0.136**	**0.240**	**0.445**	**0.179**
BRUCE DICKINSON *I can't see my feelings*	**0.527**	**0.090**	**0.146**	**0.237**
BRUCE DICKINSON *No prayer for the dying*	**0.513**	**0.214**	**0.068**	**0.205**
FREDDIE MERCURY Don't try so hard	**0.028**	**0.656**	**0.064**	**0.252**
MARC MARTEL *Bohemian rhapsody*	**0.099**	**0.534**	**0.283**	**0.084**
FREDDIE MERCURY *Ogre battle*	**0.102**	**0.720**	**0.059**	**0.119**
FREDDIE MERCURY *White Man*	**0.051**	**0.705**	**0.159**	**0.085**
JAMES HETFIELD *Escape*	**0.065**	**0.091**	**0.833**	**0.012**
JAMES HETFIELD *The memory remains*	**0.039**	**0.088**	**0.834**	**0.038**
JAMES HETFIELD *Hero of the day*	**0.089**	**0.222**	**0.499**	**0.190**

JAMES HETFIELD ***Holier than Thou***	**0.015**	**0.065**	**0.890**	**0.030**
MICHAEL JACKSON ***Bad***	**0.078**	**0.284**	**0.199**	**0.439**
MICHAEL JACKSON ***Wanna be startin' somethin'***	**0.058**	**0.155**	**0.078**	**0.710**
MICHAEL JACKSON ***Xscape***	**0.031**	**0.151**	**0.170**	**0.648**
MICHAEL JACKSON ***You are there***	**0.234**	**0.297**	**0.020**	**0.450**

Tabla 5.3 Resultados experimentales de la RNN

Si comparamos estos resultados con los de la red convolucional, vemos que efectivamente las probabilidades son muy similares e incluso iguales en algunos casos. La tasa de acierto está un poco más acertada en las pistas y en otros menos, pero a la hora de decidir qué vocalista es el que canta, coincide en todas con la red anterior, y además se equivoca en la misma pista también. Bruce Dickinson sigue siendo el vocalista más dudoso y James Hetfield el más acertado.

6. CONCLUSIONES

En este TFG se ha desarrollado y evaluado un algoritmo capaz de reconocer vocalistas en fragmentos de audio. En un principio el problema nos pareció complicado, pero tras una intensa búsqueda bibliográfica y un extenso proceso de diseño de las redes, se ha logrado alcanzar unas tasas de aciertos muy razonables.

Trabajar con audio y redes neuronales es un proceso lento y costoso debido al peso de los archivos de audio y el tiempo requerido para su procesamiento. Esto puede ser una limitación a la hora de realizar un trabajo como este, ya que el tiempo del que se dispone es limitado y hay que intentar obtener los mejores resultados posibles en el tiempo disponible. Aun así, los resultados conseguidos han sido bastante buenos. Se puede afirmar que este problema de reconocimiento se puede abordar con garantías de obtener buenos resultados. Y en concreto, con unas redes más elaboradas y complejas y un dataset más amplio, se obtendrían resultados mejores.

En lo referente a las redes neuronales, la conclusión principal es que las redes convolucionales y recurrentes tienen unos resultados notablemente mejores que las feedforward densamente conectadas debido a la forma de aprendizaje de estas. Teóricamente, las CNN son ideales para el procesamiento de imágenes y vídeo mientras que las RNN son ideales para el análisis de texto en voz, así que el hecho de haber conseguido resultados experimentales similares seguramente se deba a las arquitecturas escogidas. También se pueden hacer arquitecturas mixtas, no es necesario que sean exclusivamente de un tipo.

Algunas posibles aplicaciones de este proyecto en la vida cotidiana podrían ser:

- **Identificación de personas en casos de criminalidad**. Si en las grabaciones no se aprecia bien la imagen y no se distingue con claridad a la persona que comete el crimen, pero hay grabación de audio, se podría usar este sistema para su identificación.
- **Sistemas de seguridad**. Al igual que existen sistemas de identificación ocular mediante reconocimiento de iris, podría usarse un sistema de identificación en función del timbre de voz.
- **Evitar engaños a sistemas informáticos**. Si se consiguiera un sistema de reconocimiento de voz suficientemente robusto, se podrían evitar engaños o estafas con grabaciones o imitadores de voz profesionales, ya que cada voz es única y no hay dos iguales.

Seguramente haya más aplicaciones en las que se puedan utilizar sistemas de reconocimiento por voz, y es por esto que resultaría de interés continuar investigando en su desarrollo. Es muy probable que algún día convivamos con robots de una forma normal y cotidiana, y en tal caso, estos sistemas serán imprescindibles para que los robots solo tengan un “dueño”.

BIBLIOGRAFÍA

[1] De Luna Ortega, C. A., Martínez Romo, J. C. y Mora González, M. (2006). Reconocimiento de Voz con Redes Neuronales, DTW y Modelos Ocultos de Markov. *Conciencia Tecnológica*, vol. 32, p. 1.

[2] Julián, G. (30 de Diciembre, 2014). Las redes neuronales: qué son y por qué están volviendo. Recuperado de https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo

[3] Difference between Inductive and Deductive reasoning. (s. f.). Recuperado 24 de junio de 2020, de https://www.javatpoint.com/difference-between-inductive-and-deductive-reasoning

[4] Khamparia, A., Gupta, D., Gia Nhu, N., Khanna, A., Pandey, B. y Tiwari, P. (2019). Sound Classification Using Convolutional Neural Network and Tensor Deep Stacking Network. *IEEE Access, pp. 1-2. doi*: 0.1109/ACCESS.2018.2888882.

[5] Martínez Mascorro, G. A. y Aguilar Torres, G. (2013). Reconocimiento de voz basado en MFCC, SBC y Espectrogramas. *Ingenius*, vol. 10, pp. 12-20.

[6] Aggarwal, C. (2018). *Neural Networks and Deep Learning*, Yorktown, NY, USA: Springer.

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

[8] Wu, H. y Prasad, S. (2017). Convolutional Recurrent Neural Networks for Hyperspectral Data Classification. *Remote Sensing*, vol. 9, pp. 5-6. doi: 10.3390/rs9030298.

[9] Botvinick, M. y Plaut, D. (2006). Short-Term Memory for Serial Order: A Recurrent Neural Network Model. *Psychological Review*, vol. 9 nº 2, pp. 201-233. doi: 10.1037/0033-295X.113.2.201.

[10] ›taudemeyer, R. y Rothstein Morris, E. (2019). Understanding LSTM - A tutorial into Long Short-Term Memory.

[11] Paneru, E. (27 de Agosto, 2015). Understanding LSTM Networks. Recuperado de http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[12] Logan, Beth. (2000). Mel Frequency Cepstral Coefficients for Music Modeling. Proc. 1st Int. Symposium Music Information Retrieval.

[13] Carney, J. y Cunningham, P. (1999). The Epoch Interpretation of Learning. Department of Computer Science, University of Dublin, Trinity College, Ireland.

[14] Leonel, J. (2019, Abril 6). Hyperparameters in Machine/Deep Learning. Recuperado de https://medium.com/@jorgesleonel/hyperparameters-in-machine-deep-learning-ca69ad10b981

[15] Rodríguez, V. (2018, Noviembre 9). Dropout y Batch Normalization. Recuperado de https://vincentblog.xyz/posts/dropout-y-batch-normalization

[16] Tzanetakis, G. y Cook, P. (2002). Musical Genre Classification of Audio Signals. *IEEE Transactions on Speech and Audio Processing*. 10. 293 - 302. 10.1109/TSA.2002.800560.