Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

RECONOCIMIENTO DE IMÁGENES DE AVES CON REDES NEURONALES PROFUNDAS

Autor: Alonso Zurera Martínez-Acitores
Tutor: Ana María González Marcos

Junio 2018

RECONOCIMIENTO DE IMÁGENES DE AVES CON REDES NEURONALES PROFUNDAS

Autor: Alonso Zurera Martínez-Acitores
Tutor: Ana María González Marcos

Grupo de la EPS: Grupo de Aprendizaje Automático (GAA)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2018

Resumen

Resumen

El mundo de la inteligencia artificial se esta desarrollando a pasos agigantados, en los últimos años hemos conseguido hacer que las maquinas aprendan de la forma en la que lo haría un ser humano. Uno de los grandes retos de la inteligencia artificial es conseguir que el ordenador consiga adquirir una visión parecida a la que tenemos los humanos, y por consiguiente que consiga clasificar las imágenes por los objetos que presentan. En este trabajo hemos realizado un ejercicio básico de esta problemática y sirve como guía para aquellos que quieran comenzar en el mundo de la visión artificial.

Primero hemos realizado una parte teórica para comenzar a adentrarnos en el mundo de la inteligencia artificial y de la visión artificial. Hemos comenzado explicando conceptos básicos sobre las imágenes en el mundo digital, es decir como interpreta la imagen un ordenador. Después introducimos el tema de la inteligencia de la inteligencia artificial con el apartado de aprendizaje automático, en el que explicamos os diferentes tipos de aprendizaje, algunos conceptos básicos importantes y presentamos el tema del aprendizaje profundo y las redes neuronales. En el siguiente apartado explicamos en detalle que es una red neuronal y las características básicas de la misma. Y en el último apartado teórico explicamos las redes neuronales convolucionales con las diferentes capas de esta.

En segundo lugar, se aplicarán los conceptos teóricos expuestos con anterioridad para la creación de una red neuronal convolucional capaz de distinguir entre imágenes sean de aves de las que no. A continuación, se analizarán los datos obtenidos de la red.

Por ultimo se llegará a una conclusión del trabajo y como poder mejorar en el futuro el trabajo realizado en este.

Palabras Clave

Aprendizaje Automatico, Aprendizaje Profundo, Redes Neuronales, Redes Neuronales Convolucionales, Clasificación de Imágenes

Abstract

The world of artificial intelligence is developing by leaps and bounds, in recent years it has managed to make machines learn in a similar way to how humans learn. One of the great challenges of artificial intelligence is to get the computer to obtein a vision like the humans have, and therefore to can classify images by the objects that present it. In this work we have done a basic exercise of this problem and it serves as a guide for those who want to start in the world of artificial vision.

First we have done a theoretical part to begin to enter the world of artificial intelligence and artificial vision. We have started by explaining basic concepts about images in the digital world, we explain how a computer inerpret the image. Then he introduces the topic of artificial intelligence with the machine learning section, in which we explain the different learning methodologies, some important basic concepts and we present the topic of deep learning and neural networks. In the next section we explain in detail what it is a neural network and the basic characteristics of it. And in the last theoretical section, we explain the convolutional neural networks with its different layers.

Secondly, we will apply the theoretical concepts exposed in this work to create a convolutional neural network capable of distinguishing between images that show birds and those that do not. Next, the data obtained from the network will be analyzed.

Finally, we will come to a conclusion and how to improve the future work done in this

Key words

Machine Learning, Deep Learning, Neural Networks, Convolutional Networks, Image Clasification

Agradecimientos

Quiero agradecer a mi tutora Ana María González por haber sido mi guía y mi estímulo durante la redacción de mi trabajo de fin de grado.

Índice general

Índice de Figuras

Índice de Tablas

1

Introducción y retos

1.1. Motivación del proyecto

En este proyecto se va a realizar un estudio de la metodología de aprendizaje profundo para aplicarlo al reconocimiento automático de imágenes. El objetivo de la visión artificial se centra en que un ordenador comprenda y entienda una imagen del mundo real. Nuestro propósito principal del proyecto es que el ordenador comprenda las imágenes que le proporcionemos y sea capaz de realizar una clasificación de las mismas. El alcance puede ser muy amplio, pero vamos a acotar el problema a diferenciar, en primera instancia, aves de otros objetos, usando para ello repositorios de imágenes, como por ejemplo el de la Universidad de Caltech[1]. [9]

Para el ser humano la tarea de identificación o de comprensión de una imagen es realmente fácil, pero para un ordenador es una tarea difícil y compleja. En nuestra propuesta de trabajo, el simple hecho de que existan una gran multitud de aves diferentes no hace otra cosa salvo dificultar el problema. Además, otros factores ambientales como la iluminación de la imagen, el perfil del ave (si está en vuelo o posada) o de si el ave aparece entera o solo una parte porque está parcialmente oculta, incrementan la complejidad del problema. Algunos de estos problemas los solucionaremos con una entrada amplia de datos a nuestro programa, es decir un conjunto de imágenes robusto potenciará mejores clasificadores. En definitiva, vamos a aprender de la información contenida en los datos, que en nuestro caso son imágenes, y cuanta mayor variabilidad de imágenes exista más certeras serán nuestras predicciones.

Otra problemática adicional es la dimensionalidad de la base de datos. Sin contar con la cantidad de imágenes que vamos a tener que trabajar, una sola imagen *per se* ya tiene un gran número de atributos, cada pixel es un atributo, y dependiendo de la resolución de la imagen la dimensionalidad aumentará en concordancia.

Para resolver el reto que nos hemos planteado vamos a recurrir a metodologías desarrolladas dentro del campo del aprendizaje profundo. Nos centraremos en el uso de redes neuronales convolucionales junto con el empleo de la técnica de *pooling* que reduce la cantidad de parámetros al extraer las características más comunes de las imágenes mostradas.

Desde el punto de vista práctico, la aplicación a desarrollar se implementará en el lenguaje de programación de Python aprovechando la potencia de las bibliotecas desarrolladas para trabajar

[1] Repositorio exlusivo de imagenes de aves.

con técnicas de aprendizaje automático, como por ejemplo *scikit-learn* y *keras*.

2

Conceptos básicos sobre imágenes digitales

2.1. Introducción

Para representar una imagen en un ordenador tenemos que ser capaces de representarlo de manera matemática. Para ello tenemos que dividir la imagen completa en un número de cuadrados unitarios que representan valores de intensidad. Esta unidad es llamada pixel y la imagen es interpretado por el ordenador como una matriz numérica de dos dimensiones (en adelante, 2D).

2.2. Sistema de coordenadas

Para saber que posición de la imagen real corresponde a que posición de la imagen digital, es decir a cada pixel, vamos a utilizar un sistema de coordenadas. Nuestro sistema de coordenadas comenzará a numerarse por el número cero, esto es debido a que en múltiples lenguajes de programación, entre ellos Python, el sistema de indexado de *arrays* empieza por cero. Al contrario que las convenciones matemáticas, nuestro sistema de coordenadas se numerará de arriba a abajo. Teniendo la coordenada (0,0) en la parte superior izquierda. [1, pág. 9]

2.3. Tamaño de imagen y resolución

El tamaño de la imagen viene determinado por el número de columnas y por el número de filas de la matriz de la imagen. Por ejemplo, en la Figura 2.1 el tamaño de la imagen sera de M columnas por N filas. La resolución de una imagen es el grado de detalle o calidad de la imagen, la medida más usadas es *pixeles por pulgada* (ppp)[1]. La resolución es importante en las imágenes que tengan curvas o círculos, debido a que al estar siendo representados por cuadrados a más cuadrados mejor se adaptan a la forma curva. En nuestro proyecto es importante utilizar imágenes con un mínimo de resolución para que la red neuronal distinga bien los detalles de la imagen, porque en definitiva tener mayor resolución conlleva tener una imagen con más detalle

[1] En inglés *dots per inch* (dpi).

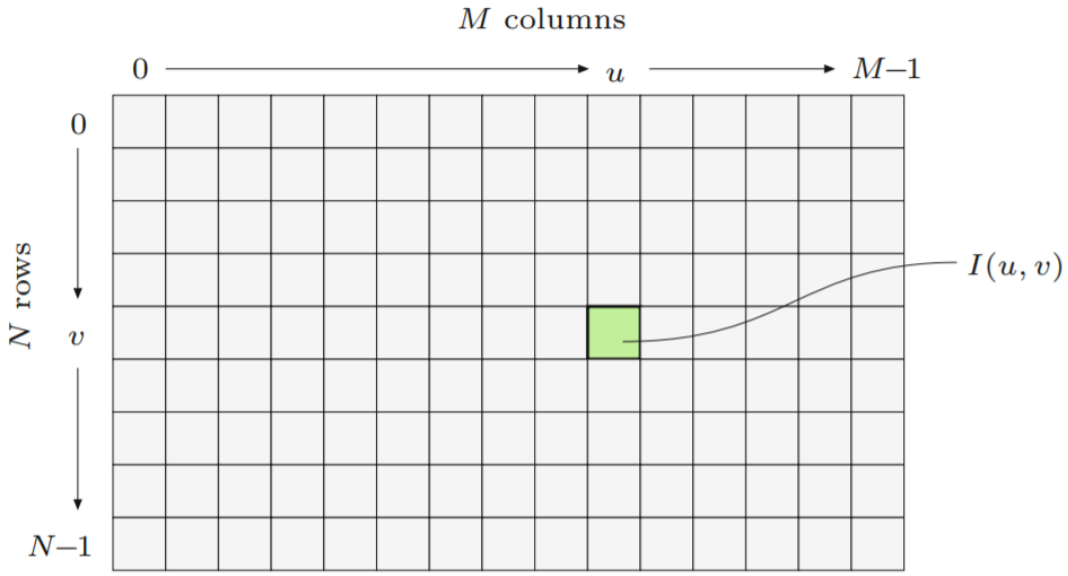


Figura 2.1: Concepto de coordenadas de una imagen. Como podemos observar se comienza a numerar a partir del cero. El pixel *I(u,v)* representa un pixel donde se cruzan la columna u con la fila v. El tamaño de la imagen es de M columnas y de N filas, con un total de (MxN) pixeles. [1, pág 10]

o calidad visual. La figura 2.2 muestra el concepto de resolución mostrando la misma imagen en diferentes resoluciones.

2.4. Valores de los Pixeles

La información de un elemento de la imagen depende del tipo de dato que la representa. El valor de los pixeles está representado por 2^k diferentes valores, donde k significa el *bit depth* de la imagen, que representa el número de valores diferentes que puede tomar cada pixel por sí solo. El valor de k suele ser igual a 8 por lo que cada pixel puede tomar 256 valores diferentes. Las propiedades de algunas imágenes están recogidas en la Tabla 2.1

Imágenes en escala de grises			
Canal	Bits/Pixel	Rango	Uso
1	1	[0,1]	Imágenes Binarias: documentos, gráficos, fax ...
1	8	[0,255]	Universal: fotos, escaneo ...
1	12	[0,4095]	Gran calidad: fotos, escaneo ...
Imágenes de color			
Canales	Bits/Pixel	Rango	Uso
3	24	$[0,255]^3$	RGB, universal: fotos, escaneo ...
3	36	$[0,4095]^3$	RGB, gran calidad: fotos, escaneo ...

Tabla 2.1: *Bit depth* en los diferentes tipos de imágenes y sus usos. [1, pág. 10]

Figura 2.2: Concepto de resolución de una imagen. De mayor la foto **a**, a menor resolución de imagen la foto **d**. [2, pág 7]

2.4.1. Imágenes en escala de grises

Imágenes Binarias

Consisten en un solo canal, y además, los valores que pueden tomar los pixeles son dos, blanco o negro. Solo usan un bit por pixel (0/1).
Este tipo de imágenes se usan, principalmente, para gráficos, documentos, No serán objeto de estudio estas imágenes en nuestro proyecto, pues no tienen la resolución que se precisa.

También pueden ser imágenes que representan la iluminación de la imagen con sus pixeles. Suelen utilizar un bit depht de 8, en este caso el 0 significaría el mínimo brillo (el color negro) y el 255 el máximo brillo (el color blanco). El *bit depth* normalmente se refiere a la cantidad de bits para representar un color, no la cantidad de bits necesarios para un pixel de color entero.

2.4.2. Imágenes de color

La mayoría de las imágenes a color usan los colores primarios, que son el rojo, verde y azul.[2] Cada color primario está representado por un canal y se necesitan ocho bits por canal para representar un color. Por lo tanto para codificar cada pixel se requerirá de $3 * 8 = 24$ bits. El rango de cada componente de color de forma individual es de [0,255]. Este es el tipo de imágenes que usaremos en nuestro proyecto. [1, pág. 9-11]

[2]En inglés *Red*, *Green* y *Blue*, es decir RGB.

3

Aprendizaje automático

3.1. Introducción

Actualmente estamos inmersos en la era del *big data*. Ejemplos de ello es que hace ya más de 10 años que *Google* alcanzo la marca de un billón[1] de URLs únicas en la red, o bien en la plataforma de *YouTube* se suben más de 400 horas de vídeo por minuto.

Toda esta cantidad de información hace que necesitemos algoritmos capaces de analizar toda esa cantidad de datos, eso es lo que las técnicas de aprendizaje automático proporcionan. Con estas técnicas se consigue localizar automáticamente patrones en los datos que permitan predecir el futuro o dar apoyo a la toma de algunas decisiones que se encuentran bajo incertidumbre. En definitiva, los algoritmos tienen la capacidad de aprender. Con el aprendizaje automático conseguimos dar valor a los datos.

We are drowning in information and starving for knowledge. —John Naisbitt.

3.1.1. Metodologías de aprendizaje

El aprendizaje suele estar divido en dos tipos principalmente:

- Aprendizaje supervisado[2], que en términos generales, son algoritmos de aprendizaje que aprender a asociar unos datos de entrada x con algunos datos de salida y, dado un conjunto de ejemplos etiquetados $\mathcal{D} = \{(x_i, y_i)\}^N{}_{i=1}$. Donde $\mathcal{D}$ se llama al conjunto de entrenamiento y N es el número de ejemplos de entrenamiento. En muchos casos los datos de salida y no son fáciles de conseguir y deben ser proporcionado por un ser humano "supervisor". El escenario más sencillo, consta de un conjunto de datos de entrada x_i formados por un vector de números de $\mathcal{D}-$dimensiones, un ejemplo seria un vector de dos números que representan la altura y el peso de una persona. Estos datos de entrada se llaman características o atributos. Cabe destacar, que x_i puede ser una estructura mas complicada, como una imagen, una frase, una gráfica, etc..

[1] Conviene aclarar que en España es igual a 1.000.000.000.000
[2] En ingles se llama *supervised learning*

De forma parecida los datos de salida o variable de respuesta puede ser cualquier estructura también, aunque lo normal es que sea: (1) una variable categórica o numeral que venga dada a partir de un conjunto finito, $y_i \in \{1..C\}$, en el ejemplo anterior sería hombre o mujer, o (2) y_i sea un valor escalar real, como el nivel de ingresos o (3) un vector d-dimensional. Cuando la variable y_i es un valor categórico, el problema se conoce como clasificación, y cuando la y_i representa valores reales, el problema se conoce como regresión.

- Aprendizaje no supervisado[3],en este caso solo tenemos los datos de entrada, $\mathcal{D} = x_i$ donde $i = 1 \dots N$, y el objetivo de este tipo de aprendizaje es encontrar una "estructura interesante". Este tipo de problemas se encuentran menos definidos, debido a que no nos dicen qué tipo de patrones tenemos que buscar, y no hay una forma obvia de saber el error métrico.

  La distinción entre algoritmos de aprendizaje supervisado o no supervisado no está formalmente definido porque no hay un test objetivo que distinga si un valor es un atributo o la variable respuesta proporcionada por un supervisor.

  Este aprendizaje es discutiblemente el aprendizaje más típico en seres humanos y animales. También se puede usar más ampliamente que el aprendizaje supervisado, debido a que no requiere de un supervisor que etiquete manualmente todos los datos.

Hay un tercer tipo de aprendizaje conocido como aprendizaje por refuerzo, pero es menos usado que los otros dos. [10, pág 1-10]

3.2. Conceptos básicos

En este apartado vamos ver algunos conceptos relacionados con el aprendizaje automático y con el aprendizaje profundo.

3.2.1. *Underfitting* y *overfitting*

El objetivo principal del aprendizaje automático es que se obtenga un buen rendimiento con datos de entrada que no "hayan sido vistos", es decir unos datos de entrada que sean nuevos. Esta habilidad de actuar bien en datos de entrada que no se han observado se llama generalización.

Normalmente, cuando vamos a entrenar un modelo de aprendizaje automático, dividimos el conjunto de ejemplos en dos partes. Una parte va a ser con la que vamos a entrenar el modelo, esta parte la llamaremos conjunto de entrenamiento. La otra parte la emplearemos para comprobar si nuestros modelos tienen la habilidad de generalización, este conjunto de datos no han sido vistos en la fase de entrenamiento, y lo denominaremos conjunto de prueba o conjunto de test. El objetivo de la metodología de aprendizaje automático es optimizar el error del conjunto de prueba, también llamado error de generalización.

Imagen cómo está dividido el conjunto de datos.

Factores decisivos para determinar si un algoritmo de aprendizaje automático va a generalizar bien son:

1. Conseguir un error de entrenamiento pequeño sin llegar al *overfitting*.
2. Conseguir que la variación entre el error de entrenamiento y el de test sea pequeño.

[3]En ingles se llama *unsupervised learning*

Estos factores corresponden con los dos desafios centrales en aprendizaje automático: *Underfitting* y *Overfitting*.

- ***Underfitting*** ocurre cuando el modelo no es capaz de conseguir un error de entrenamiento lo suficientemente pequeño y por consiguiente el modelo tiene una baja capacidad (o es un modelo de baja complejidad), no tiene el suficiente número de parámetros para poder aprender el problema al que se enfrenta.
- ***Overfitting*** ocurre cuando para el mismo modelo la variación entre el error de entrenamiento y el error de test es demasiado grande, el modelo tiene una capacidad alta. Es decir, el modelo está sobreajustado, tiene tantos parámetros que es capaz de memorizar propiedades o atributos del conjunto de entrenamiento y después no es capaz de generalizar en el conjunto de test.

Podemos controlar estos factores alterando la complejidad del modelo. Esto informalmente se refiere, a la elección adecuada del algoritmo y número de parámetros a optimizar.

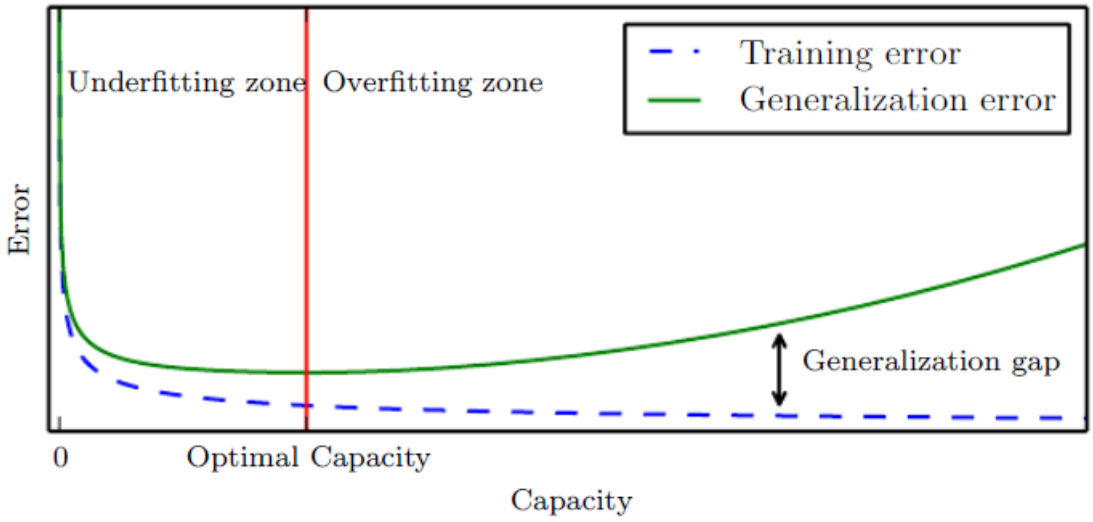


Figura 3.1: El gráfico representa la relación que existe entre el error y la complejidad del modelo. A la izquierda del gráfico, podemos observar que se produce *underfitting* , tanto el error de entrenamiento como el de generalización son altos. Cuando incrementamos la complejidad del modelo, ambos errores disminuyen hasta un determinado punto en el que el error de entrenamiento sigue disminuyendo pero el error de test comienza a aumentar. La línea roja del gráfico muestra la complejidad óptima del modelo, a partir de este punto, comienza a haber *overfitting*. [3, pág 113]

Si se diera el caso de que el modelo tenga la capacidad/complejidad óptima y aun así la variación entre el error de entrenamiento y el error de test es grande implica que el número de casos en el conjunto de entrenamiento es pequeño, de nuevo se produce *overfitting*. La solución para reducir la variación entre ambos errores es reunir más ejemplos en el conjunto de entrenamiento (aumentar el conjunto de entrenamiento). [3, pág 110-116]

3.2.2. Regularización

Para solucionar el problema de *overfitting* la técnica de regularización [3, pág 118-120] permite evitar el *overfitting* penalizando la función de coste con algún término de regularización,

esto se traduce en reducir el error de generalización al impedir que los valores de los parámetros de los modelos puedan crecer tanto como deseen de forma que se adaptan al conjunto de entrenamiento. En definitiva, se trata de suavizar los valores de los parámetros de los modelos elegidos.

3.2.3. La maldición de la dimensionalidad

Los problemas de aprendizaje automático se vuelven extremadamente complicados cuando aumenta el numero de dimensiones en los datos del problema. Esto es debido a la gran variedad de configuraciones que pueden tomar las variables y el aumento del número de instancias que es necesario para realizar una representación fidedigna de la realidad muestral.

Figura 3.2: Representación de la maldición de la dimensionalidad. A medida que el número de dimensiones de los datos incrementa (de derecha a izquierda), el número de configuraciones incrementará exponencialmente. [3, pág 155]

Como muestra la imagen 3.2 la variedad de configuraciones que pueden tomar las variables crece exponencialmente a medida que aumentan las dimensiones. Este es uno de los retos con los que tendra que lidiar los algoritmos de aprendizaje profundo. [3, pág 155]

3.3. Aprendizaje Profundo

El aprendizaje profundo es un tipo de aprendizaje automático que simula enseñar a los ordenadores a aprender como lo hacen los seres humanos, a través de la experiencia. La clave de este tipo de aprendizaje reside en el método conocido como el descenso de gradiente estocástico que puede entrenar eficazmente redes neuronales.

El aprendizaje profundo [3, pág 154] proporciona una importante estructura para trabajar con algoritmos de aprendizaje supervisado. Añadiendo más capas y más unidades por capa, una red neuronal puede representar funciones que incrementan la complejidad. Gracias a estas mejoras, la visión por ordenador es una de las áreas donde el aprendizaje profundo proporciona una mejora considerable respecto a los algoritmos tradicionales.

3.3.1. Objetivos del aprendizaje profundo

Las redes neuronales simples suelen trabajar con una estructura de dos niveles de arquitectura, este tipo de estructura se encuentra representada a la izquierda de la figura 3.4. Este tipo

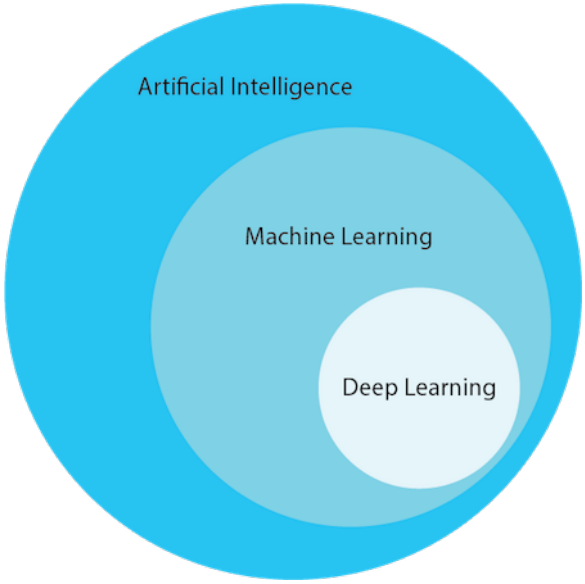


Figura 3.3: Diagrama de Venn que muestra como el aprendizaje profundo es un tipo de aprendizaje automático. [4]

de arquitectura no tiene demasiado éxito resolviendo los problemas centrales de la inteligencia artificial, como el reconocimiento de objetos en imágenes por ejemplo.

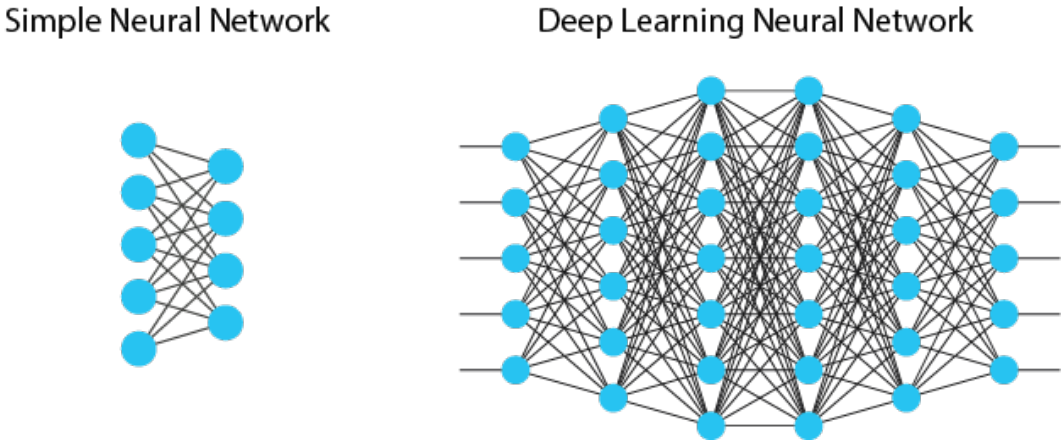


Figura 3.4: A la izquierda tenemos la representación de una red neuronal simple, que utiliza una estructura de dos niveles. A la derecha tenemos una representación de una red neuronal de arquitectura profunda. [4]

El desarrollo del aprendizaje profundo está motivado por el fracaso de los algoritmos antecesores a la hora de generalizar problemas que trabajan con grandes dimensiones de datos. En el siguiente capítulo veremos cómo trabajan las redes neuronales y en concreto las redes neuronales convolucionales.

4

Redes Neuronales

4.1. Introducción

Las redes neuronales artificiales nacen de la inspiración de la capacidad de aprender de los seres humanos. El sistema biológico de los seres humanos se compone de células que a su vez se compone de neuronas.

4.1.1. La neurona biológica

La neurona biológica [11, pág 1-7] son células que presentan una forma especial, se compone de tres partes principales:

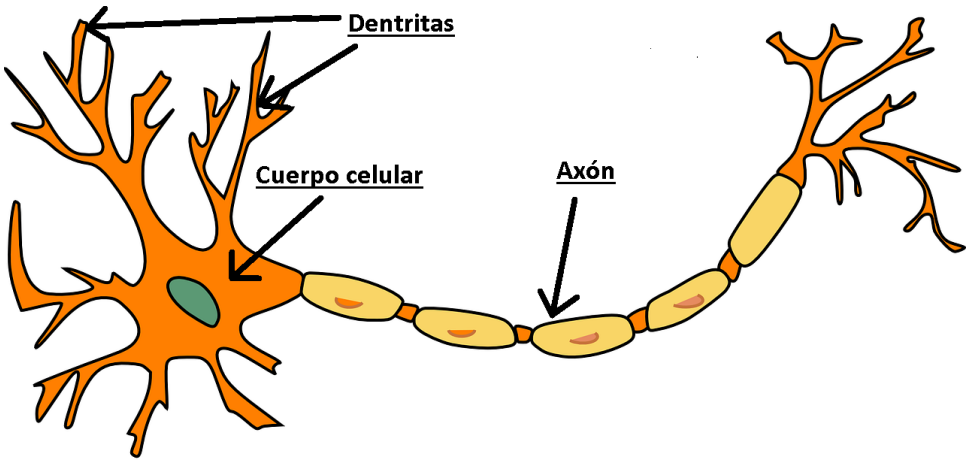


Figura 4.1: Representación de neurona biológica. [5]

- Las dendritas son el elemento receptor. Son especies de fibras que cargan de señales eléctricas al cuerpo de la célula.
- El cuerpo de la célula es el encargado de recibir las señales que le llegan de las dendritas y las interpreta.

- El axón que es una fibra larga que transmite la señal desde el cuerpo de la célula hacia otras neuronas.

El punto de contacto entre un axón de una celula y la dentrita de otra se denomina sinapsis.

4.1.2. La neurona artificial

La neurona artificial encuentra la inspiración en la neurona biológica. Esta neurona también se llama perceptrón.

> La neurona es la unidad de proceso de información fundamental es una red neuronal.
> - Haykin, 1999.

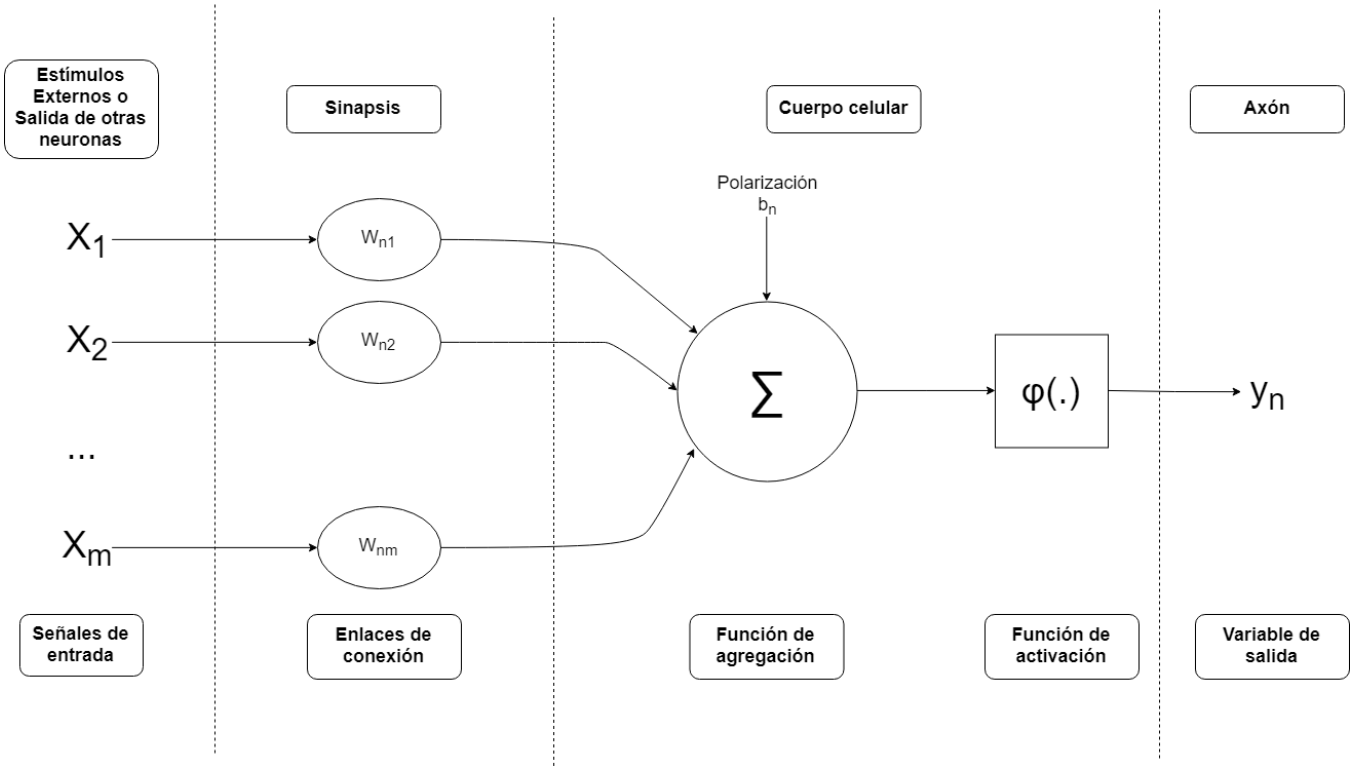


Figura 4.2: Representación de neurona artificial. Con etiquetas comparando una neurona biológica con una neurona artificial.

- Las señales de entrada que son los datos de entrada de una red neuronal o los datos de salida de otra neurona.
- Los enlaces de conexión se encuentran parametrizados por los pesos sinápticos. Hay que anotar que el primer subíndice corresponde a la neurona receptora, entonces el segundo subíndice pertenece a la neurona emisora. Si el peso sináptico es mayor que cero, entonces es una conexión excitadora; sin embargo, si el peso sináptico es menor que cero es una conexión inhibidora.
- La función de agregación suma los componentes de las señales de entrada $(x_1, x_2, .., x_n)$ multiplicadas por los enlaces de conexión$(w_{n1}, w_{n2}, ..., w_{nm})$

- La función de activación. Transformación no lineal. Lo veremos más en profundidad en el apartado ...
- Variable de salida

En términos matemáticos es posible describir a la neurona de la figura 4.2 por las siguientes ecuaciones.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{4.1}$$

y

$$y_k = \varphi(\underbrace{u_k}_{sumatorio} + \overbrace{b_k}^{polarizacin}) \tag{4.2}$$

En la formula 4.1 la variable u_k es el sumatorio de las entradas ponderadas por los pesos sinápticos. En la formula 4.2 la variable b_k es la polarización , $\varphi(.)$ es la función de activación e y_k es la señal de salida de la neurona $k-$ésima.

Polarización

La variable de polarización [1] es una parámetro externo de la neurona k que se utiliza para que el modelo sea más flexible, es decir tiene el efecto de buscar una transformación a la salida u_k, no forzando a pasar por el origen de las coordenadas de parámetros.

$$v_k = u_k + b_k \tag{4.3}$$

Entonces sabemos que la variable de polarización se puede utilizar como si fuera una neurona más. De esta forma las ecuaciones equivalentes serian las siguientes.

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{4.4}$$

y

$$y_k = \varphi(v_k) \tag{4.5}$$

En la ecuación 4.4, tenemos que añadir una nueva sinapsis (será para la neurona que actúa como el parámetro de polarización). El valor de entrada de esta neurona es, $x_0 = +1$ y el peso es, $w_{k0} = b_k$. Como podemos observar la variable de polarización es igual a un peso sináptico y por tanto esta variable tendrá la capacidad de aprender y por tanto mejorar a medida que avance el programa.

[1] En inglés se llama *bias*.

4.2. Función de activación

La función de activación [11, pág 7-8] que se denota en la 4.5 como $\varphi(.)$, va a definir la salida que tiene cada neurona. Encontramos varios tipos de funciones de activación.

1. **Función de activación escalón**: Para este tipo de activación tenemos la función siguiente, y está representada en la figura 4.3a.

$$\varphi(v) = \begin{cases} 1 \text{ si } v \geq 0 \\ 0 \text{ si } v < 0 \end{cases} \tag{4.6}$$

   Empleando la función 4.6, podemos obtener la salida de la neurona k expresado como:

$$y(k) = \begin{cases} 1 \text{ si } v_k \geq 0 \\ 0 \text{ si } v_k < 0 \end{cases} \tag{4.7}$$

   donde v_k puede ser expresada por la unión de las ecuaciones 4.1 y 4.2, por lo explicado en el final del apartado 4.1.2

$$u_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{4.8}$$

   Como se puede ver en la función 4.7 la salida solo puede tener valores igual a 0 ó 1, por lo tanto el rango de la función es $\{0, 1\}$. El principal problema de este tipo de función de activación es que cambia mucho el resultado con una variación muy pequeña de v_k. Esto se puede observar mejor en la gráfica de la función (4.3a).

2. **Función de activación sigmoidal**: Esta función soluciona los problemas que tenia la anterior, y es por esto que va a ser la función de activación más usada en la construcción de redes neuronales. Una función sigmoidal que se suele utilizar es la funcion logística, cuya expresión es la siguiente:

$$\varphi(v) = \frac{1}{1 + e^{-v}} \tag{4.9}$$

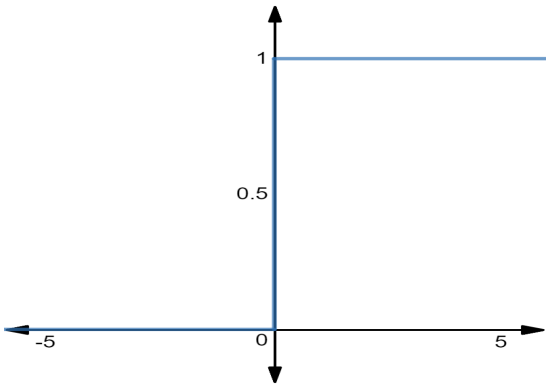
   Gracias a la representación de la función logística en la gráfica 4.3b podemos observar que el rango de valores de salida de la función es $[0, 1]$. Si queremos una función de activación que también tenga valores negativos se utiliza la función tangente hiperbólica, definida como

$$\varphi(v) = tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} \tag{4.10}$$

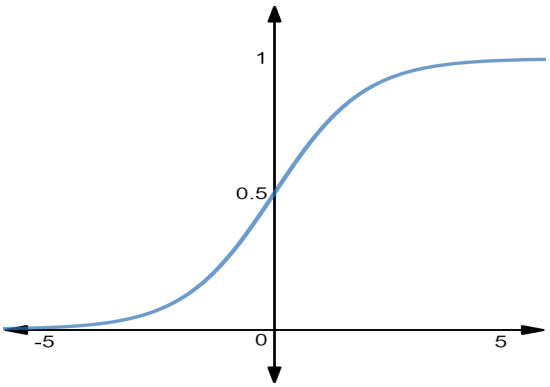
3. **Función de activación ReLU**: Esta es la función de activación mas usada en las redes neuronales convolucionales[2] (Ampliaremos información de su utilidad en 5.1.1). Esta función está definida en el rango $[0, +\infty]$ y su expresión es

$$\varphi(v) = v^{+} = max(0, v) \tag{4.11}$$

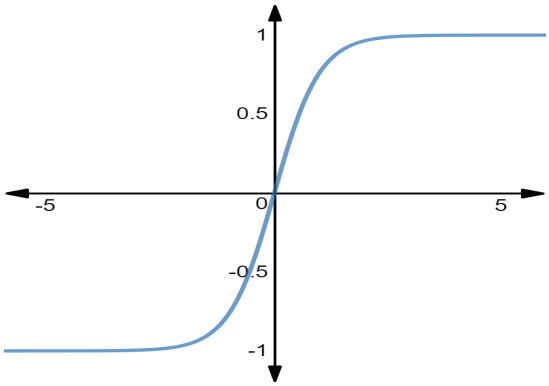
[2] Tipo de redes neuronales explicado en el apartado 5.



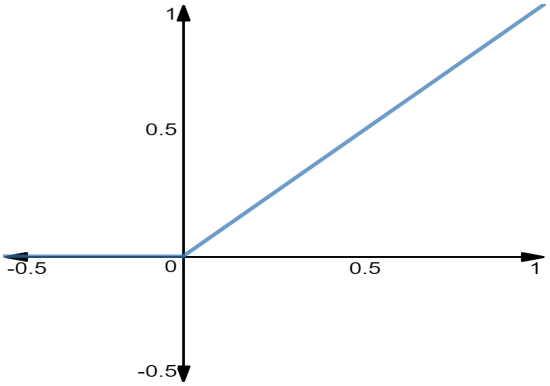
(a) Gráfica de la función de activación de escalón.



(b) Gráfica de la función logística.



(c) Gráfica de la función tangente hiperbólica.



(d) Gráfica de la función de activación ReLU.

Figura 4.3: Gráficas de los distintos tipos de funciones de activación de una neurona artificial.

4.3. El perceptrón multicapa

El perceptrón multicapa se caracteriza por tener múltiples capas. La arquitectura de un perceptron multicapa consta de una capa de entrada y una de salida y una o varias capas ocultas. En la capa de entrada es donde se encuentran las neuronas de entrada, y éstas mandan información a la primera capa oculta. Cada capa envía la información a la siguiente hasta llegar a la última capa, la capa de salida.

Todas las neuronas tienen pesos sinápticos y una función de activación que define la salida de la neurona. El objetivo principal en este tipo de problemas es saber qué pesos sinápticos van a tener cada uno de los enlaces de las neuronas. Para ello emplearemos un algoritmo que se denomina *backpropagation*, el cual explicaremos en el apartado 4.3.3.

4.3.1. Funciones de coste

Las funciones de coste se utilizan para evaluar el rendimiento que ha tenido una neurona. Es decir, podemos usar la función de coste para medir cuan distinto es el valor obtenido del valor esperado. Para poder hacer esto, necesitaremos saber cuál es el valor esperado en cada neurona. En nuestro caso, esta tarea será sencilla. Asumimos un *dataset* de entrenamiento de imágenes $x_i \in \Re^D$, cada una de ellas asociada a una etiqueta y_i (Aquí $i = 1 \dots N$ e $y_i \in 1 \dots K$). En este caso, tenemos N ejemplos, de D de dimensión y tendremos K categorías distintas. Entonces, el valor esperado va a ser la etiqueta que tenga la imagen.

Usaremos las siguientes variables en las próximas formulaciones:

- a_k representa el valor verdadero en la neurona k.
- y_k representa la predicción en la neurona k. Como podemos ver en la función 4.2.
- N representa el número de ejemplos en nuestro modelo.

Función cuadrática

$$C = \sum_k \frac{(a_k - y_k)^2}{N} = \frac{1}{N} \sum_k (\overbrace{a_k - y_k}^{error})^2 \tag{4.12}$$

Como podemos ver en la formula tiene más en cuenta el error, debido al cuadrado que hay en la formula 4.12. Este cálculo de la función puede hacer que nuestro aprendizaje disminuya la velocidad.

Cross Entropy

$$C = \frac{-1}{N} \sum_k (y_k \ln(a_k) + (1 - y_k) \ln(1 - a_k)) \tag{4.13}$$

Este tipo de función de coste hace que el proceso de aprendizaje sea más rápido. Además cuanto mayor sea la diferencia entre el valor verdadero y la predicción, más rápido aprenderá la neurona.

4.3.2. Descenso de gradiente

Es un algoritmo de optimización [12, pág 63-66] que se usa para calcular el mínimo de una función. Esto es útil para nosotros para calcular el mínimo de esa función de coste en cada neurona, así ajustar los pesos sinápticos de dicha neurona y obtener el resultado deseado.

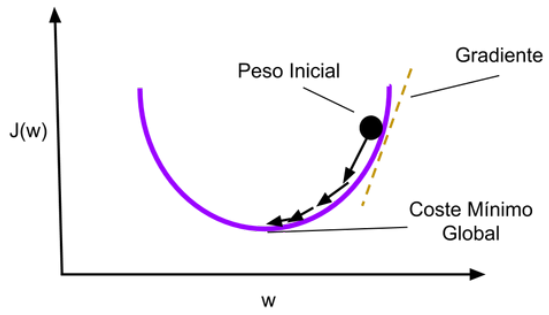


Figura 4.4: Representa el funcionamiento del algoritmo de descenso de gradiente para una función de $2 - D$.

Lo que tenemos en la figura 4.4 es la representación del algoritmo. En el eje de la y definido en la imagen como $J(w)$ tenemos la función de coste. En el eje de la x tenemos el peso sináptico de la neurona, definido en la imagen como w. Al principio suponemos un peso aleatorio, que estará representado en la imagen por la bola negra. Lo que queremos con el algoritmo es encontrar un peso que minimice la función de coste, podemos ver visualmente que ese peso estará debajo de esa parábola. Tomando la dirección negativa el gradiente de ese punto (que es la derivada de la función en ese punto), vemos que dirección tomar y paso a paso se aproxima a un mínimo. Si la función de coste fuera convexa llegaríamos a un valor óptimo.

Encontrar este mínimo es sencillo para pocas dimensiones, pero en nuestro caso vamos a tener muchos más parámetros, por lo que típicamente no ocurre que la función de coste sea convexa y se alcanzará un valor suboptimo correspondiente a una mínimo local.

4.3.3. Backpropagation

Este es el método [12, pág 66-67] más popular para entrenar el perceptrón multicapa. Este algoritmo cuenta con dos fases:

1. En la fase de alimentación hacia adelante, los pesos de la red no varían y las señales de entrada se propagan por toda la red, capa por capa, hasta alcanzar la capa de salida.

2. En la fase hacia atrás, utilizamos la función de coste al comparar la salida de la red con el resultado deseado por el modelo. La señal de error resultante de la función de coste se propaga por toda la red, pero esta vez en dirección hacia atrás. En esta fase, se realizan cambios en los pesos sinápticos de la red utilizando el algoritmo de optimización del gradiente de descenso. En definitiva, se busca conocer cuánto contribuye cada neurona al error y corregir su peso sináptico en función de dicha contribución.

5

Redes neuronales convolucionales

Este tipo de redes neuronales son específicos para tratar con señales de entrada que son matrices de datos, como los pixeles de las imágenes que queremos analizar. El nombre de convolución indica que la red aplica, en una de sus capas la operación matemática que se llama convolución. En definitiva, las redes convolucionales no son más que simples redes neuronales que en una de sus capas se usa la operación matemática de convolución sobre la matriz de pixeles.

5.1. Capa convolucional

Esta operación matemática [3, pág 330-334] que se usa en muchos ámbitos de ingeniería, no solo en el de la inteligencia artificial. Lo que queremos hacer con esta operación es combinar, en este caso dos imágenes y que la salida sea una tercera imagen que nos muestre la combinación de las dos anteriores.

La mejor forma de explicar esta capa de convolución es con el ejemplo de la imagen 5.1. Vamos a imaginar que apuntamos con una linterna sobre la imagen. Digamos que el espacio que cubre esa linterna sobre la imagen es de una matriz 3x3. En términos de procesado de imágenes, esta linterna se llama filtro [1] y la región sobre la que brilla sera la entrada de datos. También es la imagen a la que queremos hacer la operación, en el ejemplo de la imagen es 5x5. Es muy importante que el filtro tenga la misma profundidad que la entrada de datos, las dimensiones serán entonces de 3x3x1. El primer paso seria colocar el filtro en la parte superior izquierda de la imagen. A medida que el filtro se desliza alrededor de la imagen, está haciendo un sumatorio de la multiplicación de los valores del filtro con los

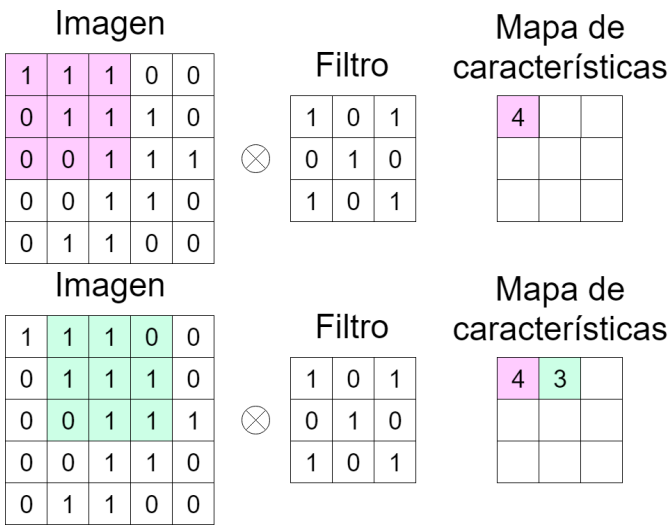


Figura 5.1: Representación de la operación de convolución sobre una imagen.

[1] También es conocida como *kernel* e incluso neurona

valores originales de la imagen. Quedando simplemente un número, que es lo que representa ese filtro sobre la imagen donde se ha colocado.
El siguiente paso seria mover el filtro hacia la derecha en un unidad, cuando se llegue al final de la imagen bajar el filtro una unidad, y así sucesivamente hasta que pase por todas las ubicaciones posibles. Después de deslizar el filtro obtenemos una matriz de números de 3x3x1. La razón de obtener una matriz 3x3 es porque hay 9 ubicaciones diferentes que un filtro 3x3 puede caber en una imagen 5x5. En cada capa de convolución podemos pasar varios filtros, obteniendo así un mapa de características por cada filtro que usemos. Cada uno de estos filtros puede considerarse como identificadores de características, es decir detectan curvas, lineas rectas, colores simples y demás. Veamos como por ejemplo en la imagen 5.2, tenemos la representación de un filtro que sirve para detectar ciertas curvas en la imagen.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

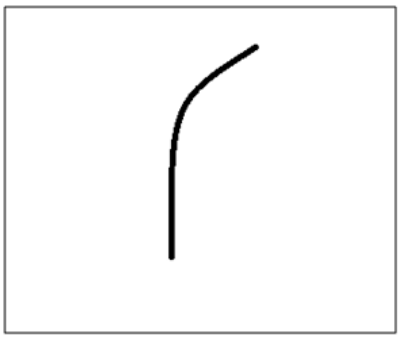
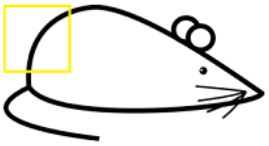


Figura 5.2: En la parte izquierda la representación de ese filtro por pixeles y en la parte derecha de la imagen podemos ver la visualización del filtro [6]
.

La Figura 5.3 representa la a aplicación del filtro anterior sobre una imagen en una zona en la que hay una curva muy parecida a la del filtro.



0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Figura 5.3: En la parte más izquierda de la imagen podemos ver la imagen donde vamos a a aplicar el filtro señalado por un cuadrado amarillo. La matriz que hay en medio de la imagen representa los píxeles de la zona amarilla. La matriz de la derecha representa el filtro en pixeles, que es el mismo de la imagen 5.2 [6]
.

Sumatorio y Multiplicacion $= (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$

Como podemos observar la aplicación de la operación da un numero muy grande, lo que significa, que en esa zona donde hemos aplicado el filtro existe esa característica que estábamos buscando con el filtro. Si aplicásemos este mismo filtro sobre otra zona que no exista esta característica, la curva, la operación nos daría un número muy bajo o incluso cero.

Cabe destacar que en una misma capa de convolución no se va a aplicar solo un filtro sobre la imagen, si no que se aplicarán diferentes filtros para conseguir detectar las diferentes

características que tiene la imagen. Además señalar que la primera capa de convolución tendrá filtros que se especializarán en características sencillas y simples. Sin embargo, en las siguientes capas de convolución poseerán filtros más complejos y específicos.

Dentro de la capa de convolución hay una serie de parámetros que se deberán ajustar a medida que la red aprende:

- El número de filtros que se utilizan en la capa de convolución
- El paso[2]. Con el que se desliza el filtro sobre la imagen. Lo normal es que este parámetro sea igual a uno, el filtro se desplaza de uno en uno. Cuando el paso es igual a dos el filtro se moverá por la imagen de dos en dos pixeles. Notar que cuando más grande sea el paso más pequeño sera el mapa de características.
- *Zero padding*. Este parámetro nos permite controlar las dimensiones del mapa de características. Lo que se hace es añadir filas y columnas de ceros alrededor de la matriz de señal de entrada. Lo habitual es preservar el tamaño espacial de la imagen de entrada [7, Apartado: Convolutional Neural Networks: Architectures, Convolution / Pooling Layers].

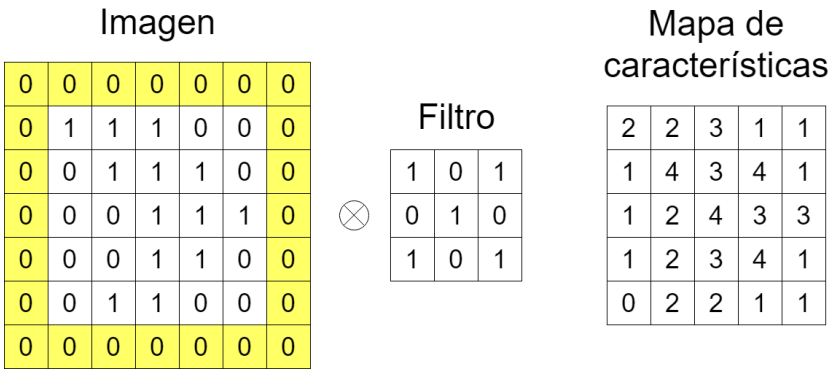


Figura 5.4: Utilización del zero padding, marcado con los cuadrados amarillos, relacionado con el problema 5.1.

5.1.1. Función de activación ReLU

Esta función de activación [3, pág 192-193] se suele realizar en la capa de convolución. ReLU es una operación matemática que se hace a cada pixel del mapa de características, reemplaza todos los negativos por ceros y los positivos no varian su valor. El objetivo de esta operación es introducir la no linealidad en nuestro modelo. La operación esta más explicada en el apartado 4.2.

5.2. Capa *Pooling*

La operación *Pooling* tiene como objetivo reducir las dimensiones de los mapas de características, para que sean más manejables, conservando la información más importante. Esta operación puede tener diferentes tipos pero el que más se utiliza actualmente es el *MaxPooling*. [3, pág 339-343] [7, Apartado: Convolutional Neural Networks: Architectures, Convolution / Pooling Layers]

[2]También llamado en inglés *stride*.

La ejecución de la operación tiene su parecido con la ejecución de la operación de convolución. En esta operación también se define un filtro, el cual se va a deslizar por todo el mapa de características realizando la operación en cada posición. La operación de *MaxPooling* es muy sencilla, el resultado de la misma seria el numero más alto de entre los números que abarque el filtro. En esta operación también existe el parámetro paso explicado en el apartado 5.1.

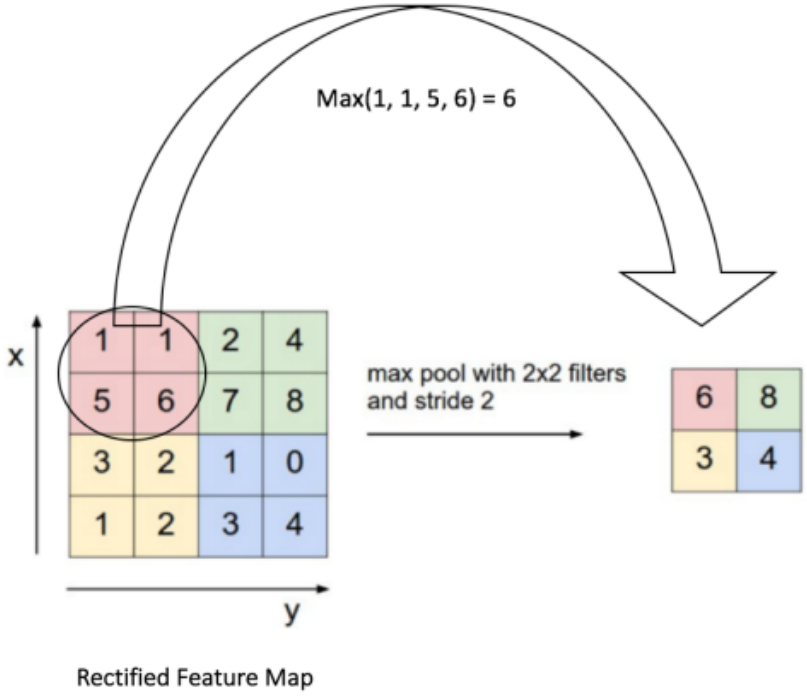


Figura 5.5: Operación *MaxPooling* en un mapa de características 4x4 con un filtro 2x2 y un paso igual a 2. [7, Apartado: Convolutional Neural Networks: Architectures, Convolution / Pooling Layers]

5.3. Capa totalmente conectada o capa densa

En la capa totalmente conectada[3] las neuronas de la capa tienen todas las conexiones a todas las actividades de la capa anterior, como una red neuronal normal. Por lo tanto esta capa comparte todas las características aprendidas en capas anteriores. En las arquitecturas de redes neuronales convolucionales se encuentran después de las capas de convolución y de *pooling*. [7, Apartado: Convolutional Neural Networks: Architectures, Convolution / Pooling Layers].

5.3.1. Capa *SoftMax*

La capa *softmax* es una capa que está totalmente conectada aunque tiene una serie de características especiales. En una arquitectura típica de red neuronal convolucional ésta es la ultima capa del modelo. Es la encargada de aproximar la probabilidad de que la imagen sea un ave o no. Para ello el número total de neuronas de la capa será igual al número total de clases de nuestro problema, en nuestro caso dos. Cada salida de la neurona representa la probabilidad de que la imagen pertenezca a una clase o a otra. Para proporcionar esa probabilidad se empleará una función de activación especifica, sera la función *softmax*.

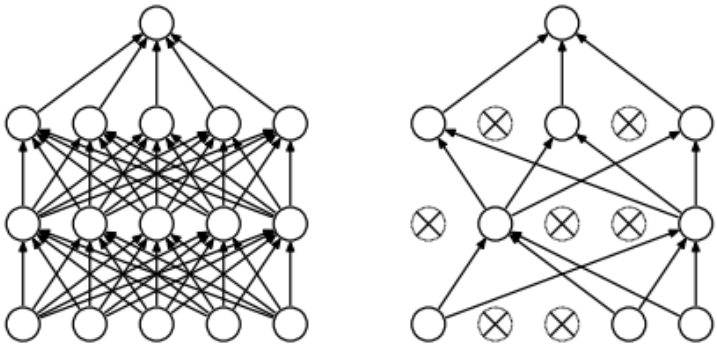
[3] En inglés se llama *fully connected*

$$y_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \text{ para } i \in \{0, 1\} \tag{5.1}$$

En la ecuación 5.1 [13, pág 209]. la variable y_i representa el valor de salida de la neurona i, y la variable z_k representa las activaciones de la neurona. Además el sumatorio de todas las variables de salida de las neuronas de esta capa debe sumar uno.

5.4. Capa de exclusión

La función de la capa de exclusión [4] [8] en las redes neuronales es que el modelo generalice el problema correctamente. Es decir, se usa para reducir el problema de sobreajuste. Lo que va a realizar esta capa es establecer a cero un conjunto aleatorio de activaciones, de esta forma las neuronas de dichas activaciones serán nulas. La red neuronal tiene que clasificar correctamente el problema incluso después de haber eliminado algunas de esas activaciones.



(a) Standard Neural Net (b) After applying dropout.

Figura 5.6: Capa de exclusión red neuronal. A la izquierda tenemos una red neuronal normal con dos capas ocultas. A la derecha tenemos esa misma red neuronal después de aplicar la capa de exclusión. [8]

Es importante saber que esta capa solo se aplica en el proceso de entrenamiento, para que el red pueda mejorar sin tener el problema de sobreajuste. Sin embargo, durante el proceso de evaluación no se empleará esta capa.

[4] En inglés se llama *dropout*.

6

Diseño red neuronal

Después de explicar qué es una red neuronal convolucional y las diferentes capas que suelen emplearse, vamos a construir nuestra propia red. En este apartado nos vamos a centrar en la construcción de la red neuronal, el código previo de la creación del repositorio y su posterior codificación vienen explicadas en el anexo A.

Para la parte práctica de este trabajo de fin de grado nos vamos a apoyar en una serie de librerías de Python. Vamos a emplear la librería *Keras* [14] para la construcción de nuestra red neuronal y otros métodos útiles. También usamos la librería *matplotlib* [15] para poder mostrar las imágenes o para la construcción de gráficas, empleamos la librería *numpy* [16] para el manejo de matrices y la librería *h5py* para poder extraer los datos de el archivo creado con el código de MATLAB.

6.1. Preprocesamiento de datos

Para la construcción de nuestra red neuronal necesitaremos de una gran cantidad de imágenes. Asimismo, como se trata de un problema de aprendizaje supervisado, tenemos que tener una etiqueta en cada imagen que sea cero o uno dependiendo si en la imagen hay un ave o no. Es por todo esto que emplearemos dos repositorios de imágenes las cuales ya vienen etiquetadas. El primero de ellos se llama CIFAR-10 [1] [17], bastante conocido en el mundo de las redes neuronales convolucionales.El segundo repositorio se llama Caltech-UCSD Birds-200-2011 [2] [9].

Entonces creamos nuestro propio repositorio de imágenes, que estará formado por la unión de los dos repositorios comentados anteriormente. Las imágenes las dividiremos, a través de carpetas, entre las que hayamos un ave en la imagen y las que no encontramos ningún ave.

Debido a la gran cantidad de imágenes que queremos usar, vamos a comprimir la información. Es por ello que creamos, con el programa MATLAB, un archivo de extensión h5 [3]. Este programa re-dimensionara las imágenes para que todas tengan el mismo numero de datos, codificara las imágenes y dividirá el conjunto de imágenes en dos partes. Conjunto de entrenamiento y la

[1] Consiste en 60000 imágenes a color 32x32 divididas en 10 clases, entre ellas se encuentra la de ave. Cada clase del repositorio cuenta con 6000 imágenes.

[2] Consiste en un repositorio solo de aves que cuenta con 11788 imágenes.

[3] Formato de archivo utilizado para el manejo de grandes conjuntos de datos.

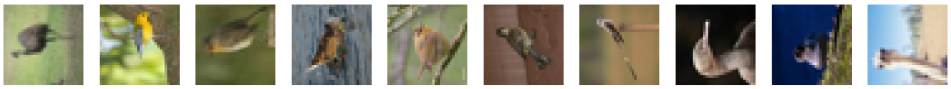


Figura 6.1: Muestra de diez imágenes de aves de nuestro conjunto de imágenes.

Figura 6.2: Muestra de diez imágenes que no hay aves de nuestro conjunto de imágenes.

conjunto de evaluación. En nuestro caso vamos a utilizar un setenta por ciento de las imágenes para entrenar nuestro modelo, y por lo tanto un treinta para evaluar el modelo.

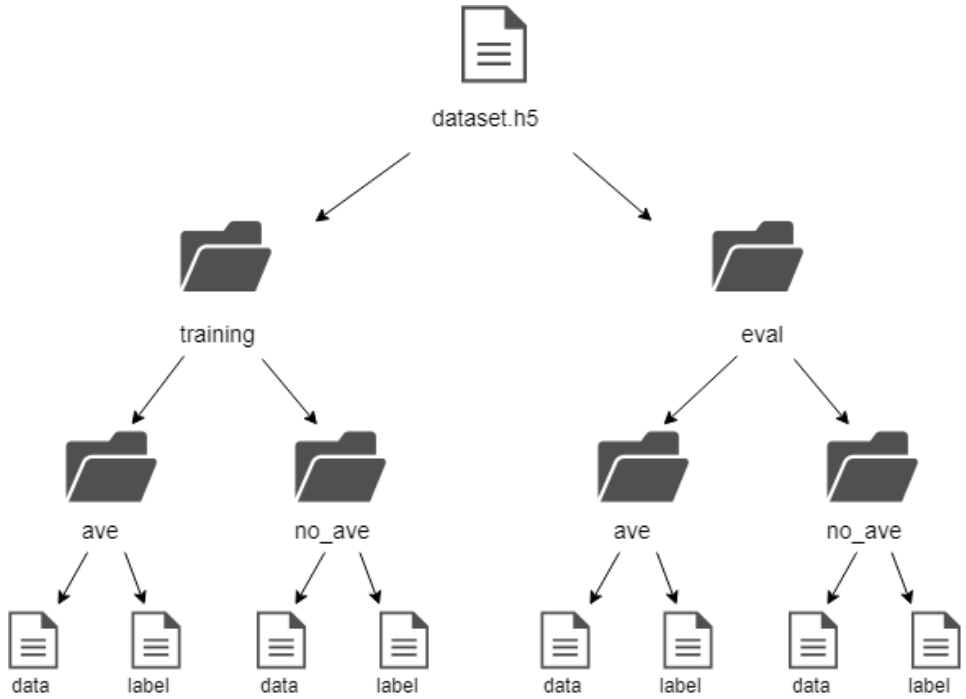


Figura 6.3: Jerarquia del archivo dataset.h5 que creamos con el codigo de MATLAB del anexo A.

Una vez tenemos el archivo donde esta recogida todo el conjunto de imágenes, crearemos un método para poder extraer la información para tenerla en Python.

```
import numpy as np
import h5py

def load_dataset(path):

    dataset=h5py.File(path,'r') #r de read o rb en python

    training_set_positive=np.array(dataset['/training/ave/data'][:])
    training_set_negative=np.array(dataset['/training/no_ave/data'][:])

    training_label_positive=np.array(dataset['/training/ave/label'][:])
    training_label_negative=np.array(dataset['/training/no_ave/label'][:])

    eval_set_positive=np.array(dataset['/eval/ave/data'][:])
    eval_set_negative=np.array(dataset['/eval/no_ave/data'][:])
```

```
    eval_label_positive=np.array(dataset['/eval/ave/label'][:])
    eval_label_negative=np.array(dataset['/eval/no_ave/label'][:])


    training_set_positive=np.rollaxis(training_set_positive,1,4)
    training_set_negative=np.rollaxis(training_set_negative,1,4)
    training_set = np.concatenate((training_set_positive, training_set_negative),
        0)

    eval_set_positive=np.rollaxis(eval_set_positive,1,4)
    eval_set_negative=np.rollaxis(eval_set_negative,1,4)
    eval_set = np.concatenate((eval_set_positive, eval_set_negative), 0)


    training_labels = np.concatenate((training_label_positive,
        training_label_negative), 0)


    eval_labels = np.concatenate((eval_label_positive, eval_label_negative), 0)

    return training_set, training_labels, eval_set, eval_labels

x_train, y_train, x_test, y_test = load_dataset('.\TFG\dataset.h5')
```

Este método devolverá cuatro matrices:

1. x_train: Esta matriz guardara la información de todas las imágenes de entrenamiento.
2. y_train: Esta matriz guardara las etiquetas de las imagines correspondientes con la matriz x_train, como un numero entero, uno en caso de que sea un ave o cero en caso contrario.
3. x_test: Esta matriz guardar toda la información de todas las imágenes de evaluación.
4. y_test: Esta matriz guardara las etiquetas de las imagines correspondientes con la matriz x_test, como un numero entero, uno en caso de que sea un ave o cero en caso contrario.

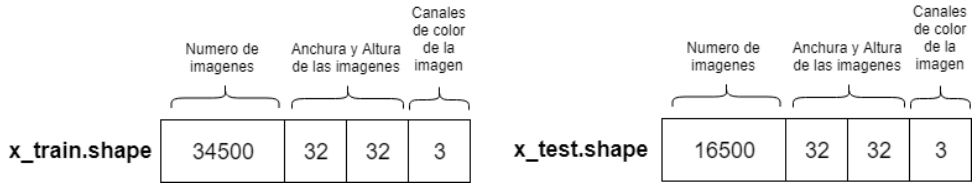


Figura 6.4: Tamaño las matrices x_train y x_test en cada dimensión. Y significado de cada dimensión.

Antes de la construcción de nuestro modelo tenemos que normalizar las imágenes y transformar el numero entero de las matrices de las etiquetas, en una matriz binaria. Como las imágenes que usamos tienen 24 bits de profundidad, es decir, que el valor máximo de cada pixel es doscientos cincuenta y cinco. Entonces convertimos las matrices en tipo *float* y las dividimos por doscientos cincuenta y cinco. Para conseguir la matriz binaria de las etiquetas usaremos la función to_categorical de la biblioteca keras.

```
#Primero convertimos los pixels en tipo float para que se puedan tener decimales
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
#Ahora dividimos todo el conjunto de datos entre 255.
x_train /= 255
x_test /= 255
```

```
num_classes = 2
from keras.utils import to_categorical
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

6.2. Construcción de la red neuronal

Para la construcción de nuestro clasificador vamos a emplear la librería Keras. Esta envuelta en la librería principal de redes neuronales que se llama TensorFlow.

Primero crearemos un método que defina la estructura de nuestra red neuronal. Esta sera una secuencia de las capas vistas en el capitulo 6. Además creamos un optimizador de la red neuronal y determinaremos la función con la que calcularemos el error.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
    MaxPooling2D
from keras.optimizers import Adam
def base_model():

    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same',input_shape=x_train.shape[1:]))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))

    optimizador = Adam(lr=0.001)

    #Utilizaremos un error de tipo crossentropy.
    model.compile(loss='categorical_crossentropy', optimizer=optimizador, metrics
        =['accuracy'])

    return model
cnn_n = base_model()
```

Empleamos el metodo summary para poder ver un resumen de nuestro modelo.

```
cnn_n.summary()
Out:
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
activation_1 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 32)	0

```
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1606144
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
_________________________________________________________________
activation_5 (Activation)    (None, 2)                 0
=================================================================
Total params: 1,663,490
Trainable params: 1,663,490
Non-trainable params: 0
```

Antes de ponernos a entrenar nuestro modelo, crearemos un objeto que genera lotes de imágenes con la posibilidad de que estas mismas tengan cierto nivel de ruido. Es decir, puede generar una imagen a partir de la original que varia en pequeños detalles. De esta forma creamos imágenes que se encuentran ya etiquetadas para el modelo sin necesidad de buscar otras nuevas.

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    #Desplaza horizontalmente una fraccion del ancho de la imagen aleatoriamente.
    width_shift_range=0.1,
    #Desplaza verticalmente una fraccion la altura de la imagen aleatoriamente.
    height_shift_range=0.1,
    #Aleatoriamente da la vuelta imagenes de forma horizontal.
    horizontal_flip=True,
    #Aleatoriamente da la vuelta imagenes de forma vertical.
    vertical_flip=True)

datagen.fit(x_train)
```

.

Una vez que tenemos todo inicializamos el numero de imágenes por lote y el numero de épocas y entrenamos nuestro modelo.

```
batch_size = 100
epochs = 100
# Entrenamos el modelo por "batches" generados por data.flow().
historia = cnn_n.fit_generator(datagen.flow(x_train, y_train,
                    batch_size=batch_size),
                    epochs=epochs,
                    validation_data=(x_test, y_test),
                    shuffle=True)
```

Cuando nuestro modelo ya ha entrenado, vamos a guardar la arquitectura del modelo y también los pesos que a calculado con el entrenamiento, para poder usarlo en el futuro. Emplearemos la función save que esta implementada en la librería keras.

```
import os
save_dir = 'C:\\Users\\Alonso\\TFG\\saved_models'
model_name = 'modelo_aves_entrenado.h5'
model_path = os.path.join(save_dir, model_name)
cnn_n.save(model_path)
print('Saved trained model at %s ' % model_path)
```

6.3. Análisis de los datos obtenidos

Vamos a generar unas gráficas,la primera observaremos la precisión del modelo en el conjunto de entrenamiento y la de evaluación a través de las épocas. La segunda podremos ver como interactúa la perdida del modelo a través de las épocas en el conjunto de entrenamiento y en la de evaluación

```
import matplotlib.pyplot as plt
%matplotlib inline

#Grafica que compara la precision del modelo a traves de las epocas.
plt.plot(historia.history['acc'])
plt.plot(historia.history['val_acc'])
plt.title('Precision del modelo')
plt.ylabel('precision')
plt.xlabel('epocas')
plt.legend(['entrenamiento', 'evaluacion'], loc='lower right')
plt.show()
```

```
#Grafica que compara el error del modelo a traves de las epocas.
plt.plot(historia.history['loss'])
plt.plot(historia.history['val_loss'])
plt.title('Error del modelo')
plt.ylabel('error')
plt.xlabel('epocas')
plt.legend(['entrenamiento', 'evaluacion'], loc='upper right')
plt.show()
```

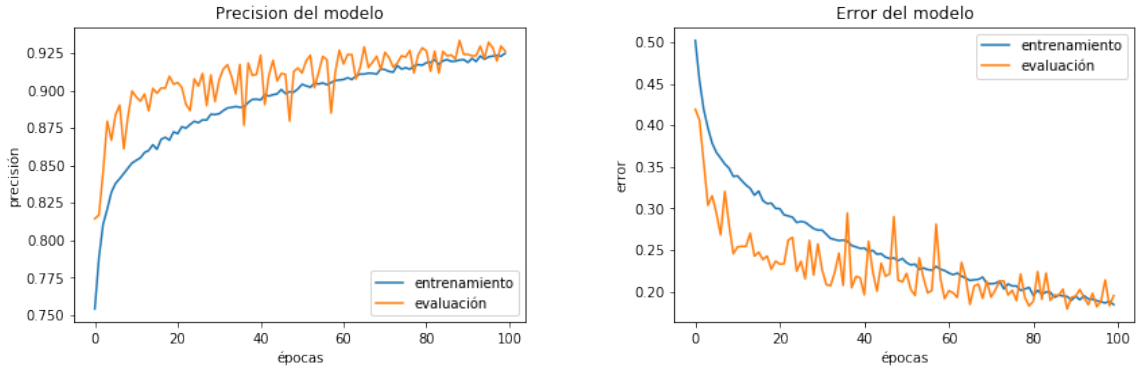


Figura 6.5: Gráficas que compara la error y la precisión del modelo a través de las épocas.

Como podemos ver en las gráficas a medida que pasan las épocas y el modelo va entrenando la precisión del modelo va incrementando. Asimismo el error del modelo también decrece inversamente a medida que pasan las épocas. Además las lineas que comparan la partición de evaluación con la de entrenamiento se encuentran juntas, lo que quiere decir que el modelo no tiene un problema de *overfitting* (Concepto explicado en el apartado 3.2.1).

Asimismo evaluamos el rendimiento del modelo con la partición de evaluación del conjunto de imágenes.

```
scores = cnn_n.evaluate(x_test, y_test)
print('Error evaluacion:', scores[0])
print('Precision evaluacion:', scores[1]*100)
Out:
16502/16502 [==============================] - 14s 820us/step
Error evaluacion: 0.1949448962143379
Precision evaluacion: 92.63725608800371
```

Como podemos ver hemos empleamos un método de la librería keras que se llama evaluate. Después de usarlo imprimos por pantalla el error y la precisión del modelo en la parte de evaluación. Podemos ver que la precisión del modelo tampoco es muy alta, esto puede ser debido a que no tenemos una gran cantidad de imagenes en el conjunto de datos, o por la simplicidad del modelo usado. También creamos la matriz de confusión para poder observar en que se equivoca el modelo.

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test[:,1],cnn_n.predict_classes(x_test))
Out:
array([[9903,  598],
       [ 617, 5384]], dtype=int64)
```

Resultado de 16500 imágenes de evaluación

(10500 imágenes no son aves, 6000 imágenes son aves)

	Predijo "no ave"	Predijo "ave"
No ave	**9903**	**598**
Ave	**617**	**5384**

Figura 6.6: Matriz de confusión de nuestro modelo.

Como podemos ver para calcular la matriz de confusión hemos empleado una función de la librería *sklearn* [4] [18].

Vamos a profundizar en esas imágenes en las que nos estamos equivocando. Primero vamos a ver las imágenes que predicen que son aves con más de un noventa y cinco por ciento de seguridad, pero en realidad no hay ningún ave en la imagen

Figura 6.7: Diez imágenes que el modelo confunde con aves.

Como podemos observar en las imágenes el modelo se equivoca normalmente con muchos aviones, por el claro parecido que tienen con las aves. Haremos el mismo análisis para las imágenes que predicen que no hay ningún ave con más de un noventa y cinco por ciento de seguridad, sin embargo, si que hay un ave en la imagen.

[4] Librería especializada en el aprendizaje automático

Figura 6.8: Diez imágenes que el modelo confunde con no aves.

Observamos que los aves que no logra clasificar bien están posados, por lo tanto es más difícil para el modelo clarificarlos correctamente. Incluso algunas de estas imágenes el ave se mimetiza con el fondo o solo sale un parte del ave.

6.4. Predicciones del modelo

Ahora vamos a seleccionar una foto aleatoria de la conjunto de evaluación del repositorio de imágenes y mostrar cuanto porcentaje de características tiene la imagen de parecido con un ave. Para ello primero tenemos que multiplicar el conjunto de imagen por doscientos cincuenta y cinco, debido a que lo dividimos por esa cantidad para normalizar el conjunto de imágenes. Y también lo tenemos que transformar a tipo 'uint8'.

```
X = x_test * 255
X = X.astype('uint8')
```

Primero vamos a calcular la probabilidad de todas las imágenes del conjunto de evaluación y lo guardamos en la variable predicciones. Gracias a la ultima capa es la softmax, y la utilización del metodo implementado predict_proba.

```
predicciones = cnn_n.predict_proba(x_test)
```

Luego inicializamos un número aleatorio, para que escoger la imagen, e imprimiremos la imagen y una frase diciendo si es un ave o no lo es. Luego obtenemos la probabilidad de que la imagen que le pasemos sea un ave o no.

```
import random
import matplotlib.pyplot as plt
%matplotlib inline
num = random.randint(0,len(x_test))
if(y_test[num][1] == 1):
    print("Es un ave!")
else:
    print("No es un ave")
plt.imshow(X[num])
```

```
labels = [u'No ave ('+str("{0:.2f}".format(predicciones[num][0]*100))+'%)',
          u'Ave ('+str("{0:.2f}".format(predicciones[num][1]*100))+'%)']
plt.pie(predicciones[num], labels = labels,explode=[0, 0.1])
```

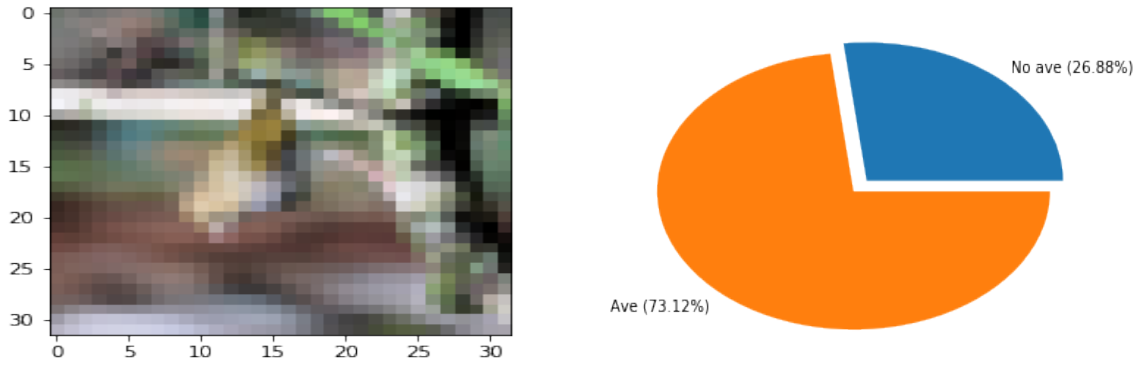


Figura 6.9: Muestra la imagen junto con la predicción que realiza el modelo.

7

Conclusiones y trabajo futuro

El presente trabajo se ha centrado en el estudio teórico de la base del aprendizaje profundo, y más concretamente de las redes neuronales convolucionales orientadas al reconocimiento de imágenes. Durante la realización de este trabajo se ha aprendido en que consiste este campo, que tanto ha crecido en los últimos años. También se ha querido dar un enfoque practico al proyecto, realizando de esta forma una red neuronal básica, que esta fundamentada en los conceptos teóricos explicados. Gracias a este enfoque hemos descubierto la extraordinaria capacidad y versatilidad de este tipo de redes neuronales convolucionales, siendo el único obstáculo nuestra propia imaginación.

Para la implementación de la parte practica del proyecto se ha empleado la librería *Keras*. Esta librería nos permite la construcción de una red neuronal de una forma muy sencilla y simple. De esta forma teniendo claros los conceptos teóricos relacionados con las redes neuronales cualquier persona seria capaz de escribir un red básica. Si quisiéramos profundizar en nuestra propia red para maximizar los resultados tendríamos que emplear la librería en la que *Keras* se basa, que es *TensorFlow*.

En concreto hemos querido centrar el estudio de nuestro trabajo en la clasificación de imágenes, entre las que se veía un ave en la imagen y en las que no. Se ha podido observar la gran potencia de este tipo de redes, hemos conseguido implementar una red neuronal que distingue consigue distinguir a las aves de otros objetos con una precisión de mas del noventa y dos por ciento. Esto ha sido empleando un red neuronal básica con menos de diez capas, con un repositorio de imágenes de baja resolución pequeño para aplicaciones del mundo real y entrenando la red con un ordenador medio. Como trabajo para el futuro se podría mejorar esa precisión del modelo, sofisticando la propia red neuronal, mejorando el repositorio de imágenes o ambas. Asimismo, una vez que el modelo reconoce que si que hay un ave, se podría mejorar la red para que clasificara las aves por familias.

El campo estudiado en este trabajo de fin de grado es relativamente nuevo, si que es cierto que los conceptos teóricos existían desde hace tiempo, pero se carecía de la capacidad computacional de la actualidad. Por eso a día de hoy, las grandes empresas tecnológicas están invirtiendo de una forma u otra en esta tecnología en la actualidad. En este trabajo se han descrito las tecnologías actuales, aunque nadie sabe si en el futuro se continuaran empleando o surgirán otro tipo de algoritmos o redes más eficientes.

Bibliografía

[1] Wilhelm Burger and Mark Burge. *Digital image processing: an algorithmic introduction using Java*. Springer, London, 2nd ed edition, 2016.

[2] Apurba Das. Introduction to Digital Image. In *Guide to Signals and Patterns in Image Processing: Foundations, Methods and Applications*, pages 1–42. Springer International Publishing, Cham, 2015.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[4] Robert Harper. Understanding the Machine Learning in AIOps, Part 1. Accedido el 21-03-2018 en https://www.moogsoft.com/blog/aiops/understanding-machine-learning-aiops/, July 2017.

[5] Imagen gratis en Pixabay - Neurona, Célula Del Nervio, Axón. Accedido el 11-04-2018 en https://pixabay.com/es/neurona-c%C3%A9lula-del-nervio-ax%C3%B3n-296581/.

[6] Adit Deshpande. A Beginner's Guide To Understanding Convolutional Neural Networks. Accedido en 23-04-2018 a https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/.

[7] CS231n Convolutional Neural Networks for Visual Recognition. Accedido en 20-04-2018 el http://cs231n.github.io/.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[9] Caltech-UCSD Birds-200-2011. Repositorio descargable desde: http://www.vision.caltech.edu/visipedia/CUB-200-2011.html. Accedido 10-06-2018.

[10] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA, 2012.

[11] Edgar Nelson Sánchez Camperos and Alma Yolanda Alanís García. *Redes neuronales: conceptos funadamentales y aplicaciones a control automático*. Pearson/Prentice-Hall, Madrid, 2006. OCLC: 954099680.

[12] Raquel Flórez López and José Miguel Fernández Fernández. *Las redes neuronales artificiales: fundamentos teóricos y aplicaciones prácticas*. Netbiblo, Oleiros (La Coruña), 1ª ed. en español edition, 2008.

[13] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, 2006.

[14] Keras Documentation. Accedido en 10-06-2018 a https://keras.io/.

[15] Overview — Matplotlib 2.2.2 documentation. Accedido en 10-06-2018 a https://matplotlib.org/contents.html.

[16] Numpy and Scipy Documentation — Numpy and Scipy documentation. Accedido en 07-06-2018 a https://docs.scipy.org/doc/.

[17] CIFAR-10 and CIFAR-100 datasets. Repositorio descargable desde: http://www.vision.caltech.edu/visipedia/CUB-200-2011.html. Accedido 10-06-2018.

[18] Documentation scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation. Accedido en 07-06-2018 a http://scikit-learn.org/stable/documentation.html.

A

Manual del programador

Lo primero que vamos a realizar es la creación de nuestro propio repositorio, uniendo el repositorio CIFAR-10 [17] y Caltech-UCSD Birds-200-2011 [9]. Tenemos que separar las imágenes de aves y de las imágenes que no tienen aves en carpetas diferentes para que nuestro código de MATLAB guarde bien las etiquetas de las imágenes. Además tenemos que cambiar el nombre de las imágenes de aves por +1, +2, +3..., +n siendo el n el numero de la imagen y el nombre las imágenes de no aves por _1, _2, _3,..., _n siendo el n el numero de la imagen.

Creación repositorio

Lo que vamos a realizar con este codigo es crear nuestro propio repositorio repositorio. Que seran dos carpetas, una de imagenes de aves y otra de imagenes de no aves.

In [2]:

```
import shutil, os
import random
```

Preparamos de todo el repositorio de imagenes

Con el siguiente codigo vamos a dividir el dataset en imagenes con aves de las que no tienen aves del repositorio CIFAR-10. Las variables path_cifar sera la ruta donde se encuentran las imagenes y path_destino sera la ruta donde queremos guardar las imagenes.

In [24]:

```
nombre_img = os.listdir(path_cifar)
#Separación de fotos de aves y de no aves.
if os.path.exists(path_cifar):
    for nombre in nombre_img:
        if nombre[-8:-4] == 'bird':
            shutil.copy(path_cifar+os.sep+nombre,path_destino+'ave')
        else:
            shutil.copy(path_cifar+os.sep+nombre,path_destino+'no_ave')
```

El repositorio Caltech-UCSD Birds-200-2011 está dividido en subcarpetas, donde dentro se encuentran las imagenes. La variable path_Caltech es donde se ecuentran esas carpetas con las imagenes y path_destino sera la ruta donde queremos guardar las imagenes.

In [ ]:

```
carpetas = os.listdir(path_Caltech)
for carpeta in carpetas:
    imagenes = os.listdir(path_Caltech+os.sep+carpeta)
    for imagen in imagenes:
        shutil.copy(path_Caltech+os.sep+carpeta+os.sep+imagen,path_destino+
'ave')
```

Preparamos una muestra del repositorio que tenemos

Vamos a crear una muestra de 51000 imagenes.(35000 imagenes que no son aves y 16000 imagenes que si son aves) Las variables path_origen y path_destino estan referenciadas a la ruta de la carpeta de origen y destino respectivamente.

Primero meteremos las imagenes de aves

In [10]:

```
nombre_img = os.listdir(path_origen+'ave')
imagenes = random.sample(nombre_img,k = 16000)
if os.path.exists(path_destino+'ave'):
    for i in range(len(imagenes)):
        shutil.copy(path_origen+'ave'+os.sep+imagenes[i],path_destino+'ave'
)
        if i%1000 == 0:
            print("Copiadas " +str(i)+" imagenes")
```

```
Copiadas 0 imagenes
Copiadas 1000 imagenes
Copiadas 2000 imagenes
Copiadas 3000 imagenes
Copiadas 4000 imagenes
Copiadas 5000 imagenes
Copiadas 6000 imagenes
Copiadas 7000 imagenes
Copiadas 8000 imagenes
Copiadas 9000 imagenes
Copiadas 10000 imagenes
Copiadas 11000 imagenes
Copiadas 12000 imagenes
Copiadas 13000 imagenes
Copiadas 14000 imagenes
Copiadas 15000 imagenes
```

Despues meteremos las imagenes que no son aves

In [18]:

```
nombre_img = os.listdir(path_origen+'no_ave')
imagenes = random.sample(nombre_img,k = 35000)
if os.path.exists(destino):
    for i in range(len(imagenes)):

shutil.copy(path_origen+'no_ave'+os.sep+imagenes[i],path_destino+'no_ave')
        if i%1000 == 0:
            print("Copiadas " +str(i)+" imagenes")
```

```
Copiadas 0 imagenes
Copiadas 1000 imagenes
Copiadas 2000 imagenes
Copiadas 3000 imagenes
Copiadas 4000 imagenes
Copiadas 5000 imagenes
Copiadas 6000 imagenes
Copiadas 7000 imagenes
Copiadas 8000 imagenes
Copiadas 9000 imagenes
Copiadas 10000 imagenes
Copiadas 11000 imagenes
Copiadas 12000 imagenes
Copiadas 13000 imagenes
Copiadas 14000 imagenes
Copiadas 15000 imagenes
Copiadas 16000 imagenes
Copiadas 17000 imagenes
```

```
Copiadas 18000 imagenes
Copiadas 19000 imagenes
Copiadas 20000 imagenes
Copiadas 21000 imagenes
Copiadas 22000 imagenes
Copiadas 23000 imagenes
Copiadas 24000 imagenes
Copiadas 25000 imagenes
Copiadas 26000 imagenes
Copiadas 27000 imagenes
Copiadas 28000 imagenes
Copiadas 29000 imagenes
Copiadas 30000 imagenes
Copiadas 31000 imagenes
Copiadas 32000 imagenes
Copiadas 33000 imagenes
Copiadas 34000 imagenes
```

Cambiamos el nombre de las imagenes

In [32]:

```
lista_aves = os.listdir(path_destino+'ave')
lista_no_aves=(os.listdir(path_destino+'no_ave'))
```

In [36]:

```
len(lista_aves)
```

Out[36]:

```
16000
```

In [37]:

```
len(lista_no_aves)
```

Out[37]:

```
35000
```

In [38]:

```
for i in range(len(lista_aves)):
    newfilename = '+'+str(i+1)
    os.rename(img1+os.sep+lista_aves[i],img1+os.sep+newfilename+'.png')
for i in range(len(lista_no_aves)):
    newfilename = '_'+str(i+1)
    os.rename(img2+os.sep+lista_no_aves[i],img2+os.sep+newfilename+'.png')
```

Ahora ese repositorio que hemos creado lo vamos a comprimir con el siguiente codigo de MATLAB. Este codigo generara un archivo "h5".

```
%% Conjunto de entrenamiento imagenes no aves
clc;
clear all;
close all;
srcFiles=dir('./no_ave/');
path='./no_ave/';
imagename=[path,'_1'];
I = imread(imagename,'png');
datasetname = '/training/no_ave/data';
datasetnamelabel = '/training/no_ave/label';
filename='dataset.h5';
imgs_dset = double(ones([size(I) length(srcFiles)]));
Isize=size(I);
dset_size = [size(I) Inf];
h5create(filename, datasetname, dset_size,'Datatype',
                class(I), 'Chunksize', [Isize(1) Isize(2) Isize(3) 1 ] );
h5create(filename,datasetnamelabel, [1 Inf],
                'Datatype', 'double', 'Chunksize', [ 1 1 ]);
for i=1:24500 %porque es el 70% de 35000
    imagename = [path,'_',int2str(i)];
    I = imread(imagename,'png');
    I = imresize(I, [32,32]);%Redimensionamos la imagen
    if i<(length(srcFiles)*0.7)
        imagename
    end
    h5write(filename, datasetname,I,[1 1 1 i],[Isize(1) Isize(2) Isize(3) 1]);
    h5write(filename, datasetnamelabel, 0, [1 i], [1 1]);
end
info = h5info(filename, datasetname)
disp(info.Dataspace.Size)

%% Conjunto de evaluacion imagenes no_aves
clc;
clear all;
close all;
srcFiles=dir('./no_ave/');
path='./no_ave/';
imagename=[path,'_1'];
I = imread(imagename,'png');
datasetname = '/eval/no_ave/data';
datasetnamelabel = '/eval/no_ave/label';
filename='dataset.h5';
imgs_dset = double(ones([size(I) length(srcFiles)]));
Isize=size(I);
dset_size = [size(I) Inf];
h5create(filename, datasetname, dset_size, ...
         'Datatype', class(I), 'Chunksize', [Isize(1) Isize(2) Isize(3) 1] );
h5create(filename,datasetnamelabel, [1 Inf], ...
'Datatype', 'double', 'Chunksize', [ 1 1 ]);
for i=24500:35000
    imagename = [path,'_',int2str(i)];
    I = imread(imagename,'png');
    I = imresize(I, [32,32]); %Redimensionamos la imagen
    if i<(length(srcFiles)-3)
        imagename
    end
    h5write(filename, datasetname,I,[1 1 1 i-24500+1],[Isize(1) Isize(2) Isize(3)
        1]);
    h5write(filename, datasetnamelabel, 0, [1 i-24500+1], [1 1]);
end
```

```
info = h5info(filename, datasetname)
disp(info.Dataspace.Size)

%% Conjunto de entrenamiento imagenes aves
clc;
clear all;
close all;
srcFiles=dir('./ave/');
path='./ave/';
imagename=[path,'+1'];
I = imread(imagename,'png');
datasetname = '/training/ave/data';
datasetnamelabel = '/training/ave/label';
filename='dataset.h5';
imgs_dset = double(ones([size(I) length(srcFiles)]));
Isize=size(I);
dset_size = [size(I) Inf];
h5create(filename, datasetname, dset_size, ...
         'Datatype', class(I), 'Chunksize', [Isize(1) Isize(2) Isize(3) 1] );
h5create(filename,datasetnamelabel, [1 Inf],
                'Datatype', 'double', 'Chunksize', [ 1 1 ]);
for i=1:10000 %porque es el 70% de 16500
    imagename = [path,'+',int2str(i)];
    I = imread(imagename,'png');
    I = imresize(I, [32,32]); %Redimensionamos la imagen
    if i<(round(length(srcFiles)*0.7))
        imagename
    end
    h5write(filename, datasetname,I,[1 1 1 i],[Isize(1) Isize(2) Isize(3) 1]);
    h5write(filename, datasetnamelabel, 1, [1 i], [1 1]);
end
info = h5info(filename, datasetname)
disp(info.Dataspace.Size)

%% Conjunto de evaluacion imagenes aves
clc;
clear all;
close all;
srcFiles=dir('./ave/');
path='./ave/';
imagename=[path,'+1'];
I = imread(imagename,'png');
datasetname = '/eval/ave/data';
datasetnamelabel = '/eval/ave/label';
filename='dataset.h5';
imgs_dset = double(ones([size(I) length(srcFiles)]));
Isize=size(I);
dset_size = [size(I) Inf];
h5create(filename, datasetname, dset_size, ...
         'Datatype', class(I), 'Chunksize', [Isize(1) Isize(2) Isize(3) 1] );
h5create(filename,datasetnamelabel, [1 Inf],
                'Datatype', 'double', 'Chunksize', [ 1 1 ]);
for i=10000:16000
    imagename = [path,'+', int2str(i)];
    I = imread(imagename,'png');
    I = imresize(I, [32,32]); %Redimensionamos la imagen
    if i<(length(srcFiles)-3)
        imagename
    end
    h5write(filename, datasetname,I,[1 1 1 i-10000+1],[Isize(1) Isize(2) Isize(3)
        1]);
    h5write(filename, datasetnamelabel, 1, [1 i-10000+1], [1 1]);
end
```

```
info = h5info(filename, datasetname)
disp(info.Dataspace.Size)
```

Construccion del modelo y entrenamiento

Primero tenemos que crear el metodo que acondicionara nuestro dataset

In [1]:

```
import numpy as np
import h5py
```

```
C:\Users\Alonso\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarn
ing: Conversion of the second argument of issubdtype from `float` to `np.fl
oating` is deprecated. In future, it will be treated as `np.float64 == np.d
type(float).type`.
  from ._conv import register_converters as _register_converters
```

In [2]:

```
def load_dataset(path):

    dataset=h5py.File(path,'r') #r de read o rb en python

    training_set_positive=np.array(dataset['/training/ave/data'][:])
    training_set_negative=np.array(dataset['/training/no_ave/data'][:])

    training_label_positive=np.array(dataset['/training/ave/label'][:])
    training_label_negative=np.array(dataset['/training/no_ave/label'][:])

    eval_set_positive=np.array(dataset['/eval/ave/data'][:])
    eval_set_negative=np.array(dataset['/eval/no_ave/data'][:])

    eval_label_positive=np.array(dataset['/eval/ave/label'][:])
    eval_label_negative=np.array(dataset['/eval/no_ave/label'][:])


    training_set_positive=np.rollaxis(training_set_positive,1,4)
    training_set_negative=np.rollaxis(training_set_negative,1,4)
    training_set = np.concatenate((training_set_positive,
training_set_negative), 0)

    eval_set_positive=np.rollaxis(eval_set_positive,1,4)
    eval_set_negative=np.rollaxis(eval_set_negative,1,4)
    eval_set = np.concatenate((eval_set_positive, eval_set_negative), 0)


    training_labels = np.concatenate((training_label_positive,
training_label_negative), 0)


    eval_labels = np.concatenate((eval_label_positive, eval_label_negative)
, 0)

    return training_set, training_labels, eval_set, eval_labels
```

Hemos dispuesto el metodo para que divida el dataset en 4 partes. x_train, y_train seran las partes de entramiento y conformaran el 70% de nuestro dataset. x_test, y_test seran las partes para evaluar el rendimiento del programa y seran un 30% de nuestro dataset.

In [3]:

```
x_train, y_train, x_test, y_test = load_dataset('.\TFG\dataset.h5')
```

In [4]:

```
x_train.shape
```

Out[4]:

```
(34500, 32, 32, 3)
```

In [305]:

```
y_train.shape
```

Out[305]:

```
(34500, 2)
```

In [5]:

```
x_test.shape
```

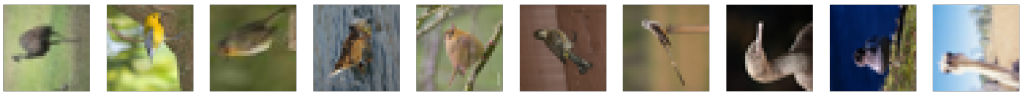
Out[5]:

```
(16502, 32, 32, 3)
```

Muestra de 10 imagenes con la etiqueta de ave

In [303]:

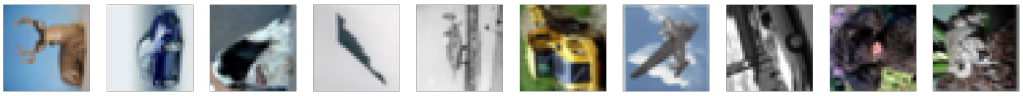
```
I = x_train*255
I = I.astype('uint8')
f, axarr = plt.subplots(1,10,figsize=(32, 32))
for i in range (10):
    num = random.randint(0,10000)
    axarr[i].axis('off')
    axarr[i].imshow(I[num])
```



Muestra de 10 imagenes con la etiqueta de no_ave

In [304]:

```
I = x_train*255
I = I.astype('uint8')
f, axarr = plt.subplots(1,10,figsize=(32, 32))
for i in range (10):
    num = random.randint(10000,34449)
    axarr[i].axis('off')
    axarr[i].imshow(I[num])
```



Convertimos las etiquetas de las imagenes en matrices binarias.

In [8]:

```
num_classes = 2
from keras.utils import to_categorical
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

```
Using TensorFlow backend.
```

Ahora vamos a normalizar las imagenes. Como las imagenes tienen una profundidad de 24 bits, quiere decir que como maximo,un pixel, puede alcanzar el valor de 255. Y por ese numero lo vamos a dividir

In [9]:

```
#Primero convertimos los pixels en tipo float para que se puedan tener deci
males
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
#Ahora dividimos todo el dataset entre 255.
x_train /= 255
x_test /= 255
```

Definimos las estructura de nuestra red neuronal convolucional

Inicializamos nuestro modelo y revisamos las capas del mismo.

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPo
oling2D
from keras.optimizers import Adam
def base_model():

    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same',input_shape=x_train.shape[1:
]))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(512))
    model.add(Activation('relu'))
```

```
    model.add(Dropout(0.5))
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))

    optimizador = Adam(lr=0.001)

    #Utilizaremos un error de tipo crossentropy.
    model.compile(loss='categorical_crossentropy', optimizer=optimizador, m
etrics=['accuracy'])

    return model
```

In [12]:

```
cnn_n = base_model()
cnn_n.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1606144
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
_________________________________________________________________
activation_5 (Activation)    (None, 2)                 0
=================================================================
Total params: 1,663,490
Trainable params: 1,663,490
Non-trainable params: 0
_________________________________________________________________
```

Creamos la posibilidad de que se modifiquen las imagenes originales

In [14]:

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
```

```
    #Desplaza horizontalmente una fraccion del ancho de la imagen de forma
aleatoria
    width_shift_range=0.1,
    #Desplaza verticalmente una fraccion la altura de la imagen forma aleat
oria
    height_shift_range=0.1,
    #Aleatoriamente da la vuelta imagenes de forma horizontal
    horizontal_flip=True,
    #Aleatoriamente da la vuelta imagenes de forma vertical
    vertical_flip=True)

datagen.fit(x_train)
```

Entrenamos el modelo

In [15]:

```
batch_size = 100
epochs = 100
# Entrenamos el modelo por "batches" generados por data.flow().
historia = cnn_n.fit_generator(datagen.flow(x_train, y_train,
                    batch_size=batch_size),
                    epochs=epochs,
                    validation_data=(x_test, y_test),
                    workers=4,
                    shuffle=True)
```

```
Epoch 1/100
345/345 [==============================] - 113s 328ms/step - loss: 0.5019 -
acc: 0.7539 - val_loss: 0.4191 - val_acc: 0.8143
Epoch 2/100
345/345 [==============================] - 117s 339ms/step - loss: 0.4538 -
acc: 0.7881 - val_loss: 0.4063 - val_acc: 0.8167
Epoch 3/100
345/345 [==============================] - 114s 332ms/step - loss: 0.4190 -
acc: 0.8106 - val_loss: 0.3554 - val_acc: 0.8454
Epoch 4/100
345/345 [==============================] - 124s 358ms/step - loss: 0.3967 -
acc: 0.8206 - val_loss: 0.3037 - val_acc: 0.8794
Epoch 5/100
345/345 [==============================] - 118s 341ms/step - loss: 0.3784 -
acc: 0.8320 - val_loss: 0.3150 - val_acc: 0.8667
Epoch 6/100
345/345 [==============================] - 109s 317ms/step - loss: 0.3673 -
acc: 0.8378 - val_loss: 0.2942 - val_acc: 0.8838
Epoch 7/100
345/345 [==============================] - 109s 315ms/step - loss: 0.3605 -
acc: 0.8410 - val_loss: 0.2681 - val_acc: 0.8902
Epoch 8/100
345/345 [==============================] - 109s 315ms/step - loss: 0.3533 -
acc: 0.8445 - val_loss: 0.3204 - val_acc: 0.8610
Epoch 9/100
345/345 [==============================] - 109s 315ms/step - loss: 0.3483 -
acc: 0.8481 - val_loss: 0.2802 - val_acc: 0.8827
Epoch 10/100
345/345 [==============================] - 108s 313ms/step - loss: 0.3384 -
acc: 0.8515 - val_loss: 0.2455 - val_acc: 0.8995
Epoch 11/100
345/345 [==============================] - 108s 312ms/step - loss: 0.3391 -
acc: 0.8533 - val_loss: 0.2536 - val_acc: 0.8956
```

```
Epoch 92/100
345/345 [==============================] - 110s 318ms/step - loss: 0.1901 -
acc: 0.9214 - val_loss: 0.2025 - val_acc: 0.9232
Epoch 93/100
345/345 [==============================] - 111s 321ms/step - loss: 0.1949 -
acc: 0.9194 - val_loss: 0.1926 - val_acc: 0.9233
Epoch 94/100
345/345 [==============================] - 109s 317ms/step - loss: 0.1911 -
acc: 0.9230 - val_loss: 0.1841 - val_acc: 0.9296
Epoch 95/100
345/345 [==============================] - 110s 318ms/step - loss: 0.1906 -
acc: 0.9208 - val_loss: 0.1979 - val_acc: 0.9215
Epoch 96/100
345/345 [==============================] - 109s 317ms/step - loss: 0.1889 -
acc: 0.9224 - val_loss: 0.1818 - val_acc: 0.9322
Epoch 97/100
345/345 [==============================] - 109s 317ms/step - loss: 0.1876 -
acc: 0.9230 - val_loss: 0.1883 - val_acc: 0.9286
Epoch 98/100
345/345 [==============================] - 110s 318ms/step - loss: 0.1862 -
acc: 0.9234 - val_loss: 0.2138 - val_acc: 0.9198
Epoch 99/100
345/345 [==============================] - 110s 318ms/step - loss: 0.1881 -
acc: 0.9229 - val_loss: 0.1829 - val_acc: 0.9296
Epoch 100/100
345/345 [==============================] - 110s 318ms/step - loss: 0.1842 -
acc: 0.9248 - val_loss: 0.1949 - val_acc: 0.9264
```

Guardamos el modelo para poder usarlo en el futuro.

In [16]:

```
import os
save_dir = 'C:\\Users\\Alonso\\TFG\\saved_models'
model_name = 'modelo_aves_entrenado.h5'
model_path = os.path.join(save_dir, model_name)
cnn_n.save(model_path)
print('Saved trained model at %s ' % model_path)
```

```
Saved trained model at C:\Users\Alonso\TFG\saved_models\newmodel_new.h5
```

Mostar la perdida del modelo con graficas.

In [22]:

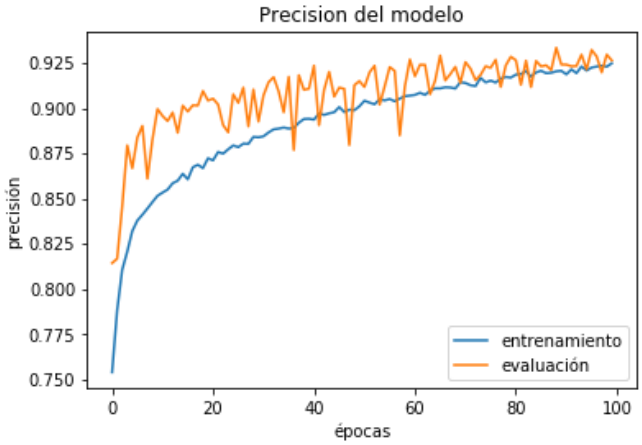
```
print(historia.history.keys())
```

```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

In [47]:

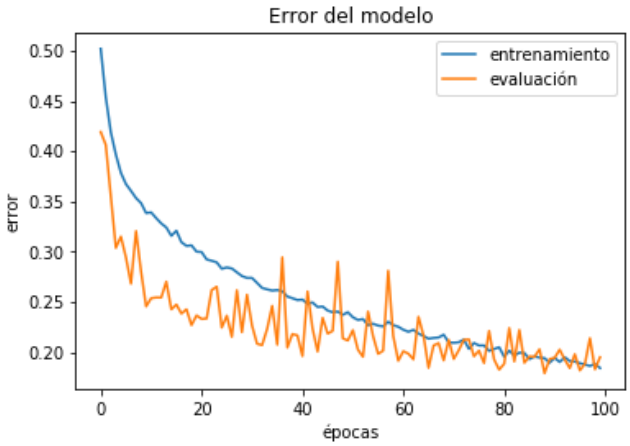
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(historia.history['acc'])
plt.plot(historia.history['val_acc'])
plt.title('Precision del modelo')
plt.ylabel('precisión')
plt.xlabel('épocas')
```

```
plt.legend(['entrenamiento', 'evaluación'],
           loc='lower right')
plt.show()
```



In [46]:

```
plt.plot(historia.history['loss'])
plt.plot(historia.history['val_loss'])
plt.title('Error del modelo')
plt.ylabel('error')
plt.xlabel('épocas')
plt.legend(['entrenamiento', 'evaluación'],
           loc='upper right')
plt.show()
```



Evaluación del modelo

Comprobamos como ha funcionado el modelo con la particion test del dataset.

In [50]:

```
scores = cnn_n.evaluate(x_test, y_test)
print('Error evaluación:', scores[0])
```

```
print('Precision evaluacion:', scores[1]*100)
```

```
16502/16502 [==============================] - 13s 816us/step
Error evaluación: 0.1949448962143379
Precisión evaluación: 92.63725608800371
```

In [21]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test[:,1],cnn_n.predict_classes(x_test))
```

Out[21]:

```
array([[9903,  598],
       [ 617, 5384]], dtype=int64)
```

Mostramos imagenes de aves que son erroneas

In [234]:

```
etiquetas  = y_test[:,1]
```

In [235]:

```
predicciones = cnn_n.predict_proba(x_test)
```

In [277]:

```
#Para calcular las que no son aves y dice que lo son.
posiciones_ave=[]
for i in range(len(brr)-1):
    if(etiquetas[i] == 0):
        if(predicciones[i][1] > 0.95):
            posiciones_ave.append(i)
```

In [300]:

```
#Para calcular las que son aves y dice que no lo son.
posiciones_noave=[]
for i in range(len(brr)-1):
    if(etiquetas[i] == 1):
        if(predicciones[i][0] > 0.95):
            posiciones_noave.append(i)
```

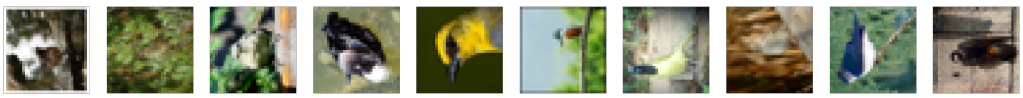
In [302]:

```
import random
posiciones = random.sample(posiciones_ave,k=10)

f, axarr = plt.subplots(1,10,figsize=(32, 32))
for i in range (10):
    axarr[i].axis('off')
    axarr[i].imshow(X[posiciones[i]])
```

Out[302]:

```
[array([[0.9879951 , 0.01200487]], dtype=float32),
 array([[0.9657448 , 0.03425525]], dtype=float32),
 array([[0.97129875, 0.0287013 ]], dtype=float32),
 array([[0.9830647, 0.0169353]], dtype=float32),
 array([[0.98184055, 0.01815947]], dtype=float32),
 array([[0.9556826 , 0.0443173 ]], dtype=float32)
```

```
array([[0.9556826 , 0.04431739]], dtype=float32),
array([[0.9655276 , 0.03447243]], dtype=float32),
array([[0.9985991 , 0.00140092]], dtype=float32),
array([[0.9955552, 0.0044448]], dtype=float32),
array([[0.9987495 , 0.00125046]], dtype=float32)]
```



Predicciones.

In [25]:

```
#Ahora multiplicamos todo el dataset entre 255.
X = x_test * 255
#Primero convertimos los pixels en tipo float para que se puedan tener deci
males
X = X.astype('int64')
X = X.astype('uint8')
```

In [193]:

```
predicciones = cnn_n.predict_proba(x_test)
```

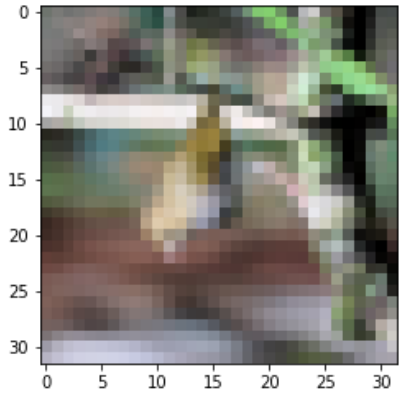
In [216]:

```
import random
import matplotlib.pyplot as plt
%matplotlib inline
num = random.randint(0,len(x_test))
if(y_test[num][1] == 1):
    print("Es un ave!")
else:
    print("No es un ave")
plt.imshow(X[num])
```

```
Es un ave!
```

Out[216]:

```
<matplotlib.image.AxesImage at 0x157ad0b9d30>
```



In [225]:

```
labels = [u'No ave ('+str("{0:.2f}".format(predicciones[num][0]*100))+'%)',
          u'Ave ('+str("{0:.2f}".format(predicciones[num][1]*100))+'%)']
plt.pie(predicciones[num], labels = labels,explode=[0, 0.1])
```

Out[225]:

```
([<matplotlib.patches.Wedge at 0x157acbcef28>,
  <matplotlib.patches.Wedge at 0x157acbd2438>],
 [Text(0.730534,0.822386,'No ave (26.88%)'),
  Text(-0.796947,-0.897149,'Ave (73.12%)')])
```