ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

Trabajo de Fin de Grado

RECONOCIMIENTO DE IMÁGENES MEDIANTE REDES NEURONALES CONVOLUCIONALES

Grado en Ingeniería del Software

Autor: Javier Martínez Llamas
Tutor: Luis Fernando de Mingo

" Todo nuestro conocimiento empieza por los sentidos; de aquí pasa al entendimiento, y termina en la razón. Sobre ésta no hay nada más alto en nosotros para elaborar la materia de la intuición y ponerla bajo la suprema unidad del pensamiento."

Immanuel Kant

Resumen

Este proyecto presenta un estudio e implementación de una red neuronal convolucional que permita identificar y reconocer especímenes de ballena jorobada a partir de los patrones únicos de sus colas.

Partiendo de un *dataset* compuesto de imágenes de colas de ballenas se detallan todas las fases del proceso de creación y entrenamiento de una red neuronal. Desde el análisis y preprocesado de las imágenes a la elaboración de predicciones, utilizando TensorFlow y Keras como vehículo.

Así mismo se explican otras posibles alternativas a la hora de afrontar este problema y la problemática surgida durante el proceso de elaboración del proyecto.

Abstract

This project presents a study and implementation of a convolutional neural network to identify and recognize humpback whale specimens from the unique patterns of their tails.

Starting from a dataset composed of images of whale tails, all the phases of the process of creation and training of a neural network are detailed. From the analysis and pre-processing of images to the elaboration of predictions, using TensorFlow and Keras as frameworks.

Other possible alternatives are also explained when it comes to tackling this problem and the complications that have arisen during the process of developing this project.

Índice general

Índice de figuras

Capítulo 1

Introducción

La Inteligencia Artificial (IA) ha visto en los últimos años como, a medida que la capacidad de cómputo se incrementaba y se hacía más accesible al gran público, su impacto tecnológico y social crecían exponencialmente.

Esto ha propiciado su inclusión en prácticamente cualquier campo de la informática y que gigantes tecnológicos como Google, Facebook, Amazon o Microsoft estén ofreciendo servicios, soluciones y herramientas basadas en la IA. Con aplicaciones tan variadas como pueden ser videojuegos, asistentes virtuales, servicios financieros o comportamientos autónomos entre otros.

Una de las áreas que ha experimentado un mayor avance y desarrollo es el reconocimiento de imágenes. Esto es mayoritariamente debido a su gran utilidad social y a la mejora en tiempo y precisión que su desempeño ofrece comparado con el de los humanos, desde sistemas de vigilancia y reconocimiento facial a aplicaciones médicas como la identificación de tumores.

Tal es su potencial e implicación social que su desarrollo debe estar estrechamente ligado a la responsabilidad ética. No exenta de controversia, la inteligencia artificial, así como el reconocimiento de imágenes se han visto sometidos a críticas por parte de la población debido a prácticas abusivas o a su falta de ética y privacidad.

1.1 Objetivos

El objetivo detrás de este proyecto reside en el estudio e implementación de una red neuronal convolucional que permita reconocer y clasificar un espécimen de ballena jorobada. Por tanto se centrará en la consecución de los siguientes puntos:

El estudio de los distintos enfoques y técnicas actuales en lo relativo a la inteligencia artificial, concretamente en el campo del aprendizaje profundo o *deep learning* y cómo puede ser aplicado a nuestro proyecto.

La comprensión de las bibliotecas y tecnologías implementadas durante el desarrollo de la aplicación como son TensorFlow y Keras.

Capítulo 2

Inteligencia Artificial

2.1 ¿Qué es la Inteligencia Artificial?

Desde la irrupción de la informática la posibilidad de máquinas con capacidad de pensamiento y raciocinio ha resultado atractiva, imaginada por escritores y artistas se han planteado androides indistinguibles de humanos e inteligencias artificiales con capacidades sobrehumanas y acceso a infinitud de conocimiento. Sin embargo para comprender la viabilidad de estos supuestos es necesario entender que es la inteligencia artificial y como funciona.

El término inteligencia artificial se ha planteado históricamente desde distintos puntos de vista: la capacidad de pensamiento o la habilidad de actuar inteligentemente [7]. Centrándose el primero en la aproximación a una idea humana de inteligencia, en la que las máquinas piensen y sean racionales. El segundo planteamiento se basa no tanto en el proceso como en el resultado, considerando inteligencia artificial a la capacidad de actuar y emular lo que sería resultado de una acción estrictamente racional.

2.2 Aprendizaje Automático y Deep Learning

Parte fundamental de la inteligencia reside en el aprendizaje, proceso por el cual mediante información, estudio y experiencia se logra una determinada formación. Es por ello que surge la necesidad dentro del marco de la IA de dotar a los sistemas de conocimiento.

Con este objetivo nació el aprendizaje automático o *machine learning*, que, gracias al procesado de datos busca la identificación de patrones comunes que permitan la elaboración de predicciones cada vez más perfeccionadas. Sin embargo estos algoritmos han requerido históricamente de un conocimiento estadístico complejo.

Siguiendo la evolución del *machine learning* surge en los últimos años el conocido como *deep learning* o aprendizaje profundo. Este, a diferencia del aprendizaje automático, entiende el mundo como una jerarquía de conceptos [1]. Diluyendo la información en diferentes capas mediante el uso de módulos, que transforman su representación en un nivel más alto y abstracto. Esto permite la amplificación de la información relevante y eliminar la superflua [6].

2.3 Redes Neuronales

Las redes neuronales artificiales o *neural networks* son modelos matemáticos que tratan de emular el comportamiento natural de las redes neuronales biológicas.

Se establece una red de unidades lógicas o neuronas interconectadas entre sí que procesan la información recibida y emiten un resultado a la siguiente capa determinado por una función de activación que tiene en cuenta el peso de cada entrada, dotando así de mayor importancia a conexiones entrantes concretas.

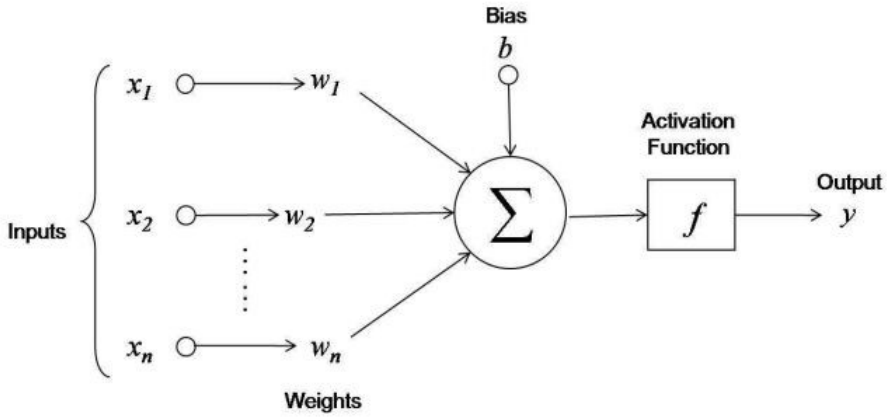


Figura 2.1: Esquema Neurona Artificial

En este modelo, la salida neuronal y vendría dada por:

$$y = f(\sum_{1}^{n} w_n x_n + b)$$

Durante la fase de entrenamiento de una red neuronal los parámetros peso y *bias* son reajustados con el fin de adaptar el modelo a una tarea concreta y mejorar las predicciones. La función de activación f será seleccionada de acuerdo al problema que se pretende resolver.

Las redes neuronales multicapa organizan y agrupan las neuronas artificiales en niveles o capas. Cuentan con una capa de entrada y otra de salida y entre ellas puede contar con un número indefinido de capas ocultas o *hidden layers*.

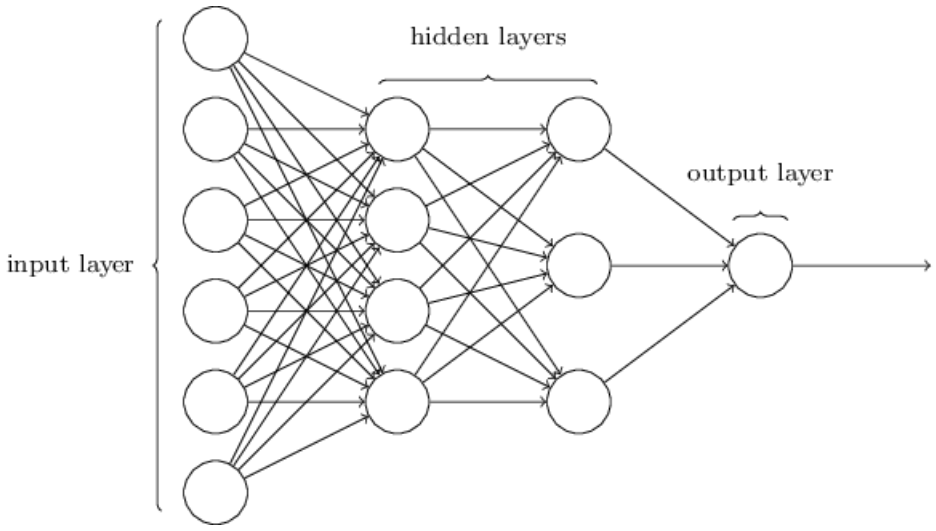


Figura 2.2: Esquema Perceptrón Multicapa

La capa de entrada está formada por neuronas que introducen a la red la información, en ellas sin embargo no se produce procesamiento, por lo que únicamente actúan como receptor y propagador. Las capas ocultas están formadas por aquellas neuronas que tanto la entrada como la salida conectan con otras capas de neuronas. La capa de salida es el último nivel de la red y produce los resultados de esta.

Las conexiones de las redes neuronales multicapa suelen dirigirse hacia adelante, conectando las neuronas con su siguiente capa. Reciben el nombre de redes *feedforward* o redes alimentadas hacia adelante.

2.4 Redes Neuronales Convolucionales

Las redes neuronales convolucionales o *convolutional neural network* (ConvNet o CNNs) son una clase de redes neuronales multicapa *feedforward* especialmente diseñadas para el reconocimiento y clasificación de imágenes.

Los ordenadores perciben las imágenes de diferente forma a los humanos, para estos una imagen consiste en un vector bidimensional con los valores relativos a los píxeles. Contando con un canal para imágenes en escala de grises o tres para el color (RGB).

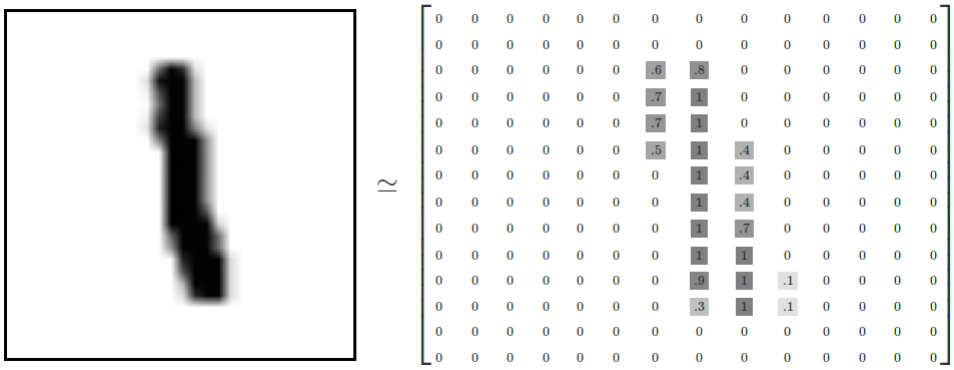


Figura 2.3: Vector Imagen de MNIST

Las redes convolucionales siguen una determinada estructura, existiendo principalmente tres tipos de capas: *Convolutional Layer*, *Pooling Layer* y *Fully-Connected Layer*. Se efectúan una serie de convoluciones y submuestreos o reducciones alternadas hasta que finalmente mediante una serie de capas completamente conectadas (perceptrón multicapa) se obtiene la salida deseada, equivalente al número de clases.

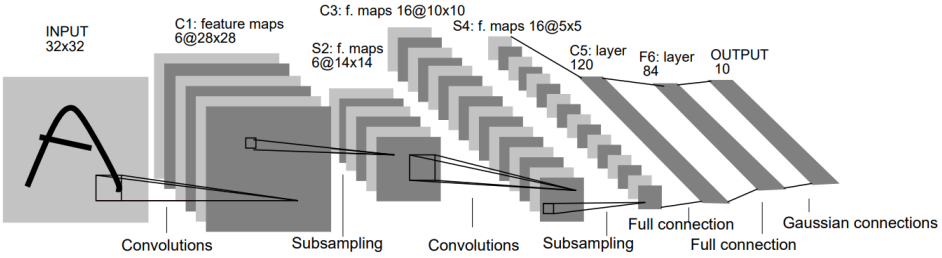


Figura 2.4: Arquitectura LeNet-5

La convolución o *convolution* efectúa una serie de productos y sumas entre la matriz de partida y una matriz *kernel* o filtro de tamaño n. Por otra parte el submuestreo reduce la dimensión de la matriz de entrada dividiéndola en subregiones y permitiendo generalizar las características.

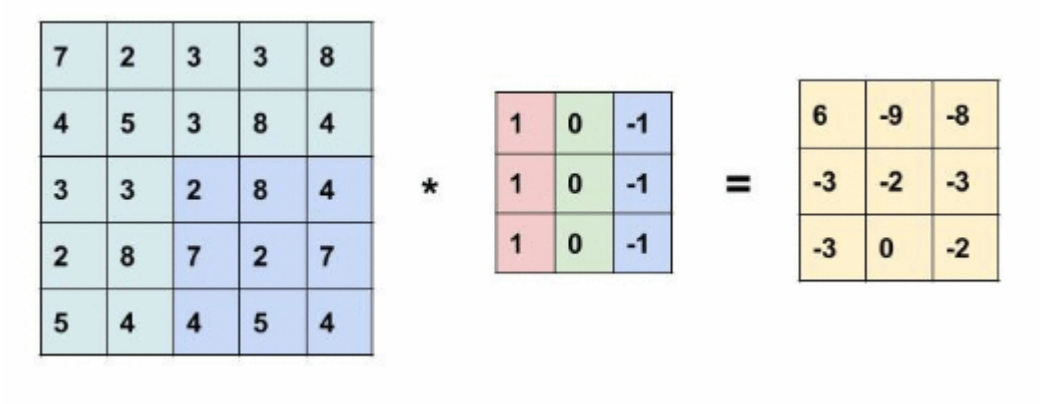


Figura 2.5: Convolución

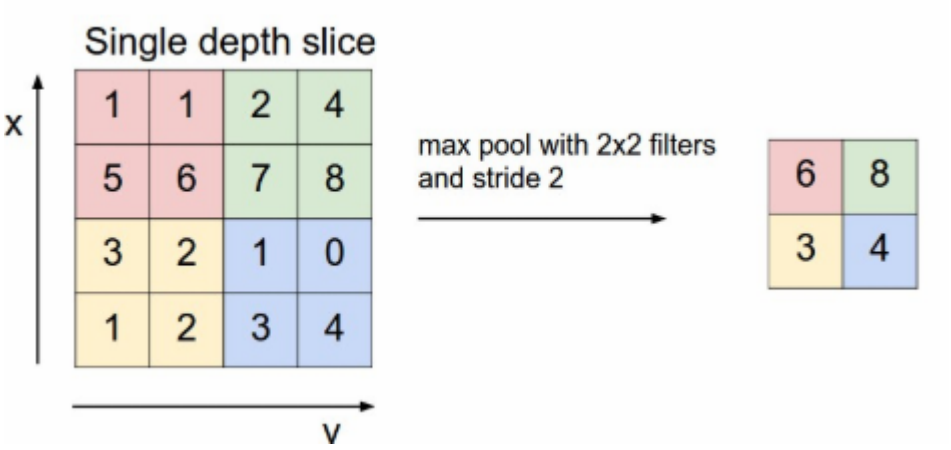


Figura 2.6: Max Pooling

Actualmente existen distintas arquitecturas que son consideradas como estado del arte como pueden ser *AlexNet*, *Inception* o *VGGNet*, destacando entre ellas las redes residuales, *residual neural networks* o ResNet [2]

Capítulo 3

Tecnologías y Herramientas

Una vez comprendidos los conceptos que comprende el reconocimiento de imágenes y las redes neuronales es necesario elegir las herramientas y tecnologías que se utilizarán durante el desarrollo.

3.1 Dataset

Parte indispensable del aprendizaje automático es la información que se pretende analizar pues en ella se basa todo el desarrollo posterior. Para nuestro caso de estudio se ha optado por la comunidad de Kaggle, enfocada al *machine learning* y que cuenta con numerosos *datasets*.

En concreto se ha seleccionado una colección de imágenes de ballenas jorobadas [4], comprendida en una de sus competiciones. Se pretende reconocer y clasificar mediante el estudio de los patrones en su cola individuos de esta familia de cetáceos.

3.2 Lenguaje de Programación

Entre los lenguajes de programación más utilizados en el campo de la inteligencia artificial destacan especialmente dos, Python y R. Este último orientado en gran medida al análisis estadístico.

En lo relativo a *deep learning* el claro dominador es Python, en parte gracias al amplio número de bibliotecas y *frameworks* desarrollados para este lenguaje como pueden ser PyTorch, Caffe, Theano, TensorFlow o Keras. Es por ello que se ha optado por Python en su versión 3.6 para este proyecto.

Además se emplearán distintas bibliotecas implementadas para este lenguaje con el objetivo de facilitar el proceso de desarrollo. Algunas de los paquetes más relevantes son los siguientes:

NumPy, enfocada a la computación científica, aporta vectores o *arrays* así como potentes herramientas matemáticas sobre ellos.

Pandas para el análisis de datos, principalmente la lectura de los distintos archivos *csv* con los que se trabaja. Permitiendo el rápido acceso y lectura de estos así como el potente tratamiento de sus datos.

Scikit-learn, una biblioteca de *machine learning*, con potentes herramientas estadísticas que usaremos principalmente durante el preprocesado de los datos, previo a la implementación de la red neuronal.

Python Imaging Library (PIL) empleada para la lectura y conversión de las imágenes.

En la fase de desarrollo, aunque eliminada en la versión final, se ha utilizado la biblioteca Matplotlib que permite la creación de gráficos. En concreto se ha utilizado únicamente para la visualización y obtención de imágenes, originales y procesadas, durante la elaboración este documento. Por tanto carece de relevancia y utilidad en las versiones finales.

3.3 TensorFlow

Desarrollada por Google con el fin de satisfacer sus necesidades en el ámbito del *machine learning*, TensorFlow es una biblioteca para cálculos numéricos mediante el uso de diagramas de flujos de datos. Publicada bajo licencia de código abierto en 2015 se ha situado desde entonces como uno de los referentes en el desarrollo de sistemas de aprendizaje profundo y redes neuronales.

3.4 Keras

Keras nace con la flexibilidad y usabilidad en mente, se trata de una biblioteca para redes neuronales de alto nivel desarrollada para Python que busca simplificar en gran medida su desarrollo. Permite ejecutarse sobre bibliotecas como TensoFlow, Theano o CNTK como *backend*, entendiéndose más como una interfaz más que como un *framework*.

En 2017 Keras fue integrado en el código fuente de TensorFlow permitiendo su desarrollo con un nivel mayor de abstracción.

3.5 CUDA y cuDNN

El *deep learning* y su desarrollo se han visto en gran medida facilitados por el uso de GPUs (*Graphics Processing Unit*). El alto número de cálculos que se efectúan y su elevada complejidad, especialmente durante el entrenamiento de una red neuronal, hacen que sea necesario un *hardware* de altas prestaciones.

Es en este escenario donde entran en juego las GPUs. Utilizadas en el entrenamiento de las redes neuronales permiten reducir considerablemente los tiempos en comparación al uso exclusivo de CPUs. Esto es debido a su alto número de núcleos que permiten el procesamiento en paralelo de multitud de operaciones.

CUDA, la arquitectura de cálculo paralelo de NVIDIA, junto a cuDNN, su librería para redes neuronales profundas, permiten la utilización de GPUs en TensorFlow para nuestro proyecto.

3.6 Hardware

El rendimiento de la aplicación puede variar considerablemente en función de las especificaciones disponibles, siendo crítico en el los resultados finales y, especialmente, en el tiempo de ejecución.

El sistema utilizado cuenta con un procesador *Intel Core i7-8700* de 6 núcleos con una frecuencia base de *3,2GHz* hasta *4,6GHz* y *16GB* de memoria *RAM DDR4*. Además cuenta con una tarjeta gráfica de NVIDIA *GeForce GTX 1060* de *3GB* de memoria *VRAM DDR5* con un total de 1152 núcleos CUDA.

Este *hardware* utilizado es capaz de realizar el entrenamiento de la red neuronal y el procesado de imágenes de forma holgada, sin embargo distintas aproximaciones al problema, como la mencionada en la sección 4.7, requerirían de mayor memoria en la GPU.

Capítulo 4

Diseño

4.1 Estudio del Dataset

Aspecto clave para afrontar la construcción de una red neuronal es el estudio y análisis del *dataset* sobre el que se pretende trabajar. En este proyecto el *dataset* está constituido por un conjunto de imágenes de ballenas jorobadas, más concretamente sus colas.

Figura 4.1: Imagen 1a36b244

Figura 4.2: Imagen 1c5d333f

Figura 4.3: Imagen 1efa630a

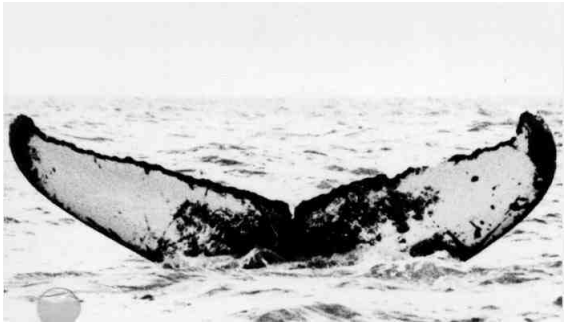


Figura 4.4: Imagen 2c77045a

Como se puede apreciar las imágenes presentas diferentes características. La diferencia más notable *a priori* es la variación de color entre ellas, pudiendo presentarse en escala de grises o a color. También es relevante el contraste de resoluciones o la presencia de elementos externos a la propia ballena como son las anotaciones.

En total hay aproximadamente 25.000 imágenes divididas en dos conjuntos, uno de entrenamiento de 9851 imágenes y un otro de evaluación de 15.600 imágenes. Esta distribución dificulta el posterior entrenamiento de la red al ser el tamaño del conjunto de entrenamiento la mitad del de evaluación.

El conjunto de entrenamiento proporciona junto con las imágenes un archivo *csv* que recoge las etiquetas o *labels* de cada imagen.

	Image	Id
0	00022e1a.jpg	w_e15442c
1	000466c4.jpg	w_1287fbc
2	00087b01.jpg	w_da2efe0
3	001296d5.jpg	w_19e5482
4	0014cfdf.jpg	w_f22f3e3

Figura 4.5: Cabecera Archivo Entrenamiento

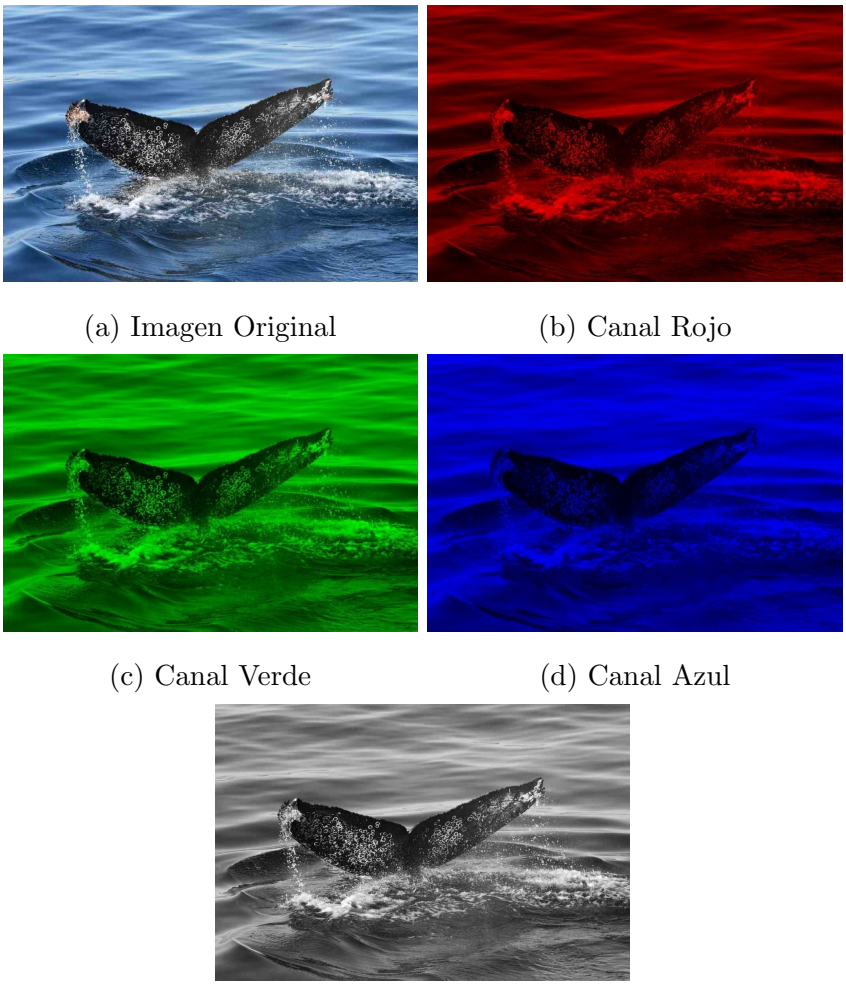
Nuestra red neuronal tratará el siguiente problema: Identificar los especímenes de ballena jorobada y en caso de no haber registro de ella se catalogará como nueva ballena con la etiqueta *new_whale*, contando en total con 4251 clases o individuos diferentes.

4.2 Procesado de Imágenes

Con el correcto funcionamiento de la red en mente es necesario procesar con anterioridad cada imagen con el fin de facilitar la extracción de las características o *features* presentes en esta. Homogeneizando todos los datos se consigue este efecto.

En primer lugar se procede a convertir la imagen a una escala de grises, pasando de tres canales de color a uno solo.

Esto es debido a dos motivos, la existencia de imágenes originales en el *dataset* en blanco y negro y a la ausencia de características e información en el color, únicamente siendo de utilidad el patrón de la cola.



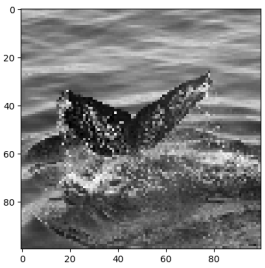
(a) Imagen Original (b) Canal Rojo

(c) Canal Verde (d) Canal Azul

(e) Escala de Grises

Figura 4.6: Imagen RGB a Grises

La red neuronal necesita que la dimensión de los vectores de entrada sea fija, es por ello que cada imagen debe ser reescalada con anterioridad. Dando como resultado en este caso matrices de tamaño 100x100 con un sólo canal.



(a) Imagen Reescalada

```
[[182 179 179 ... 102 101 104]
 [202 202 195 ...  95  99 102]
 [172 175 179 ... 114 118 122]
 ...
 [ 64  64  64 ...  95 100 126]
 [ 64  64  64 ...  30  81 114]
 [ 68  72  68 ... 135 106 151]]
```

(b) Representación Matricial

Figura 4.7: Reescalado de Imagen

Cada píxel tiene un valor comprendido entre 0 y 255, esta amplitud de rango, debido al funcionamiento de las redes convolucionales, posibilita la incorrecta identificación de las características para cada vector. Para corregir esto es conveniente normalizar previamente la imagen, en un proceso conocido como *zero mean and unit variance normalization*.

El primer paso es centrar la imagen en el valor 0 restando a cada valor de la matriz su media.

$$M = \begin{Bmatrix} 67{,}8595 & 64{,}8595 & \dots & -13{,}140503 & -10{,}140503 \\ 87{,}8595 & 87{,}8595 & \dots & -15{,}140503 & -12{,}140503 \\ \dots & & & & \\ -50{,}140503 & -50{,}140503 & \dots & -33{,}140503 & -0{,}14050293 \\ -46{,}140503 & -42{,}140503 & \dots & -8{,}140503 & 36{,}859497 \end{Bmatrix}$$

Posteriormente se comprime el rango dividiendo cada valor entre la desviación estándar de la matriz.

$$M = \begin{Bmatrix} 1{,}4328924 & 1{,}3695457 & ... & -0{,}2774693 & -0{,}21412258 \\ 1{,}855204 & 1{,}855204 & ... & -0{,}31970045 & -0{,}25635374 \\ ... & & & & \\ -1{,}0587456 & -1{,}0587456 & ... & -0{,}6997808 & -0{,}0029668 \\ -0{,}97428334 & -0{,}88982105 & ... & -0{,}17189142 & 0{,}7783096 \end{Bmatrix}$$

4.2.1 Data Augmentation

Práctica común dentro de la clasificación de imágenes mediante redes de neuronas es el aumento de los datos presentes en el *dataset*. Especialmente útil donde la proporción de clases no está balanceada y existen numerosas categorías con una única muestra.

Para aumentar el número de datos disponibles para el entrenamiento se realizan una serie de procesados sobre una imagen original y así generar una serie de imágenes derivadas. Entre las posibles modificaciones existentes se encuentran la rotación, traslación, ruido, etc.

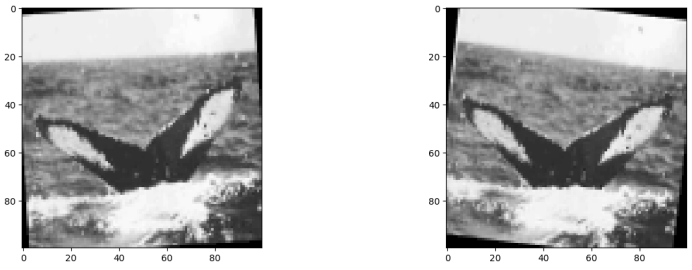


Figura 4.8: Rotación de Imágenes

La eficacia de esta técnica reside en la forma en que las redes neuronales entienden las imágenes y sus características. Si una imagen es modificada ligeramente es percibida por la red como una imagen completamente distinta perteneciente a la misma clase.

Esto disminuye las probabilidades de que la red se centre en orientaciones o posiciones mientras mantiene la relevancia de las características deseadas como puede ser el patrón de la cola en este proyecto.

Se ha realizado un aumento del conjunto de entrenamiento siguiendo el siguiente criterio, si una clase cuenta con menos de 10 imágenes se genera la diferencia con su número original de muestra. De tal manera que si un espécimen contará con 4 muestras se generarían 6 adicionales a partir de las imágenes asociadas a una ballena.

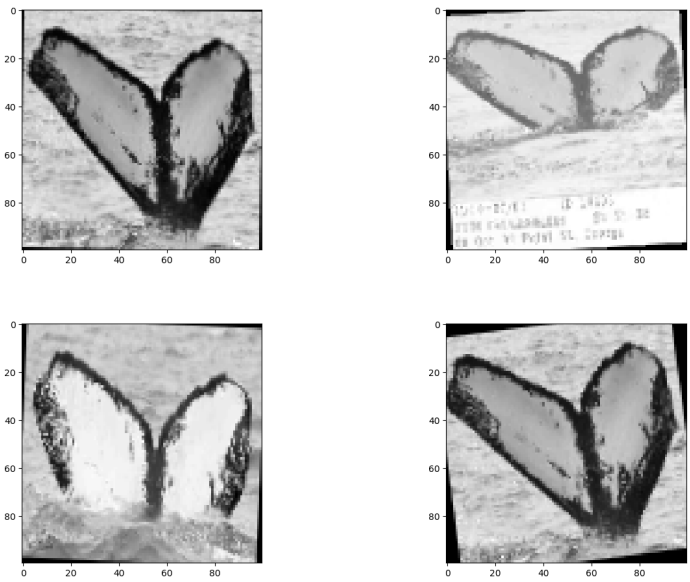


Figura 4.9: Aumento sobre el espécimen w_964c1b3

4.3 Arquitectura de la Red Neuronal

La red implementada cuenta con la siguiente estructura:

Layer (type)	Output Shape	Param #
conv0 (Conv2D)	(None, 94, 94, 32)	1600
bn (BatchNormalization)	(None, 94, 94, 32)	128
activation (Activation)	(None, 94, 94, 32)	0
max_pool (MaxPooling2D)	(None, 47, 47, 32)	0
conv1 (Conv2D)	(None, 45, 45, 64)	18496
activation_1 (Activation)	(None, 45, 45, 64)	0
avg_pool (AveragePooling2D)	(None, 15, 15, 64)	0
flatten (Flatten)	(None, 14400)	0
ReLU (Dense)	(None, 450)	6480450
dropout (Dropout)	(None, 450)	0
softmax (Dense)	(None, 4251)	1917201

Total params: 8,417,875
Trainable params: 8,417,811
Non-trainable params: 64

Figura 4.10: Arquitectura de la Red Implementada

Se establece una capa convolucional seguida de una normalización del *batch* [3] y una reducción *max pool* como la ejemplificada en la figura 2.5. A continuación se define otra capa convolucional seguida de una reducción mediante la media o *average pooling*. Esto permite en primer lugar extraer las características más importantes de la imagen y posteriormente en el siguiente nivel de profundidad suavizar la extracción para no perder información relevante.

En ambas capas se utiliza una función de activación ReLU (*rectified linear unit* o unidad lineal rectificada) definida por:

$$f(x) = x^+ = \text{máx}(0, x)$$

Donde x es la entrada de la neurona, retornando el valor x si es positivo o 0 si es negativo. Esto permite acelerar considerablemente el entrenamiento al ser fácilmente computable en comparación a otras funciones de activación como puede ser *softmax*.

Finalmente se conecta una red neuronal multicapa convencional compuesta por una red densa completamente conectada de 450 neuronas con *dropout* o abandono y finalmente una capa densa de salida completamente conectada con *softmax* como función de activación y 4251 neuronas, equivalentes al número total de clases a clasificar. La primera capa recibe como entrada un vector de una dimensión, para ello se comprime la salida de dos dimensiones obtenida de las capas convolucionales.

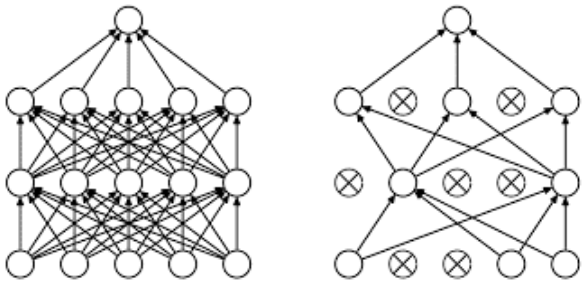


Figura 4.11: Esquema Dropout

El *dropout* permite ignorar un porcentaje de las unidades o neuronas de entrada durante la dase de entrenamiento, en este caso un 80 %. Esto da lugar a una red neuronal más pequeña que la original y por tanto menos parámetros, lo que disminuye las dependencias entre neuronas y evita así el sobreentrenamiento.

La función de activación *softmax* es utilizada en problemas de clasificación con múltiples opciones como es este caso. Devuelve las probabilidades de cada clase y viene dada por:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{4251} e^{x_j}}$$

Comprime los valores entre 0 y 1 y hace que la suma de todos los valores resultantes sea igual a 1. Siendo x un vector de entrada de tamaño igual al número de unidades o neuronas de la capa (4251).

4.4 Entrenamiento

Durante esta sección se utilizará la herramienta TensorBoard de TensorFlow para el seguimiento y visualización de nuestra red neuronal durante la fase de entrenamiento. Realizando una simulación y obteniendo las gráficas de precisión y pérdida.

Aspecto clave de las redes de neuronas es la función de pérdida o *loss function*. Esta permite saber cómo de alejadas están las predicciones de un modelo ($\hat{y}$) de sus etiquetas reales y. Se trata de un valor positivo que a la vez que decrece mejora el rendimiento del modelo.

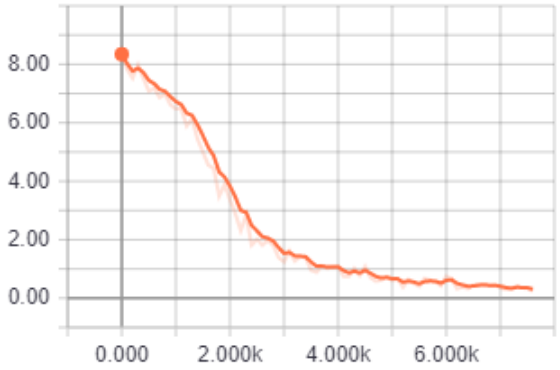


Figura 4.12: Pérdida durante el entrenamiento

En el modelo implementado la función loss utilizada es *Categorical Cross-Entropy* [9], especialmente útil en problemas donde hay múltiples clases excluyentes. La función viene dada por:

$$H(y, \hat{y}) = \sum_x y_x \log \frac{1}{\hat{y_x}} = -\sum_x y_x \log \hat{y_x}$$

Donde x es una variable discreta e $\hat{y}$ la predicción para la distribución real y.

La precisión del modelo durante el entrenamiento aumenta a medida que el valor de la función pérdida disminuye.

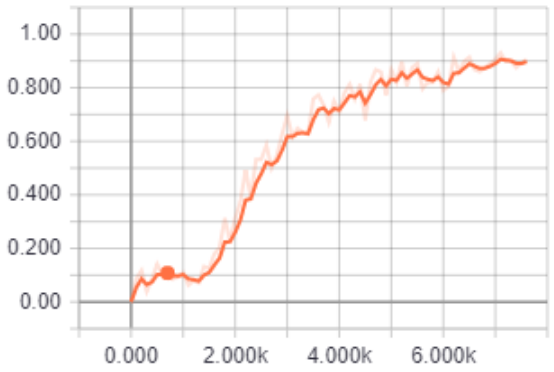


Figura 4.13: Precisión durante el Entrenamiento

La simulación del entrenamiento realizada toma como referencia la salida de la primera capa densa antes de proceder a la clasificación. Esto no permite una exacta representación de lo que sería la salida final con las clases totales pero ejemplifica con facilidad y similaridad el comportamiento de la red neuronal y los datos durante la fase de entrenamiento.

En el caso de estudio se diferencian cuatro situaciones durante la ejecución. El comienzo donde la red no está entrenada (figura 4.8). Las primeras iteraciones (figura 4.9) donde se considera la mayoría como *new_whale* debido a sus características comunes entre todas las imágenes y a su mayor proporción en comparación a las otras clases. Se empiezan a diferenciar *features* de las demás clases (figura 4.10). La red finaliza su entrenamiento y diferencia con mayor precisión (figura 4.11). Se ha seleccionado un grupo con características similares para seguir su agrupamiento.

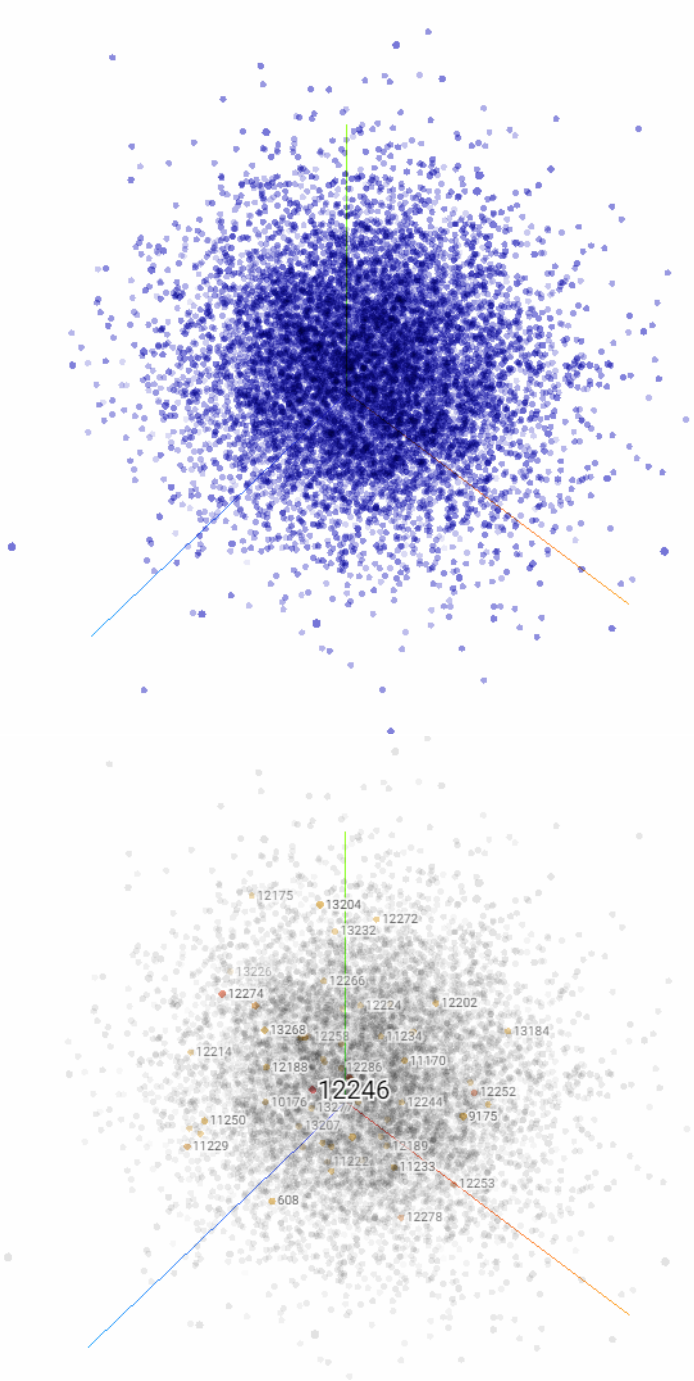


Figura 4.14: Comienzo de la Simulación

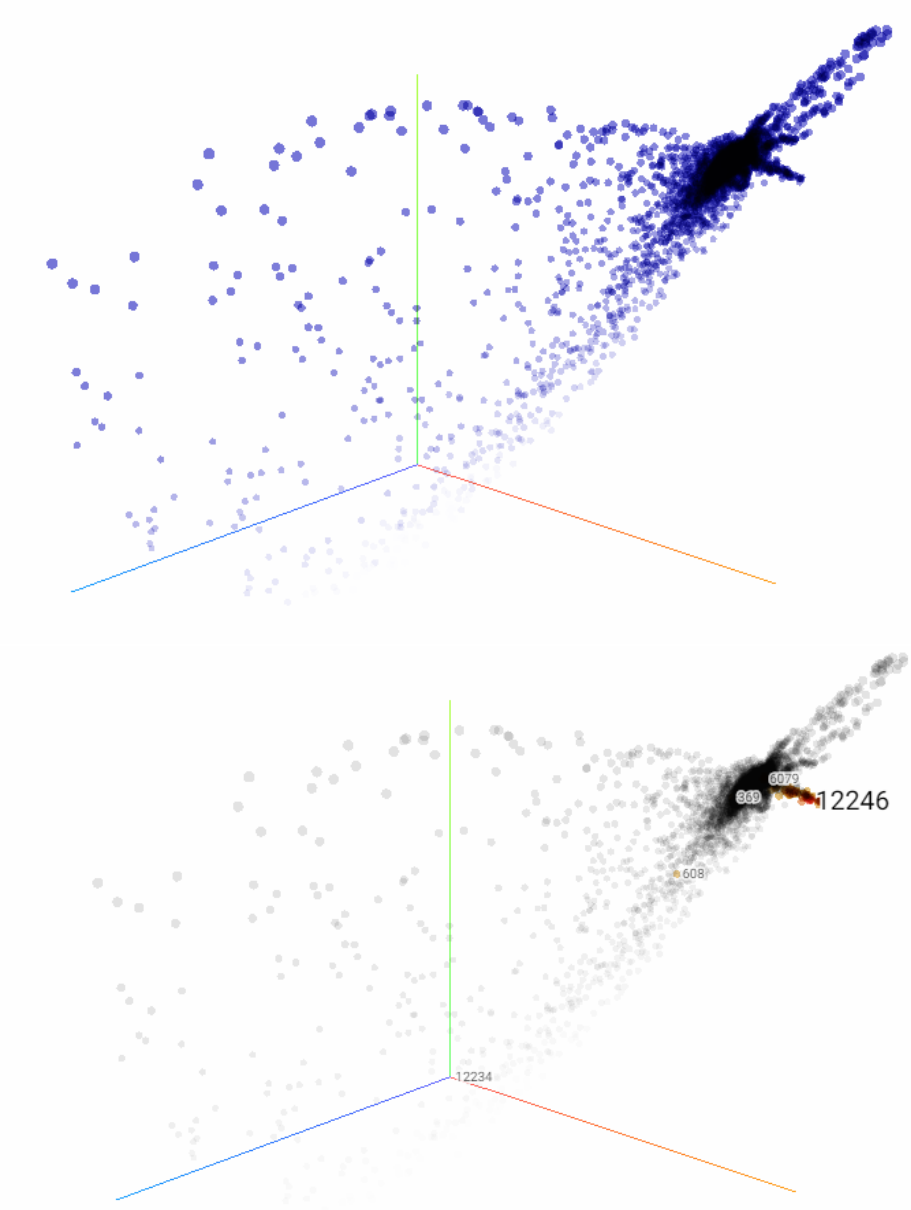


Figura 4.15: Primeras iteraciones de la Simulación

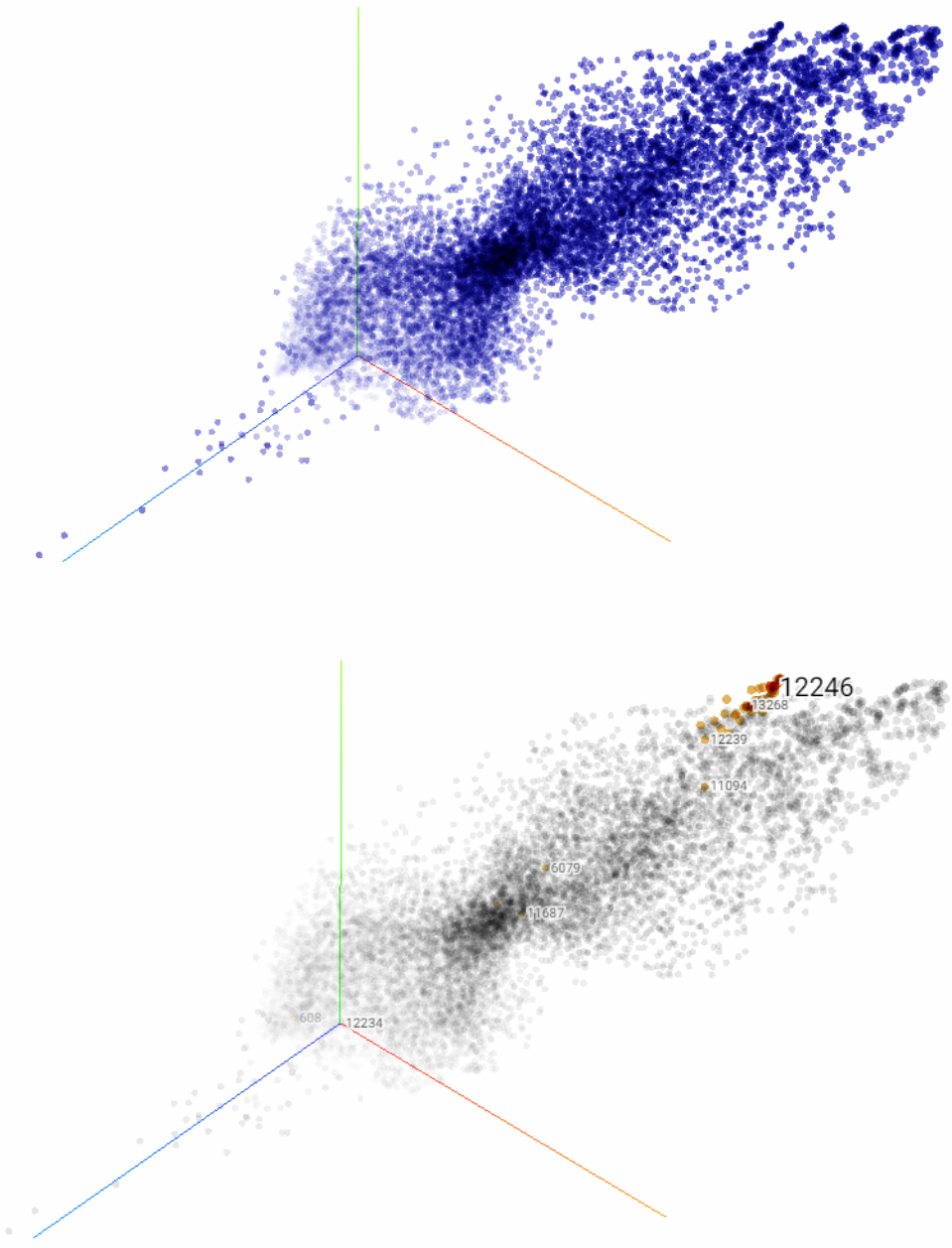


Figura 4.16: Iteraciones avanzadas en la Simulación

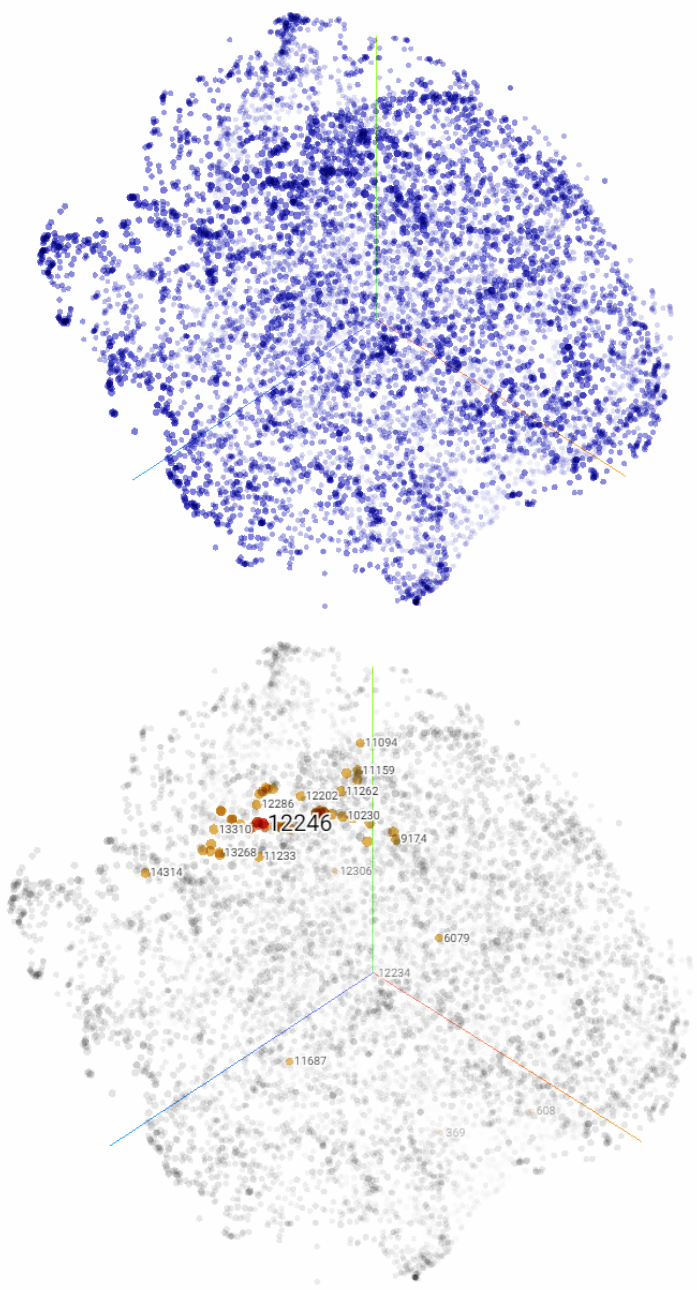


Figura 4.17: Final de la Simulación

4.4.1 Algoritmo de Optimización

El proceso de aprendizaje se resume en una idea: la minimización de la función de pérdida mediante la actualización de los parámetros, w, de la red. Convirtiéndolo en un problema de optimización.

Adam o *Adaptive Moment Estimation* [5] es un algoritmo de optimización que permite modificar el ritmo de aprendizaje o *learning rate* a medida que se desarrolla el entrenamiento. En concreto se trata de una variante del descenso de gradiente estocástico o *stochastic gradient descent*.

La red neuronal cuenta con una serie de hiperparámetros entre los que se encuentra la tasa de aprendizaje. Este parámetro permite en los algoritmos de descenso de gradiente determinar el siguiente punto tal que:

$$w_j = w_j - \alpha \frac{\partial f(w_j)}{\partial w_j}$$

Donde w_j es uno de los parámetros, f la función de costo o pérdida y α la tasa de aprendizaje.

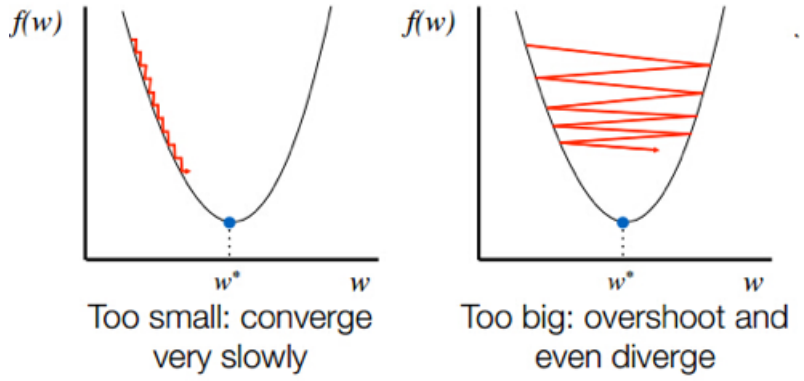


Figura 4.18: Esquema Tasa de Aprendizaje

Si su valor es demasiado pequeño la convergencia, y por tanto el aprendizaje, llevarán demasiado tiempo, de la misma forma si la tasa es suficientemente grande el siguiente punto se excederá del mínimo haciendo imposible alcanzarlo. Es por tanto que un valor adecuado del *learning rate* es clave para el correcto desempeño durante el entrenamiento.

El descenso del gradiente es un proceso iterativo donde los pesos se actualizan en cada iteración, es por ello que para obtener el mejor resultado es necesario procesar el *dataset* más de una vez. Debido a que procesar un dataset completo en una iteración es complicado por restricciones de memoria este es dividido en partes llamadas *batch* de un determinado tamaño, si el *dataset* contiene 1000 imágenes y se establece un tamaño de *batch* de 100, en total se tendrán 10 divisiones.

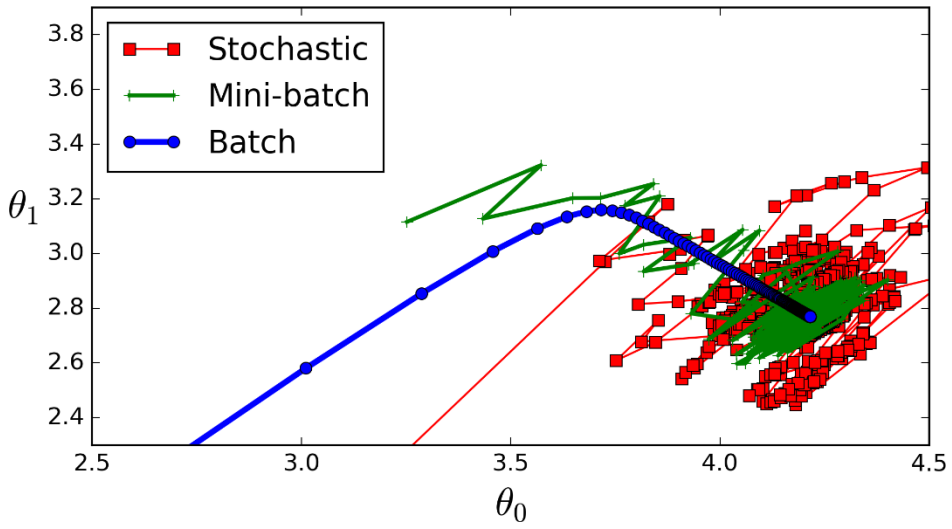


Figura 4.19: Batch Gradient

Por tanto el proceso de aprendizaje se realiza mediante iteraciones sobre los distintos *batch*. Dependiendo del tamaño varía la frecuencia con la que se actualizan los pesos de la red, cuanto menor es el tamaño mayor es la frecuencia de actualización. Generalmente se utilizan tamaños menores al tamaño del *dataset*, *mini-batch*. Si el tamaño es igual se denomina *batch*, y si es igual a 1 se conoce como *stochastic*.

En nuestro proyecto se ha definido un *batch size* de 128 sobre un total de 9850 imágenes del conjunto de entrenamiento, lo que produce un total de 77 iteraciones para la consecución de la totalidad de los datos, además se han establecido 100 *epochs* o recorridos sobre el *dataset* completo.

```
4352/9850 [============>.................] - ETA: 3s - loss: 1.4541 - acc: 0.6312
4480/9850 [============>.................] - ETA: 3s - loss: 1.4526 - acc: 0.6317
4608/9850 [=============>................] - ETA: 3s - loss: 1.4529 - acc: 0.6311
4736/9850 [=============>................] - ETA: 3s - loss: 1.4621 - acc: 0.6301
4864/9850 [=============>................] - ETA: 2s - loss: 1.4667 - acc: 0.6293
4992/9850 [==============>...............] - ETA: 2s - loss: 1.4700 - acc: 0.6300
5120/9850 [==============>...............] - ETA: 2s - loss: 1.4694 - acc: 0.6314
5248/9850 [==============>...............] - ETA: 2s - loss: 1.4704 - acc: 0.6313
5376/9850 [===============>..............] - ETA: 2s - loss: 1.4741 - acc: 0.6306
5504/9850 [===============>..............] - ETA: 2s - loss: 1.4726 - acc: 0.6310
5632/9850 [================>.............] - ETA: 2s - loss: 1.4732 - acc: 0.6300
5760/9850 [================>.............] - ETA: 2s - loss: 1.4787 - acc: 0.6280
5888/9850 [================>.............] - ETA: 2s - loss: 1.4780 - acc: 0.6279
6016/9850 [=================>............] - ETA: 2s - loss: 1.4727 - acc: 0.6287
6144/9850 [=================>............] - ETA: 2s - loss: 1.4747 - acc: 0.6273
6272/9850 [==================>...........] - ETA: 2s - loss: 1.4731 - acc: 0.6276
6400/9850 [==================>...........] - ETA: 2s - loss: 1.4769 - acc: 0.6272
6528/9850 [==================>...........] - ETA: 1s - loss: 1.4732 - acc: 0.6290
6656/9850 [===================>..........] - ETA: 1s - loss: 1.4765 - acc: 0.6285
6784/9850 [===================>..........] - ETA: 1s - loss: 1.4757 - acc: 0.6285
6912/9850 [====================>.........] - ETA: 1s - loss: 1.4778 - acc: 0.6282
7040/9850 [====================>.........] - ETA: 1s - loss: 1.4789 - acc: 0.6274
7168/9850 [====================>.........] - ETA: 1s - loss: 1.4786 - acc: 0.6270
7296/9850 [=====================>........] - ETA: 1s - loss: 1.4778 - acc: 0.6275
7424/9850 [=====================>........] - ETA: 1s - loss: 1.4827 - acc: 0.6270
7552/9850 [======================>.......] - ETA: 1s - loss: 1.4853 - acc: 0.6262
7680/9850 [======================>.......] - ETA: 1s - loss: 1.4851 - acc: 0.6262
7808/9850 [======================>.......] - ETA: 1s - loss: 1.4852 - acc: 0.6262
7936/9850 [=======================>......] - ETA: 1s - loss: 1.4844 - acc: 0.6256
8064/9850 [=======================>......] - ETA: 1s - loss: 1.4819 - acc: 0.6265
8192/9850 [=======================>......] - ETA: 0s - loss: 1.4787 - acc: 0.6274
8320/9850 [========================>.....] - ETA: 0s - loss: 1.4807 - acc: 0.6274
8448/9850 [========================>.....] - ETA: 0s - loss: 1.4803 - acc: 0.6274
8576/9850 [========================>.....] - ETA: 0s - loss: 1.4802 - acc: 0.6280
8704/9850 [=========================>....] - ETA: 0s - loss: 1.4784 - acc: 0.6281
8832/9850 [=========================>....] - ETA: 0s - loss: 1.4829 - acc: 0.6270
8960/9850 [==========================>...] - ETA: 0s - loss: 1.4814 - acc: 0.6272
9088/9850 [==========================>...] - ETA: 0s - loss: 1.4800 - acc: 0.6275
9216/9850 [==========================>...] - ETA: 0s - loss: 1.4792 - acc: 0.6276
9344/9850 [===========================>..] - ETA: 0s - loss: 1.4820 - acc: 0.6263
9472/9850 [===========================>..] - ETA: 0s - loss: 1.4817 - acc: 0.6261
9600/9850 [===========================>..] - ETA: 0s - loss: 1.4848 - acc: 0.6254
9728/9850 [============================>.] - ETA: 0s - loss: 1.4832 - acc: 0.6258
9850/9850 [==============================] - 6s 590us/step - loss: 1.4813 - acc: 0.6264
Epoch 72/100
```

Figura 4.20: Proceso de Entrenamiento de Keras

El dataset sometido a un aumento durante las fases intermedias del desarrollo cuenta con un total de 124065 comparadas con las 9850 imágenes originales. Lo que produce un total de 970 iteraciones por epoch, habiéndose limitado estas a 50.

4.4.2 Codificación de las Etiquetas

Muchos de los algoritmos de *machine learning*, debido a las operaciones matemáticas que realizan, no permiten trabajar sobre datos categóricos (etiquetas o *labels*), requiriendo de valores numéricos.

Para la implementación del proyecto se ha utilizado codificación *One-Hot*. Donde se genera un vector por cada categoría donde únicamente uno de los valores puede ser 1, el resto 0.

El proceso de codificación involucra dos operaciones:

$$V = \begin{bmatrix} Azul & Rojo & Verde & Blanco & Negro \end{bmatrix}$$

En primer lugar se convierten las categorías a un valor entero.

$$V = \begin{bmatrix} 0 & 3 & 4 & 1 & 2 \end{bmatrix}$$

Y posteriormente se codifican como vectores One-Hot.

$$M = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{Bmatrix}$$

4.5 Evaluación

Uno de los principales inconvenientes durante el entrenamiento es el *overfitting* o *underfitting* de los datos. Si los datos de entrenamiento no son suficientes el sistema no será capaz de reconocer correctamente, por el contrario si se entrena en exceso un modelo este aumentará la precisión sobre el conjunto de entrenamiento pero será incapaz de generalizar y empeorará si recibe datos nuevos.

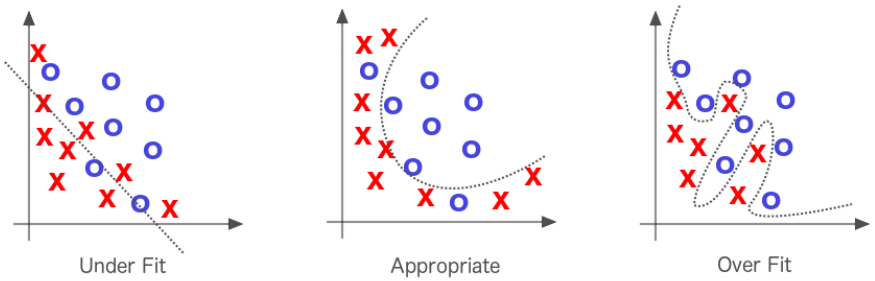


Figura 4.21: Esquema Ajuste del Modelo

Es por ello que generalmente se dividen los datos de entrenamiento en dos, un conjunto de entrenamiento y otro de evaluación. El conjunto de prueba debe ser representativo de los datos totales del *dataset* y ser suficientemente grande como para resultar efectivo, generalmente entre un 70 % y un 80 % del tamaño original. También puede existir un tercer conjunto de prueba o test para comprobar la versión final de un modelo, proporcionando nuevos datos y asegurando una evaluación no sesgada.

Esto permite obtener unos resultados más próximos a la realidad ya que los datos del conjunto de validación permanecen ocultos a la red durante el entrenamiento y de esta manera evita el *overfitting* del modelo.

En la aplicación implementada el conjunto de validación se ha establecido en un 10 % del total mientras que el conjunto de entrenamiento se sitúa en un 90 % Únicamente se ha dividido durante la fase de desarrollo, mediante el aumento del conjunto de datos como descrito en la sección 4.2.1, para ajustar los hiperparámetros de la red y estudiar su desempeño, utilizando la totalidad del *dataset* original en versiones avanzadas del modelo y no el *dataset* aumentado. Esto es debido a dos factores, el considerable incremento en el tiempo tanto de preprocesado de imágenes como en el tiempo de entrenamiento de la red neuronal así como la mejoría en el porcentaje de acierto final de apenas un 1 %, sin embargo si el fin es conseguir la máxima precisión se debería incorporar el aumento de datos en la versión final además de un procesado más agresivo sobre los datos.

Las decisiones que justifican la necesidad de establecer un proceso de aumento de imágenes son varias. En primer lugar la distribución asimétrica del conjunto de datos, existiendo categorías con un único ejemplo, lo que resultaría la presencia de valores exclusivamente en uno de los conjuntos al dividirlos durante la evaluación resultando contraproducente. El segundo motivo y el de mayor importancia es la existencia de un conjunto de datos de test, del que desconocemos los valores, proporcionado por Kaggle [4], permitiendo generar un archivo *csv* con los resultados, subirlos a la competición y contrastar así el rendimiento del modelo. Por tanto este conjunto de pruebas se utilizará como conjunto de validación en las versiones finales de la aplicación.

4.6 Predicción

Todo el proceso de diseño, implementación y entrenamiento de una red neuronal se resume en la predicción. Donde se espera que la red responda con una determinada precisión a una serie de entradas.

El modelo implementado alcanza una precisión del 0.4407, no es una probabilidad apta para producción pero debido a las características del *dataset* seleccionado (4251 clases asimétricas) y a las limitaciones *hardware* que pudieran existir en términos de memoria se considera aceptable. Si comparamos la clasificación pública de la competición de Kaggle el modelo desarrollado se situaría en la posición número 54 de un total de 528 participantes, siendo la probabilidad más alta de un 0.78563 y la segunda de un 0.64924.

Para extraer los resultados y las clases resultantes es necesario revertir el proceso de codificación de las etiquetas. Para ello se obtienen el vector de predicciones de longitud 4251 por cada entrada donde se establece una probabilidad para cada clase, se seleccionan las cinco probabilidades más altas y se revierten sus posiciones en el array a la etiqueta original.

4.7 Posibles Mejoras

4.7.1 Redes Siamesas y Triplet Loss

El caso de estudio de este proyecto, la identificación de ballenas por su patrón de la cola donde existen numerosas clases y se trata de una misma especie, comparte similitudes con las aplicaciones de reconocimiento facial. Dentro de este ámbito una de las técnicas con mayor impacto y rendimiento son las redes siamesas y el uso de *triplet loss* [8].

Sin embargo este procedimiento no se ha implementado debido a las restricciones de memoria del *hardware* utilizado, requiriendo esta técnica de una gran cantidad de memoria.

Su fundamento reside en la comparación de imágenes para encontrar similaridades entre ellas. En concreto partiendo de una imagen a analizar, *anchor image*, se obtiene una imagen de la misma clase, *positive image*, y otra de una distinta, *negative image*. Se compara la imagen ancla con ambas imágenes y se intenta minimizar la distancia entre la imagen original con la positiva a la vez que se maximiza la distancia con la negativa.

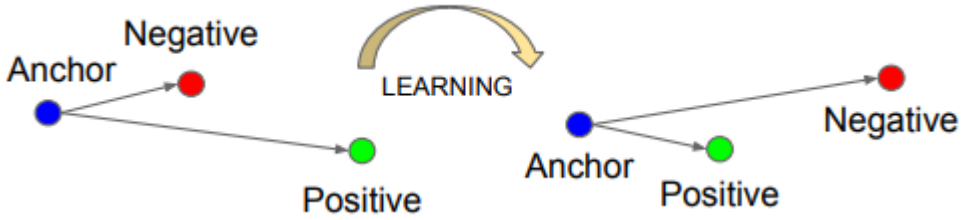


Figura 4.22: Aprendizaje Triplet Loss

Es aquí donde ganan importancia las redes siamesas. Se trata de una arquitectura que contiene dos o más subredes idénticas, en el caso del *triplet loss* cuenta con tres redes convolucionales. Estas subredes comparten parámetros que se actualizan simultáneamente en cada iteración.

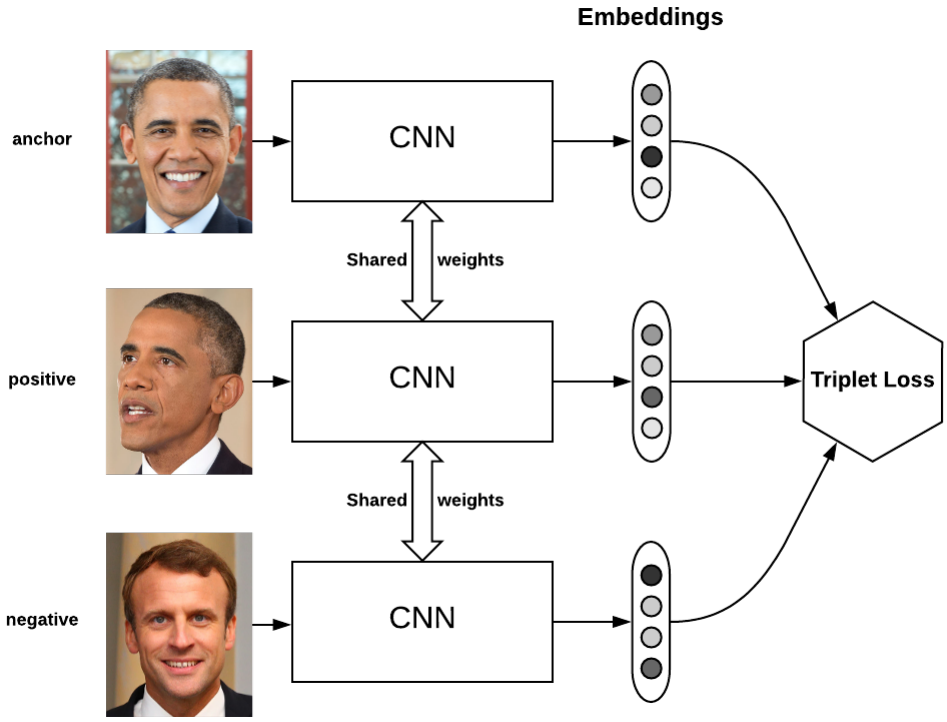


Figura 4.23: Esquema Funcionamiento Triplet Loss

Se analizan las tres imágenes y se computan sus distancias, ajustando los pesos en función de la similaridad de las imágenes y sus distancias. Cada red da como resultado una interpretación de la imagen o *embedding*, por ello, al usarse el mismo modelo para la identificación, las características extraídas serán semejantes en caso de ser la misma clase o diferentes en caso contrario.

Capítulo 5

Conclusiones

El planteamiento original se este proyecto ha sido el estudio y la profundización en los campos de la inteligencia artificial, el *deep learning* y las redes neuronales convolucionales. Es por ello que, a pesar de que la precisión lograda no supone un valor cercano al estado del arte, se han explorado los diferentes métodos y técnicas utilizadas durante los desarrollos de sistemas que permitan reconocer imágenes. Así mismo se ha analizado la influencia e importancia del trabajo previo a las propias redes neuronales como son el procesado de imágenes y la comprensión de los conjuntos de datos.

Durante el desarrollo se alteraron varias de las decisiones originales de diseño. Se planteó la utilización de una única biblioteca para redes neuronales, TensorFlow, pero en posteriores versiones se adoptó además Keras debido a dos factores: su previa inclusión en la biblioteca de TensorFlow y en mayor medida para ofrecer un acercamiento de mayor abstracción a nivel de programación en comparación a TensorFlow.

Se ha podido comprobar como la importancia del rendimiento de una red neuronal no radica exclusivamente en la arquitectura que pueda tener esta sino que el conjunto de datos del que se disponga altera sustancialmente el resultado final, siendo crítico un *dataset* balanceado y con gran cantidad de muestras.

Las principales complicaciones encontradas se dividen entre los dos focos de desarrollo más relevantes: el preprocesado de imágenes y el modelo. El *dataset* seleccionado ha lastrado en gran medida la precisión final de la aplicación, requiriéndose un aumento de datos mucho más agresivo del que se ha realizado actualmente, sin embargo, pese a no proporcionar los resultados óptimos, ofrece una clara visión sobre la importancia de los datos disponibles, así como del funcionamiento de las redes neuronales en cuanto a la percepción de imágenes y como son tratadas. En cuanto a los problemas acontecidos en la implementación del modelo se encuentran las grandes demandas *hardware*, especialmente en términos de memoria RAM de la que no se dispone en los equipos utilizados, lo que ha imposibilitado implementar una arquitectura más eficiente como la detallada en la sección 4.6.

5.1 Impacto Social y Responsabilidad Ética

Las aplicaciones de reconocimientos de imágenes suponen un gran impacto social, desde la utilización en sistemas de realidad aumentada para reconocer objetos o localizaciones al reconocimiento facial o sistemas médicos. Todos estos sistemas funcionan mediante el análisis masivo de imágenes, que en ocasiones puede ser confidencial o sensible, y por tanto esta información debe ser consecuente con la ética y el uso intencionado de la aplicación o de la población destinada a utilizarlo.

Es por todo ello que la inteligencia artificial y, en concreto, el ámbito tratado, no debe permanecer ajena a este tipo de responsabilidades y debe tratarse siempre desde el máximo cuidado y atención a las posibles consecuencias sobre los usuarios o la población que se pueda ver afectada de manera indirecta.

Además su implementación puede ver utilidad más allá del consumo humano, pudiendo aplicarse también en tareas de conservación o medioambientales como puede ser el caso de la aplicación desarrollada, pudiendo mantener un registro e identificación de determinadas especies animales o vegetales.

Bibliografía

[1] Ian Goodfellow y col. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[2] Kaiming He y col. "Deep residual learning for image recognition". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 770-778.

[3] Sergey Ioffe y Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". En: *arXiv preprint arXiv:1502.03167* (2015).

[4] Kaggle. *Humpback Whale Identification Challenge*. URL: `https://www.kaggle.com/c/whale-categorization-playground` (visitado 2018).

[5] Diederik P Kingma y Jimmy Ba. "Adam: A method for stochastic optimization". En: *arXiv preprint arXiv:1412.6980* (2014).

[6] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. "Deep learning". En: *nature* 521.7553 (2015), pág. 436.

[7] Stuart J Russell y Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[8] Florian Schroff, Dmitry Kalenichenko y James Philbin. "Facenet: A unified embedding for face recognition and clustering". En: *Proceedings of*

the IEEE conference on computer vision and pattern recognition. 2015, págs. 815-823.

[9] Theano. *Categorical Cross Entropy*. URL: `http://deeplearning.net/software/theano/library/tensor/nnet/nnet.html#theano.tensor.nnet.nnet.categorical_crossentropy` (visitado 2018).