ESCUELA TÉCNICA SUPERIOR DE INGENIERIA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Estudio de detección de emociones a través de reconocimiento de características faciales con técnicas de AI

Autor: Francisco Alfredo Moreno Urrea

Director: María Dolores Cano Baños

Codirector: Andrea Martínez Martínez

Agradecimientos

A mis padres y a mi hermana por la paciencia tenida durante todo el trayecto académico.

A mi directora de proyecto Lola y a mi codirectora Andrea por su inestimable ayuda y dedicación durante todos estos meses para que este proyecto pudiera salir adelante.

Gracias

Contenido

Relación de figuras

Capítulo 1. Introducción y Objetivos

1.1 Introducción

Este proyecto está basado en la detección y posterior clasificación de diversas emociones obtenidas a través de imágenes. Las principales bases de este proyecto son los denominados Machine Learning y Deep Learning (Aprendizaje automático y aprendizaje profundo en castellano respectivamente).

Para poder entender que es y cómo funciona la detección de emociones, primero es necesario explicar que es el reconocimiento facial, puesto que estas dos tecnologías están íntimamente ligadas.

1.2 Relación entre Reconocimiento Facial y detección de emociones.

Se denomina reconocimiento facial a verificar o reconocer la identidad de una persona en función de sus características fisiológicas empleando un algoritmo automático que se basa en la detección de puntos faciales *(1)*.

Dentro del reconocimiento facial es necesario una serie de pasos para poder realizar correctamente el mismo.

1. Detección del rostro en una imagen dada.
2. Extracción de características faciales obteniendo su información biométrica
3. Comparación y cotejo de la información biométrica con datos previamente almacenados en una base de datos.
4. Toma de decisión en función del porcentaje obtenido en el paso anterior.

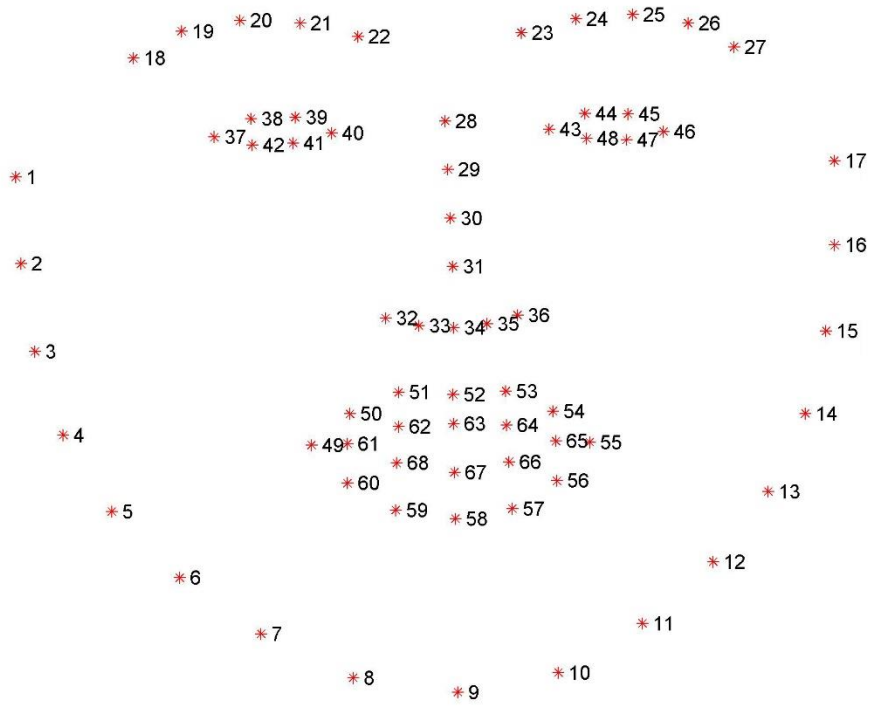


Figura 1. Puntos de referencia faciales (2)

Como podemos comprobar en la imagen anterior, los puntos faciales de los que se sirve el algoritmo se distribuyen de la siguiente forma:

- Puntos del 1 al 17 corresponden al contorno de la cara.
- Puntos del 18 al 27 corresponden a las cejas.
- Puntos del 28 al 36 corresponden a la nariz.
- Puntos del 37 al 48 corresponden a los ojos.
- Puntos del 49 al 68 corresponden a la boca.

Con estos puntos faciales se puede determinar la identidad de una persona.

En cuanto a la detección de emociones, el proceso es similar ya que distintas combinaciones del rostro permiten identificar la emoción en la que se encuentra la persona.

Las distintas emociones trabajadas son:

1. Neutral: se corresponde con el rostro en estado de reposo.
2. Felicidad: se corresponde con las comisuras de los labios estirados hacia arriba levemente (puntos 49, 50, del 54 al 56, 60, 61 y 65) y los ojos rasgados (puntos 38, 39, 41, 42, 44, 45, 47 y 48).
3. Enfado: ceño fruncido, lo que se traduce en los párpados apretados (puntos 38, 39, 41, 42, 44, 45, 47 y 48), las cejas juntas y hacia abajo (puntos de 20 al 25) y los labios apretados (puntos del 50 al 54, y del 56 al 60).
4. Tristeza: se corresponde con las comisuras de los labios estirados hacia abajo levemente (puntos 49, 50, del 54 al 56, 60, 61 y 65).

5. Sorpresa: se muestran las cejas elevadas (puntos del 19 al 26), los párpados más abiertos de lo habitual (puntos 38, 39, 41, 42, 44, 45, 47 y 48) y boca entreabierta o abierta del todo(puntos del 50 al 54, del 56 al 60, del 62 al 64, y del 66 al 68).

1.3 Usos, ventajas e inconvenientes

Esta tecnología tiene diversos usos y en distintos ámbitos tan dispares como nivel de gestión de personal en una empresa (detección de emociones para su proceso de selección de personal (3), seguimiento anímico de sus trabajadores, detectar episodios de agotamiento y/o depresión), nivel de marketing empresarial (estudios de mercado, también conocido como neuromarketing (4)) con el que se puede obtener en tiempo real la evaluación facial de un cliente ante un determinado producto. El nivel científico es otro de los ámbitos en el que se emplea dicha tecnología, concretamente en el ámbito de neurocirugía (permite cierta comunicación no verbal en aquellas personas con daño cerebral o degeneración neuronal) (5) (6).

Así pues, del punto anterior podemos concluir que nos encontramos ante un mercado muy amplio y creciente, ya que las últimas noticias acerca de inversión de capital estimaban que en los próximos 5 años se iban a invertir más de 40 mil millones de dólares para mejorar y potenciar esta tecnología. Algunas potencias económicas tales como EE. UU. o China han invertido una gran suma de dinero en esta tecnología pues creen que hay una relación entre la emoción del individuo y su comportamiento. Unos de los argumentos que ofrecen es que, analizando la emoción de las personas, se podrían prevenir futuros ataques terroristas (7) (8).

A pesar de las ventajas potenciales mencionadas anteriormente, también existe una serie de inconvenientes a tener en cuenta. El principal inconveniente es que, en numerosas ocasiones, no siempre se corresponde la expresión facial con como una persona se siente de forma interna realmente, luego se puede engañar y provocar falsos positivos (9). Por otro lado, no hay una regularización de estos datos, lo que se traduce en que el usuario pierde privacidad de estos (10). Otro punto a tener en cuenta es que esta tecnología está catalogada como racista y machista, aunque dicen que es probable que sea debido a que no dispongan de gran variedad de muestras para entrenar el modelo (11).

1.4 Justificación del proyecto

Para justificar la emoción de este proyecto es necesario hablar primero de la detección de emociones utilizando el reconocimiento facial y después del sesgo algorítmico.

Se estima que la tecnología encargada de analizar las expresiones faciales y deducir sentimientos gracias a ellas es un mercado en expansión. Actualmente, grandes empresas como Microsoft y Apple están apostando por este modelo de negocio que ronda los 18 mil millones de euros. (12)

La Conferencia General de la Organización de las Naciones Unidas se reunió en 1978 para aprobar y proclamar la presente *Declaración sobre raza y prejuicios raciales* indicando en el artículo 9.2 que deben tomarse medidas especiales para garantizar la

igualdad en dignidad y en los derechos de los individuos, evitando dar un carácter discriminatorio en el plano racial. (13)

Sin embargo, en el tema de la Inteligencia Artificial, no se cumple. Diversas investigaciones han concluido que esta tecnología puede ampliar los prejuicios sociales en dichos algoritmos. Los grupos más desfavorecidos son las mujeres y las personas de color.

Una de las claves de estos factores es que los grupos dominantes están claramente beneficiados con mayores índices de precisión en comparación con los minoritarios. Esto es consecuencia de que la población de hombres blancos está sobrerrepresentada y es por esta razón que este proyecto solo se centrará en imágenes de personas de color (14).

1.5 Objetivos

El objetivo de este TFG es conocer y aplicar, de forma práctica, técnicas de reconocimiento de emociones mediante el reconocimiento facial usando modelos de AI. Además de otros como:

- Aprendizaje del lenguaje de programación Python y el uso de librerías de alto nivel como Keras y TensorFlow.
- Adquisición de conocimientos de técnicas AI.
- Uso de herramientas cloud para conocer el estado de técnicas en reconocimiento facial de emociones.

1.6 Contenido del resto de capítulos.

En primer lugar, empezaremos con el desarrollo de la base teórica, en el que haremos un resumen de los anteriores proyectos o investigaciones sobre la detección de emociones, las soluciones propuestas, así como también de su funcionamiento.

En segundo lugar, hablaremos de las herramientas utilizadas a lo largo de este proyecto: instalación, configuración de los programas, etc.

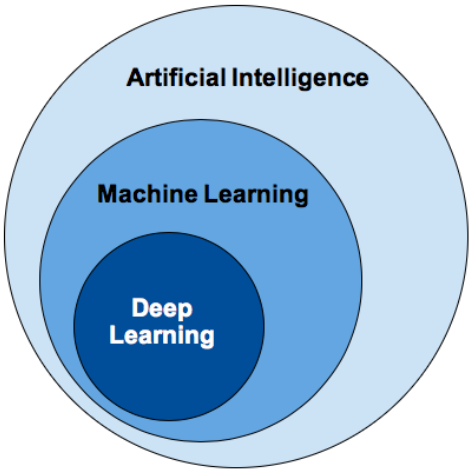
A continuación, explicaremos paso a paso el código utilizado a lo largo del proyecto, los resultados y sus análisis.

Para terminar, concluiremos el proyecto con un resumen, además de añadir líneas futuras para posibles continuaciones.

Capítulo 2. Base teórica.

2.1 Introducción teórica.

Para poder entender mejor los algoritmos realizados en el mundo del reconocimiento de emociones lo primero de todo será contextualizar esta tecnología. En primer lugar, hablaremos de que es la inteligencia artificial. Posteriormente hablaremos de Machine Learning y, por último, Deep Learning, puesto que Deep Learning es un subtipo de Machine Learning y este es un subgrupo de la inteligencia artificial. Gráficamente, podría verse en la siguiente imagen.



(15)

La inteligencia artificial (IA) es una rama perteneciente a las ciencias de la Computación que tiene como objetivo principal brindar soluciones a problemas de alta complejidad utilizando equipos informáticos como hardware y software emulando la capacidad cognitiva de los seres humanos.

El aprendizaje automático, en inglés *Machine Learning*, es una rama perteneciente a la IA cuyo objetivo es aprender mediante ejemplos y estudiar el reconocimiento de patrones para poder generalizar correctamente. En otras palabras, obtener una predicción precisa ante una entrada no encontrada durante el entrenamiento.

El aprendizaje profundo, en inglés *Deep Learning*, es un subgrupo de *Machine Learning* en el que se caracteriza por la presencia de un número elevado de capas ocultas. Estas redes empezaron a desarrollarse en los años 60.

En el año 1958 Rosenblatt propuso el perceptrón monocapa, la capa de salida. Posteriormente esta fue ampliada al Perceptrón multicapa, la cual ya disponía de un mayor número de capas, concretamente disponía de capas ocultas localizadas entre la capa de entrada y la capa de salida. (16)

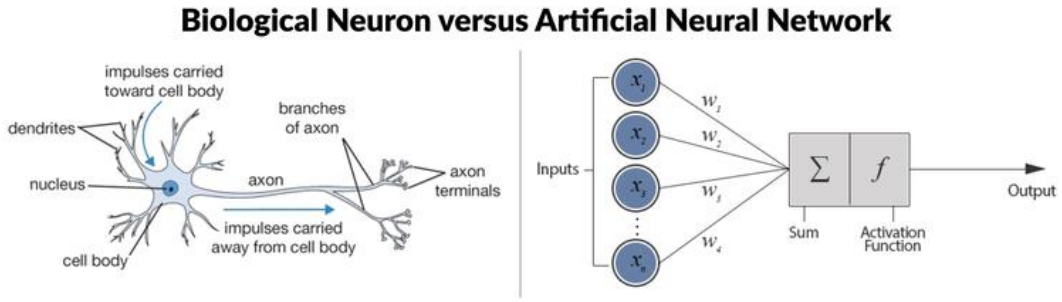


Figura 2. Comparación entre neuronas biológicas y artificiales (17)

A partir de finales de los años 70 y principios de los 80, Rumelhart describía un nuevo procedimiento de aprendizaje, *backpropagation*, con el que se podían calcular los pesos de las neuronas correspondientes a las distintas capas de la red neuronal (18).

A partir de los años 90, se popularizó este tipo de aprendizaje debido a la introducción de la GPU para procesamiento general, esta aparición permitía la resolución del alto coste computacional que provocaba una red con un gran número de neuronas. Actualmente, ya no se necesita tener una GPU física debido a que existen grandes proveedores que ofrecen servicios de infraestructura en la nube que nos proporcionan este hardware (19).

El aprendizaje puede dividirse en tres tipos fundamentales (20):

- Aprendizaje supervisado:

  Se basa en la generación de conocimiento a través de la proporción de una respuesta deseada ante una determinada entrada, dicha entrada no forma parte del conjunto de entrenamiento. Este tipo de aprendizaje es empleado en clasificación y en regresión para desarrollar modelos predictivos.

  Las técnicas de clasificación se caracterizan por predecir respuestas del tipo discreto, tales como detectar si un tumor es benigno o maligno o detectar si un correo electrónico es spam o no. El objetivo de esta técnica es poder determinar las regiones de decisión que separan las diferentes clases existentes de entrada. Alguno de los algoritmos que pertenecen a esta técnica de clasificación son: redes neuronales, árboles de decisión, máquinas de soporte vector y clasificadores bayesianos entre otras.

  Las técnicas de regresión se caracterizan por predecir respuesta del tipo continuas, tales como predecir el precio de un producto o el tiempo de permanencia de un usuario suscrito a un producto. El objetivo de esta técnica consiste en abstraer una función continua que relacione la entrada y la salida dependiendo de un número limitado de ejemplos. Alguno de los algoritmos más populares de esta técnica es: modelo lineal, modelo no lineal, regularización y también redes neuronales.

- Aprendizaje no supervisado:

  Se basa en el reconocimiento de patrones interesantes a través de los datos de entrada que, a diferencia del aprendizaje supervisado, estos no están etiquetados. La técnica más común de este tipo es el *clustering*. Uno de los ejemplos de *clustering* podría ser la segmentación de clientes para poder ofrecer productos y/o servicios que necesiten.

  Las ventajas de usar este tipo de aprendizaje es eliminar redundancia y la eliminación de almacenamiento de los datos. Alguno de los algoritmos más conocidos de este tipo son los de k-means, modelo de Markov oculto y mapas autoorganizados.

- Aprendizaje por refuerzo:

  Una vez obtenida la salida de la red neuronal, una señal de refuerzo, conocida como *feedback*, indicará si esta es correcta o, por el contrario, es errónea. La explicación a este tipo de aprendizaje es el siguiente: primero el algoritmo recibe como entrada un dato desconocido, sin etiqueta, y dará un resultado, dependiendo de la respuesta que obtiene el algoritmo este puede percibir las características de esos datos con el objetivo de mejorar las predicciones futuras.

2.2 Redes Neuronales Convolucionales (CNN)

El tipo más popular y que más éxito tiene a la hora de entrenar un algoritmo para identificación de rostros y de estados de ánimos es el aprendizaje supervisado, también conocido como CNN, debido a ello será el tipo de aprendizaje en el que nos centraremos. Gracias a este tipo de aprendizaje se puede modelar una red de neurona artificial.

Las Redes Neuronales Artificiales (RNA) son maquinas inspiradas en las redes neuronales biológicas, están diseñadas para modelar la forma en el que el cerebro realiza alguna tarea en concreto.

Una red neuronal convencional (CNN) es un tipo de RNA que las neuronas corresponden a campos receptivos similares a la corteza visual primaria (V1). Son muy efectivas para la clasificación y segmentación de imágenes y visión artificial. A pesar de que esta red neuronal fue introducida en el año 1980, no fue hasta el año 2012 cuando empezó a popularizarse gracias a la utilización de la GPU durante el entrenamiento para reducir el tiempo y el error de manera muy significativa en el campo del reconocimiento de imagen. Esta arquitectura de red se denomina *AlexNet*.

A pesar de no ser el primer diseño de una red neuronal utilizando una GPU, si fue la más popular, ya que ganó hasta cuatro concursos de imagen y se mejoró el rendimiento respecto a otras arquitecturas y no fue hasta 2015 cuando esta red CNN fue superada por una red de Microsoft Research Asia.

La arquitectura más habitual dentro de las redes neuronal convolucional es la mostrada en la siguiente figura

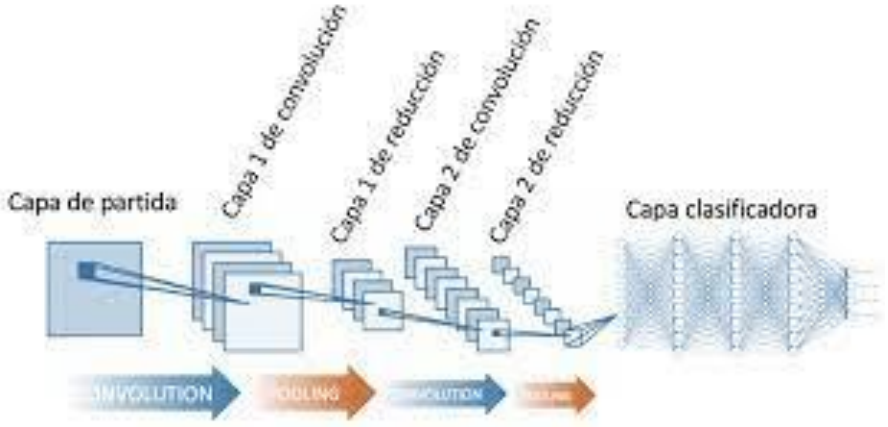


Figura 3. ANN (21)

Como se puede apreciar, esta arquitectura está compuesta de capas conectadas entre la siguiente y la anterior. A pesar de denominarse capas, cada una de ellas realiza una función distinta.

Las capas en las que nos centraremos son las siguientes:

- Capas de Convolución
- Capas de Pooling
- Capas de Activación
- Capas totalmente conectadas
- Capas de Normalización de *Batch*
- Capas Dropout

Empezaremos analizando cada capa indicando la descripción y su funciones.

- Capa de Convolución (CONV)

  Esta es la capa principal de la red neuronal cuya función es realizar la operación matemática de la convolución. Se realizan operaciones de productos y sumas entre la capa de partida y en filtros (o kernel) para generar un mapa de características. Estas características corresponden a posibles ubicaciones del filtro en la imagen temporal. Si estuviésemos en la primera capa, la de entrada de la red neuronal, la profundidad, sería el número de canales de la imagen. Si, por el contrario, la capa es distinta, la profundidad sería el número de filtros de la capa anterior.

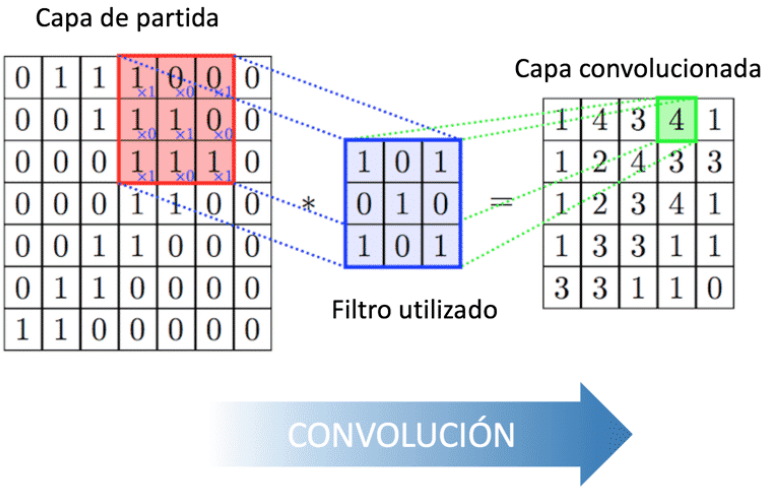


Figura 4. Ejemplo convolución (21)

Dentro de esta capa hay dos parámetros muy importantes, *stride* y *zero-padding*.

- *Stride* es el número de pixeles desplazados sobre la matriz de entrada, si el stride es de valor 2, moveremos los filtros a dos pixeles a la vez.

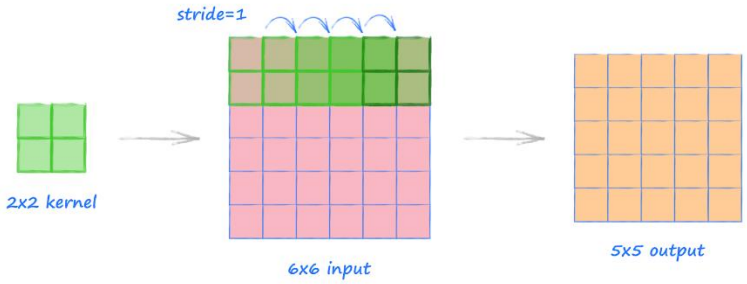


Figura 5. Ejemplo Stride (22)

- *Zero-padding* consiste en el rellenado de los bordes utilizando ceros en nuestra imagen para que conserve su tamaño original al aplicarle la convolución. Esto es importante debido a que si el tamaño de la entrada disminuye rápidamente no podríamos entrenar redes profundas.

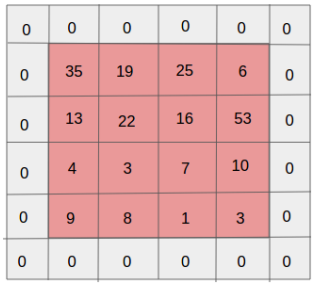


Figura 6. Ejemplo Zero-Padding (23)

- Capa de Pooling

  Aunque las capas de convolución permiten reducir la cantidad de parámetros, es habitual utilizar capas Pooling para acelerar esta reducción de dimensiones. Estas reducciones de características generan pérdidas de información que se traducen en pérdidas de precisión, a pesar de ello esto se realiza para mejorar la compatibilidad.

  Existen dos maneras para reducir el tamaño de la entrada. Una es mediante capas de convolución como vimos anteriormente y otra es utilizando capas pooling. La principal función es reducir progresivamente el tamaño especial del volumen de entrada con el fin de reducir el número de parámetros de la red y su computación.

  Existen tres tipos de capas de agrupación, agrupación máxima, agrupación promedio y agrupación global.

  - Agrupación máxima: Operación que selecciona el elemento máximo de la región del mapa de características cubierta por el filtro.

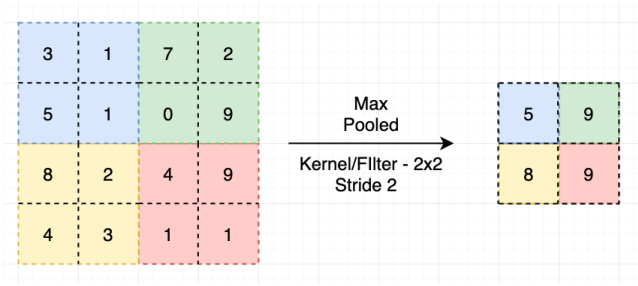


Figura 7. Ejemplo agrupación máxima (24)

    En la imagen anterior podemos observar cómo es su funcionamiento para un filtro de 2x2 y un *stride* de 2. Para el primer valor obtendremos el valor máximo entre los valores de 3,1,5,1. Una vez obtenido el máximo valor la ventana se desliza a lo largo de la imagen de salida reduciendo la dimensión.

  - Agrupación media: Operación que selecciona el valor promedio de la región del mapa de características cubierta por el filtro. El proceso es muy similar al de agrupación máxima, a excepción de que el valor obtenido es la media de los valores.

    Si usamos la imagen anterior como ejemplo, el primer valor de la nueva dimensión se obtendría haciendo la media entre 3,5,1 y 1, obteniendo así el valor de 2.5.

    Este tipo de agrupación se utiliza en LeNet.

- Agrupación global: Operación que reduce cada canal en el mapa de características a un solo valor. Esta agrupación puede ser una agrupación máxima global o puede ser una agrupación promedio global.

- Capas de Activación

  A pesar de que se denomina capa no es una capa como tal, ya que no aprende ningún valor y a menudos estas son omitidas en los diagramas de arquitecturas de red, debido a que está implícito que después de una capa de convolución le sigue una de activación.

  Las funciones de activación más populares se pueden dividir en tres: funciones de cresta, radiales y de plegado.

  - Las funciones de activación de cresta actúan sobre una combinación lineal de las variables de entrada. Las más conocidas y utilizadas son lineal, Heaviside, logística y en la que nos centraremos a continuación, ReLU.

    ReLU es la abreviación de Unidad Lineal Rectificada.

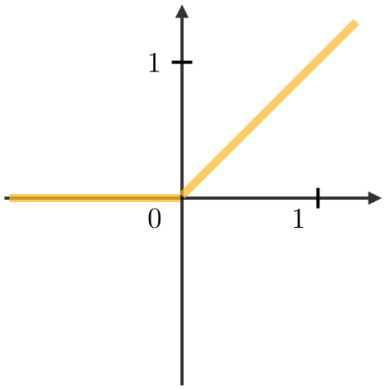


Figura 8. Función de activación ReLu (25)

Tal y como se puede apreciar en la imagen anterior, esta función toma valores de 0 para valores negativos y para valores positivos se incrementa linealmente. Las ventajas de uso de esta función son: computacionalmente eficiente y la no saturación, por el contrario las desventajas de esta función son que no es diferenciable en 0, aunque también es cierto que este problema llegó a ser solucionado en la versión de *ReLU* con fugas (en inglés Leaky ReLU) que permite un pequeño gradiente positivo cuando la unidad no se encuentra activa. En vez de darle un valor de 0 para valores negativos le da un valor de 0.01x.

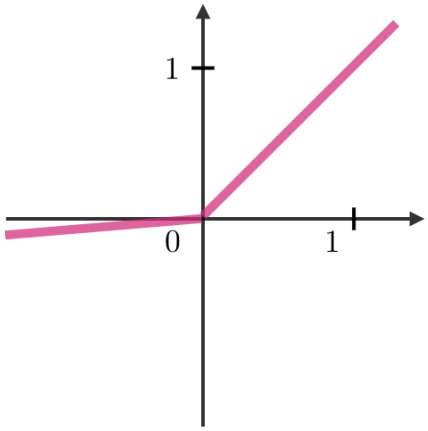


Figura 9. Función de activación ReLu con fugas (25)

También es utilizada la versión ELU, puesto que se pueden obtener una precisión mayor de clasificación respecto a los *ReLU*. Esta función es diferenciable en todas sus partes.

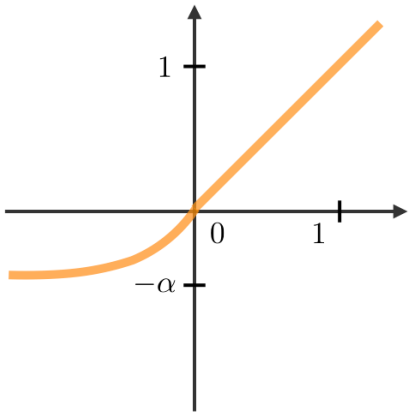


Figura 10. Función de activación ELU (25)

- Las funciones de activación radial se utilizan en redes RBF, se caracterizan por tener un número de neuronas de salida mucho menor que el de neuronas en la capa oculta. La función de activación más popular es el Gaussiano.

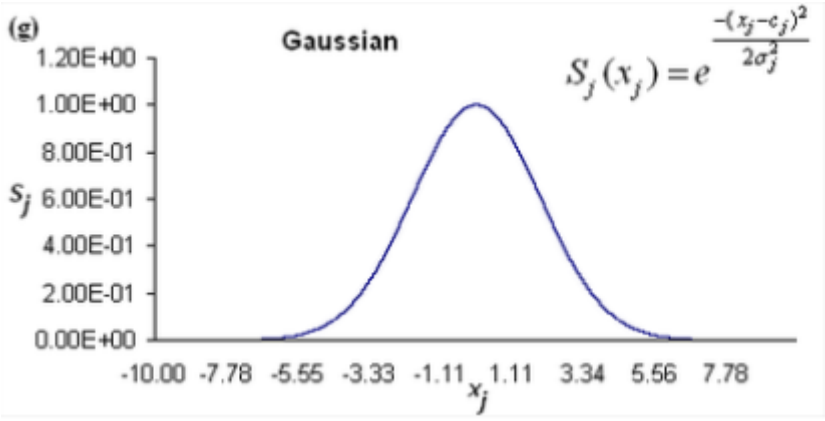


Figura 11. Función de activación radial (26)

- Las funciones de activación plegables se utilizan en las capas de salida de redes de clasificación multiclase, la función más conocida es la *SoftMax*, que se utiliza para convertir valores en probabilidades de actuación, esta función correlaciona la recompensa que se espera al realizar una determinada acción y la probabilidad de escoger esa misma.

$$\varphi(v_j) = \text{softmax}(v_j) = \frac{\exp(v_j)}{\sum_{i=1}^{r} \exp(v_i)} \approx P(C_j|x)$$

Figura 12. Función de activación SoftMax

- Capas totalmente conectadas

  Este tipo de capas se caracteriza porque las neuronas en una capa completamente conectada se encuentran enlazadas todas con todas, con las activaciones de la capa anterior. Estas capas suelen encontrarse al final de la red neuronal y se pueden utilizar para optimizar objetivos como pueden ser el puntaje de clases.

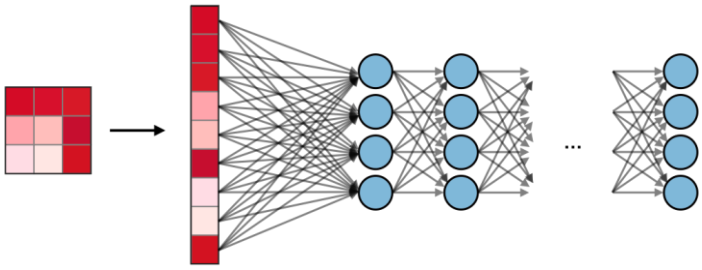


Figura 13. Ejemplo capas totalmente conectadas (25)

- Capas de Normalización de *Batch*

  Como su propio nombre indica, la función de esta capa es la de normalizar el resultado de las activaciones de una entrada. Esta capa se sitúa entre la entrada de activación y la capa de red de la siguiente.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

(27)

  Donde x es nuestro *Batch* de activaciones. También es posible encontrarse dentro de la raíz de Var un factor E que es un valor positivo muy pequeño.

  Aplicando esta ecuación, la normalización resultante tendrá una media 0 y una desviación típica de 1.

Esta capa es muy útil para reducir el número de iteraciones necesarias para entrenar una red neuronal, a costa de aumentar el tiempo de entrenamiento y recursos computacionales.

- Capa de *Dropout*

  Esta capa ignora neuronas al azar durante la fase de entrenamiento. Con este deshecho de neuronas se pretende evitar un sobreajuste.

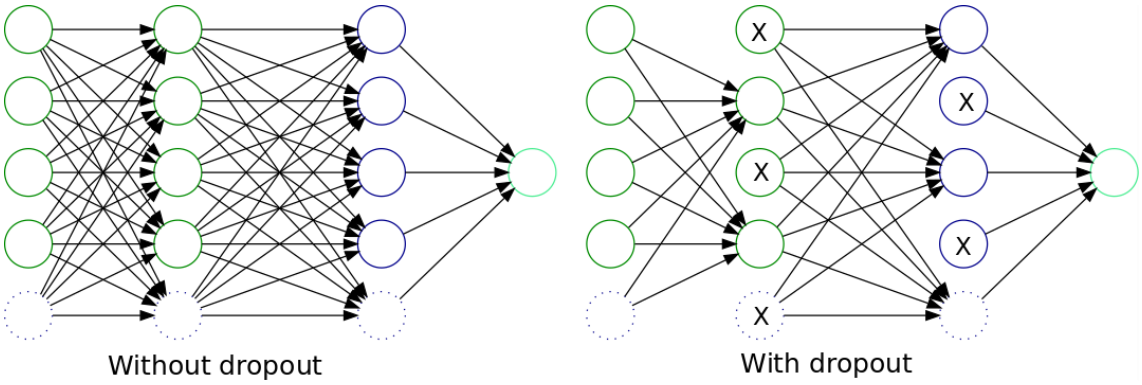


Figura 14. Capa Dropout (28)

El funcionamiento es el siguiente, en cada *Batch*, la capa tiene una probabilidad p de desconectar la entrada. Al desactivar esta neurona y al hacerlo de forma aleatoria nos aseguramos de que ninguna de ella es capaz de reconocer un patrón determinado.

Con esta capa se consigue disminuir el coste computacional, ya que se utilizan un número menor de neuronas y la reducción de *overfitting*.

2.3 Arquitecturas utilizadas en clasificación de imágenes.

La clasificación de imágenes es un campo avanzado dentro de las redes neuronales. A pesar de la existencia de infinitas formas de organizar las capas de neuronas de las redes convolucionales, lo más habitual es configurar estas capas utilizando modelos comunes y realizando alguna que otra modificación.

Estos modelos comunes datan del Reto de Reconocimiento Visual de Gran Escala organizado por *ImageNet*, también conocido como *ILSVRC*, que desde el año 2010 hasta el año 2017 usuarios y empresas de todo el mundo compiten por ver quien presenta el modelo más preciso en el tema de la clasificación de imágenes y detección de objetos sobre un conjunto de datos. En esta competición se generaron alguna de las arquitectura de redes neuronales convencionales más exitosas que se encuentran y provocó un gran avance en el desarrollo de nuevas arquitecturas de redes neuronales generales.

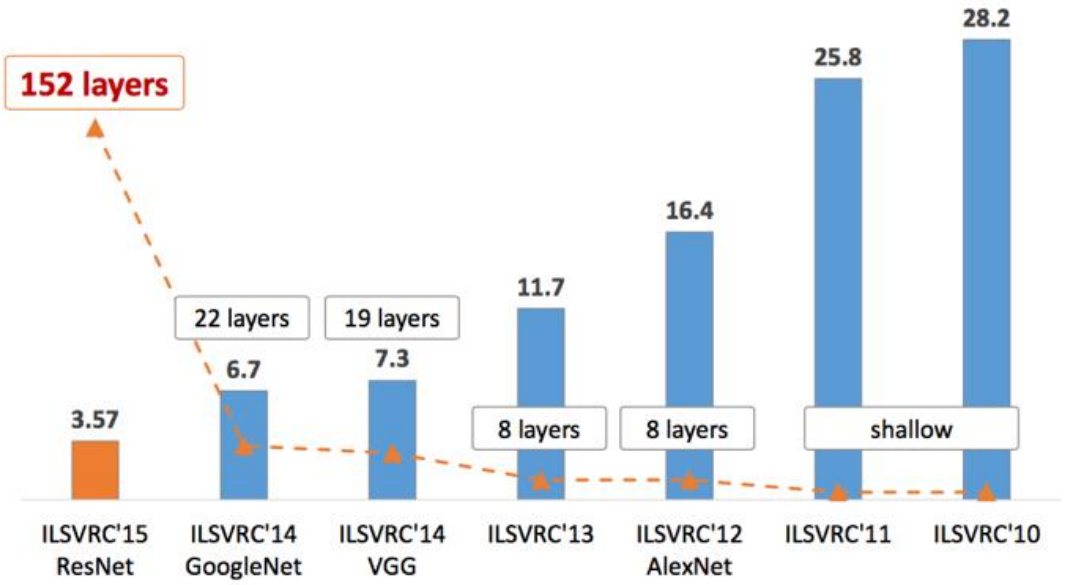


Figura 15. Evolución de las arquitecturas CNN (29)

Las arquitecturas más exitosas e importantes de las CNN en la clasificación de imágenes son: *AlexNet, ZFNet,VGGNet,GoogleNet, ResNet*.

2.3.1 Arquitectura AlexNet (29) (30)

Se trata de una arquitectura diseñada y presentada por Alex Krizhevsky en *ILSVRC* del año 2012 compuesta por 8 capas (5 etapas convolucionales y 3 capas completamente conectadas). A pesar de que era un modelo computacionalmente costoso, era factible debido a la utilización de la GPU para su entrenamiento. Aun así, el entrenamiento duró 6 días. Consiguió rebajar la tasa de error del 25% al 17% que tenía *LeNet* desde el año 1998. Se caracterizaba, entre otras cosas, por tener 60 millones de parámetros y 650 mil neuronas. Parte de su éxito residía en usar una ReLu en la función de activación en lugar de la función tanh que era la más popular en ese momento.

El problema de sobreajuste lo solucionó con un aumento de datos (realizando modificaciones en las fotos tales como ampliarlas o rotarlas) y estableciendo una tasa de Dropout.

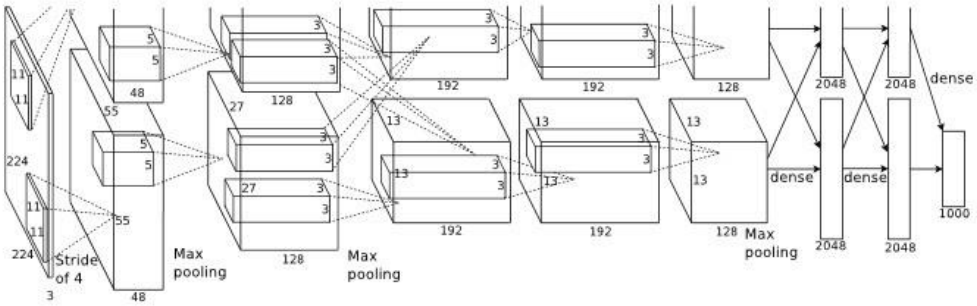


Figura 16. Arquitectura AlexNet (30)

2.3.2 Arquitectura ZFNet (31) (29)

Ganadora del reto *ILSVRC* en 2013, posee una arquitectura muy similar a la diseñada por Krizhevsky el año anterior, a diferencia de que los tamaños de los filtros se reducían (de 11x11 a 7x7) y el tamaño de los *strides* también. La reducción en el filtro tenía su explicación en que al utilizar filtros más grandes se estaba perdiendo mucha información en los pixeles. Esta red se entrenó utilizando para su activación *ReLu* y descenso de gradiente estocástico por lotes.

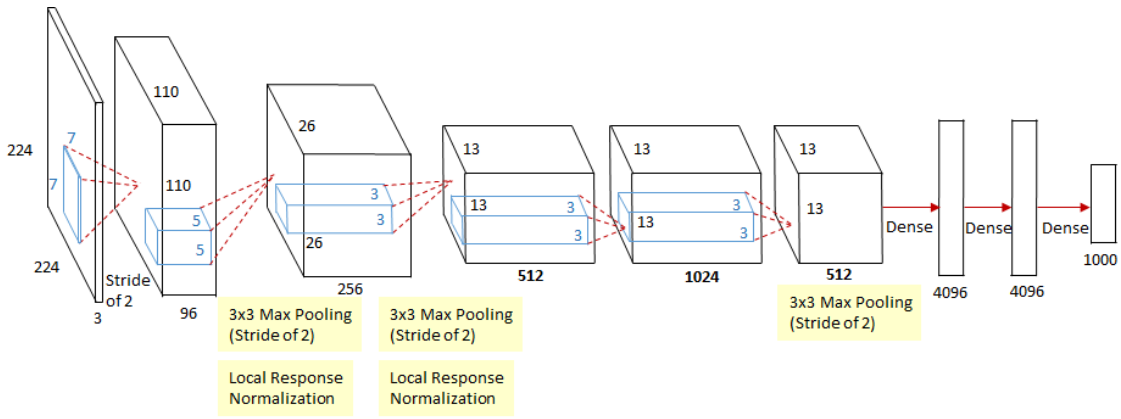


Figura 17. Arquitectura ZFNet (32)

2.3.3 Arquitectura VGGNet (33) (29)

Aunque la idea se presentó en 2013, no fue hasta 2014 cuando se quedó finalista en la competición, se presentaba como una arquitectura simple si se comparaba con las anteriores ganadoras.

Utilizaba filtros 3x3 en lugar de los 11x11 y los 7x7 de las arquitecturas anteriores. Argumentaban los autores que si se tenía dos filtros consecutivos de 3x3 darían un campo receptivo efectivo de 5x5 y si se utilizaban tres filtros consecutivos darían un campo receptivo efectivo de 7x7.

Esta red se hizo muy popular debido a su sencillez y a su gran funcionamiento, ya que consiguió una tasa de error del 7.2%

Figura 18. Arquitectura VGGNet (34)

2.3.4 Arquitectura GoogleNet (29) (35)

Ganadora del reto *ILSVRC* en 2014, fue presentada por Google y homenaje al autor de la red *LeNet* ya que fue inspirada en dicha red. Se propuso un módulo denominado módulo de inicio que incluía saltar conexiones en la red formando un Mini modulo. Este algoritmo elimina todas las capas completamente conectadas y utiliza la agrupación promedio. La ventaja que tiene esta arquitectura es que ahorra muchísimos parámetros, pero se sale de la línea que se tenía hasta el momento de apilar capas de redes convolucionales de manera secuencial.

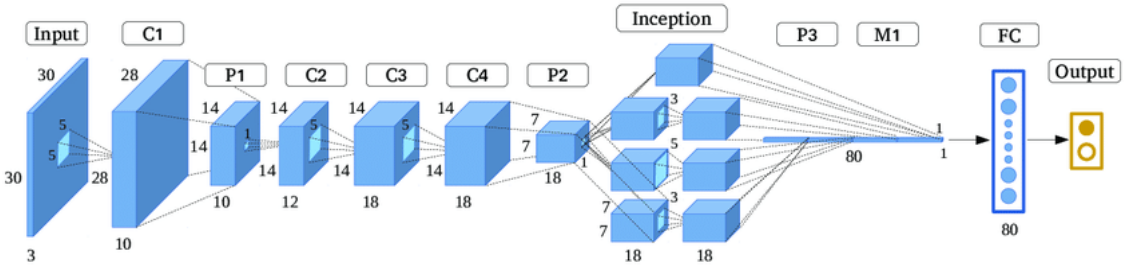


Figura 19. Arquitectura GoogleNet (36)

2.3.5 Arquitecturas ResNet (37) (29)

Arquitectura aparecida en 2015 que soluciona el problema de la profundidad de la red utilizando el concepto de bloque de aprendizaje visual. Este era necesario puesto que en las redes neuronales clásicas dejaban de funcionar correctamente cuando la profundidad de la red superaba un cierto umbral.

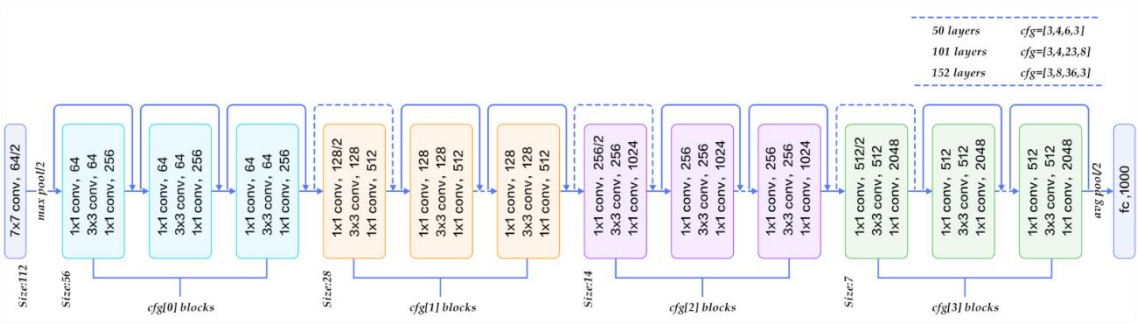


Figura 20. Arquitectura ResNet (34) (36)

En la siguiente imagen se tiene una tabla resumen de las tecnologías, incluyendo la tasa de error y el número de parámetros.

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Figura 21. Tabla comparativa CNN (29)

3. Herramientas

Para la elaboración de este proyecto se han utilizado dos herramientas: *Google Colab* y *Anaconda*.

Google Colab es un servicio cloud que nos permite un uso gratuito pero limitado de las GPUs y TPUs del propio Google. Además, nos proporciona unas bibliotecas básicas instaladas y necesarias para la resolución de este proyecto tales como *TensorFlow* y *Keras* -todo esto basado en los Notebooks de *Jupyter*-. El uso de la GPU y las bibliotecas básicas instaladas nos confirman que la herramienta de *Google Colab* es una excelente opción para practicar y mejorar en técnicas que requieran un nivel de computación elevado tales como Machine Learning, desarrollo de aplicaciones, etc. Por otro lado, nos ofrece una opción económica en caso de no tener financiación para mejorar los recursos hardware

A pesar de los límites de uso, Colab nos permite acceso gratuito a los recursos computacionales y no requiere ninguna instalación previa ni configuraciones para usarlo, solo se necesita una cuenta de Gmail. También nos ofrece la posibilidad de acceder a GPUs más rápidas, tiempos de ejecución más largos y contar con mayor memoria RAM y espacio en el disco a cambio de una facturación periódica.

Debido a estas limitaciones, se ha buscado una solución para contar con un tiempo de ejecución más larga sin tener que contratar la suscripción mencionada en el párrafo anterior. Dicha solución la hemos encontrado con la herramienta *Anaconda*.

Este software nos permite ejecutar el proyecto en local pudiendo así ejecutar códigos que requieren tiempos de ejecución más largos de los que permite la versión gratuita de *Google Colab*, a coste de contar con un tiempo de procesado mayor.

A continuación, se explicarán las configuraciones llevadas a cabo para las distintas herramientas y en el caso de *Anaconda*, los pasos a seguir para poder hacer uso de la GPU o de la CPU

3.1 Configuración *Google Colab*

Como hemos comentado anteriormente, *Google Colab* no necesita ninguna instalación previa. Lo único que se necesita es una cuenta de GMAIL.

Para cargar los datos en el notebook hay diversas maneras, de las cuales las tres más populares son:

- Acceder a nuestro Google Drive
- Cargar los datos desde una instancia S3 de Amazon mediante los comandos !wget y !unzip.
- Obtener los datos en nuestro disco local utilizando el botón de subir al almacenamiento de sesión.

Si se desea utilizar imágenes almacenadas en drive, se necesitaría primero montar la carpeta.

```
from google.colab import drive
drive.mount('/content/drive')
```

Figura 22. Configuración Google Colab

Para ello, se debe de importar primero drive de *Google Colab* y, a continuación, montaremos el archivo. Para llevar a cabo este proceso, nos pedirá un código de autorización con el fin de habilitar el acceso.

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_i

Enter your authorization code:
```

Figura 23. Código de verificación Google Colab

Para obtener dicho código, pincharemos el enlace que nos aparece. En la nueva página, seleccionaremos nuestra cuenta de Gmail y permitiremos el acceso de "Google Drive for desktop" a nuestra cuenta de Google. En la siguiente página, nos aparecerá ya el código y solo nos quedaría copiarlo y pegarlo en la casilla que nos apareció en *Google Colab*.

Si todo es correcto, ya nos aparecería en la parte izquierda las carpetas que tenemos en nuestro GMAIL.

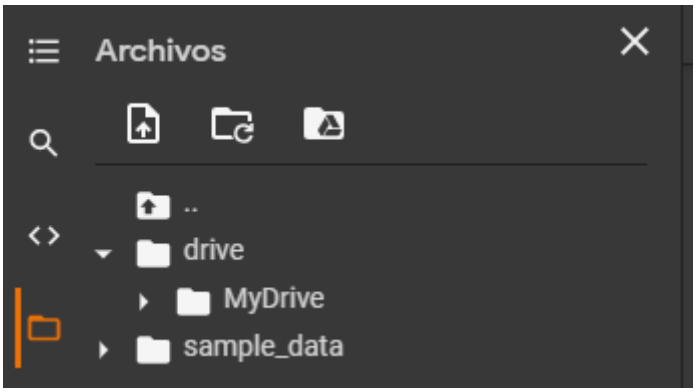


Figura 24. Carpetas Google Colab

Lo único que nos quedaría por hacer sería establecer el entorno de ejecución. Esto se realiza haciendo clic en Entorno de ejecución y posteriormente en Cambiar de tipo de entorno de ejecución.

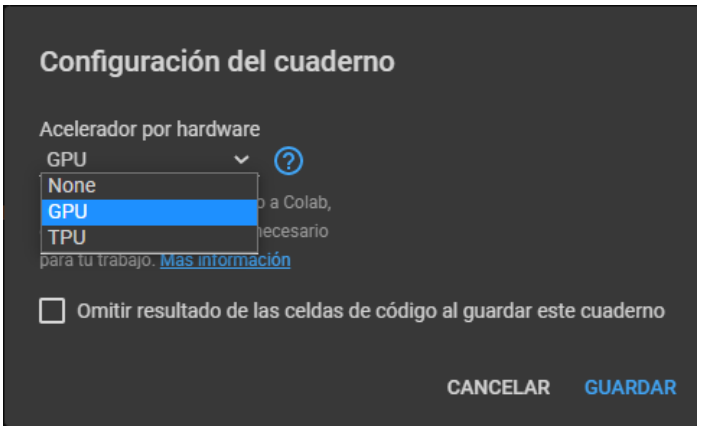


Figura 25. Configuracion del cuaderno Google Colab

Seleccionaríamos GPU y ya tendríamos el notebook preparado para ejecutar el código usando la GPU de Google

3.2 Configuración *Anaconda*

Al contrario que para *Google Colab*, *Anaconda* si requiere de instalación, ya que se ejecuta en local. Para este caso dividiremos la configuración en dos partes: la primera es una parte común, la necesaria para que nos ejecute el programa utilizando una CPU y, la segunda parte si queremos que ejecute la GPU de nuestro dispositivo.

Una vez instalado *Anaconda* lo primero que hay que hacer, no es obligatorio pero si recomendable -puesto que así se podría tener distintos entornos de trabajo con distintas características-, es la creación de un nuevo entorno de trabajo.

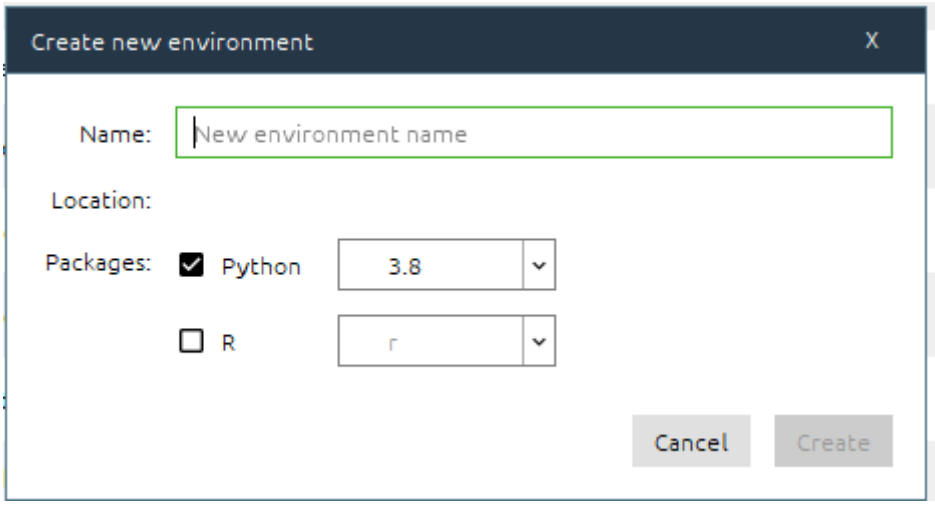


Figura 26. Creación entorno Anaconda

Estableceremos el nombre que queramos en la casilla de nombre y nos fijaremos que esté seleccionado la versión 3.8 de Python. Le damos al botón de crear y esperamos hasta que el entorno esté creado.

El siguiente paso sería instalar las librerías. Para poder tener compatibilidad total con *Google Colab* es necesario saber las versiones de *Keras* y *Tensorflow* de *Colab*.

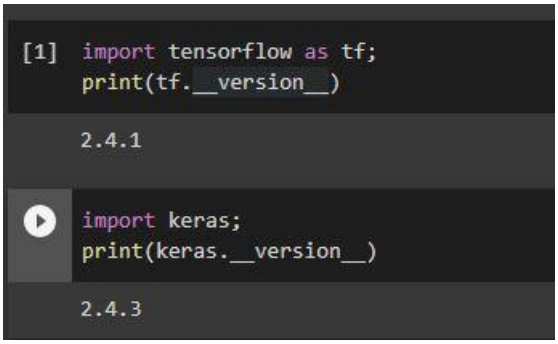


Figura 27. Versiones de Keras y TensorFlow en Google Colab

Una vez sepamos las versiones, ya podremos instalarlo en *Anaconda*, junto a los otros paquetes que necesitaremos en nuestro código, como son opencv y matplotlib.

- Opencv: Acrónimo de *Open Computer Vision*, biblioteca libre empleada en tareas de visión artificial como detección de movimiento o reconocimiento de objetos. Permite la manipulación de imágenes.

- Matplotlib: Librería utilizada para creación de figuras y gráficos.

Figura 28. Paquetes Matplotlib y opencv

Con estos cuatro paquetes adicionales ya podríamos ejecutar el código sin tener problemas. El único inconveniente es que nos iría a una velocidad mucho menor que si lo comparamos con el tiempo de ejecución en *Colab* debido a que estaríamos usando nuestra CPU y no la GPU.

Si queremos guardar el resumen del modelo de forma gráfica, será necesario la instalación de otras dos librerías nuevas como son pydot y graphviz.

- Pydot: Interfaz para graphviz

- Graphviz: Programa para la visualización de gráficos que nos permite representar información estructural.

Figura 29. Paquetes Graphviz y pydot Anaconda

Con las versiones más recientes que tenga, en este caso 2.38 y 1.4.1 respectivamente.

3.2.1 Configuración GPU en *Anaconda*

Para poder usar la GPU es necesario la instalación de otras librerías diferentes como se detallarán a continuación.

Antes de comenzar, es necesario explicar una serie de datos relativo a *Anaconda* y a la GPU de manera que sea más fácil seguir los pasos.

En primer lugar, hay que mencionar que, para poder utilizar la GPU en *Anaconda*, con una versión de *Python* de 3.8, es necesario la instalación de dos paquetes, CUDA y cuDNN.

- -CUDA: Acrónimo de Compute Unified Device Architecture. Arquitectura que permite el uso de GPUs para cálculos matemáticos. Muy útil para proyectos con algoritmos de cálculo intensivo.
- -cuDNN: Acrónimo de CUDA Deep Neural Network. Biblioteca acelerada por GPU para redes neuronales profundas.

GPU

Versión	Versión de Python	Compilador	Herramientas de compilación	cuDNN	CUDA
tensorflow-2.4.0	3.6 a 3.8	GCC 7.3.1	Bazel 3.1.0	8.0	11.0
tensorflow-2.3.0	3.5 a 3.8	GCC 7.3.1	Bazel 3.1.0	7.6	10.1
tensorflow-2.2.0	3.5 a 3.8	GCC 7.3.1	Bazel 2.0.0	7.6	10.1

Figura 30. Versión de TensorFlow para GPU

Como podemos apreciar en la anterior imagen, necesitaríamos una versión de *TensorFlow* entre 2.2 y 2.4. Sin embargo, descartamos la versión 2.4 por dos razones: la primera es que la versión más reciente que podemos alcanzar de *TensorFlow* en *Anaconda* es la versión 2.3.0 y la segunda razón es que cuDNN 8.0 no trabaja en *Anaconda*.

Otra cosa que hay que saber es que *TensorFlow* ya no incluye los paquetes cuDNN y cudatoolkit en la versión 2.3, esto significa que deberemos de instalarlo manualmente.

El método más sencillo para poder instalarlos es yendo a la página oficial de *NVIDIA* en mi caso, y descargar la versión de cuDNN 7.6.5 y la versión CUDA 10.1

Empezaremos instalando CUDA, empezaremos yendo al archivo del conjunto de herramientas CUDA y buscaremos la última actualización de la versión que buscamos

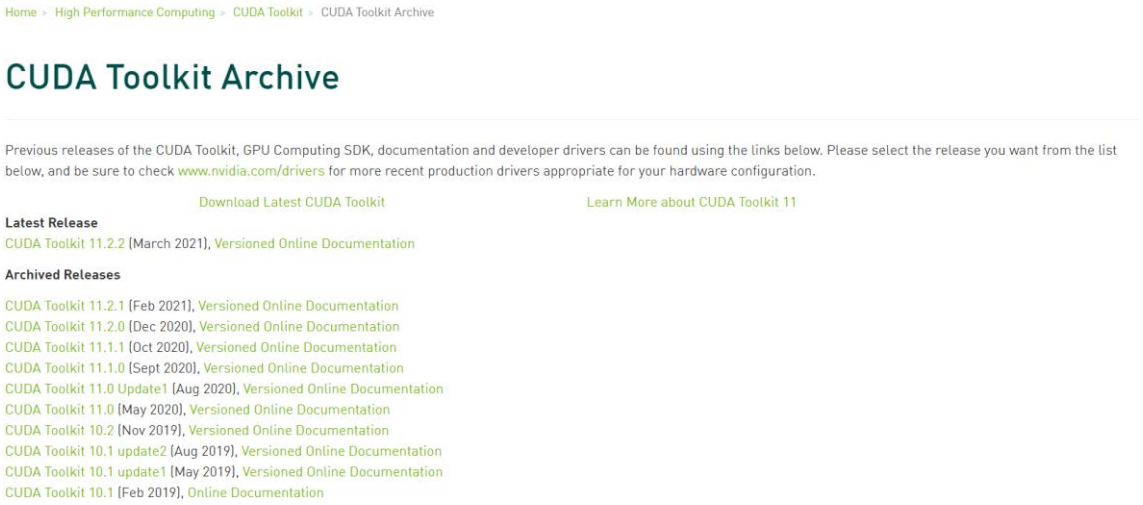


Figura 31. CUDA Toolkit Nvidia

Seleccionamos la versión de agosto 2020 y posteriormente nos descargamos el archivo.

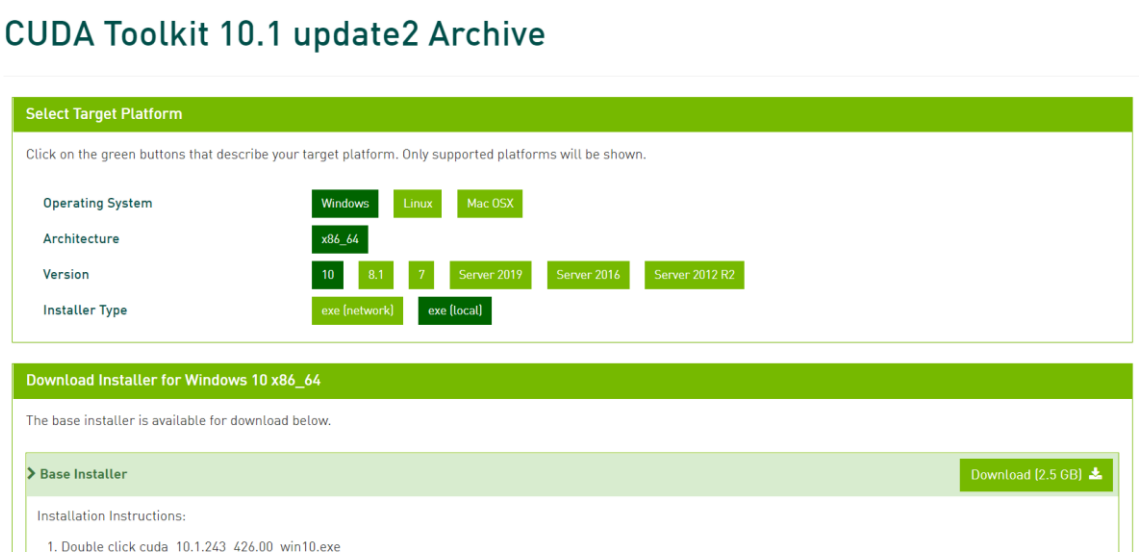


Figura 32. CUDA Toolkit 10.1

Para cuDNN el proceso es bastante similar, la única diferencia es que tienes que unirte al programa de desarrolladores de *NVIDIA* para tener acceso a dichos archivos.

Una vez descargados, instalamos el ejecutable (cuda_10.1). Copiamos el interior de la carpeta de cuda (que se encuentra dentro de la carpeta cudnn), lo pegamos en C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1 y, si es necesario, sobrescribimos los archivos.

Ahora solo necesitaríamos instalar *TensorFlow GPU* en nuestro entorno. El primer paso es instalar CMD.exe Prompt puesto que instalaremos CUDA usando línea de comandos.

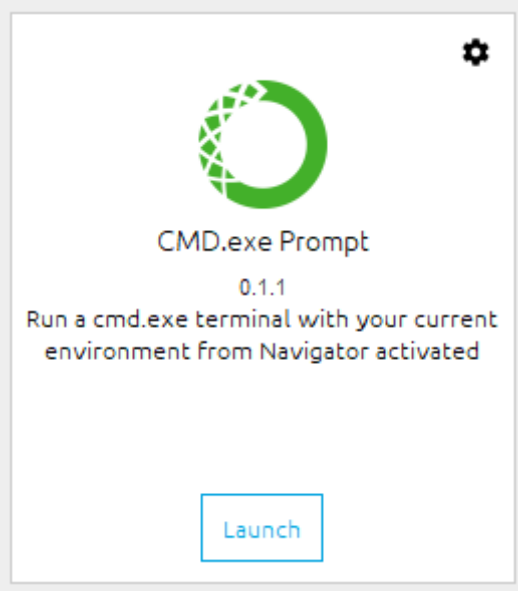


Figura 33. Prompt CMD.exe de Anaconda

Después, tendremos que introducir el comando conda install tensorflow-gpu=2.3 primero y después conda install tensorflow=2.3=mkl_py38h1fcfbd6_0 .Con esto, ya tendríamos configurado correctamente la GPU para en entorno de trabajo creado.

Para comprobar que efectivamente se ha configurado, importaremos *TensorFlow* y mediante un comando print obtendremos la respuesta:

```
print("Num GPUs:" ,len(tf.config.experimental.list_physical_devices('GPU')))
Num GPUs: 1
```

Figura 34. Comprobación del funcionamiento de la GPU

Para el caso de *Anaconda*, es necesario la instalación de tres librerías más:

- **Dlib**: contiene algoritmos de aprendizaje máquina y análisis de imagen. Utiliza detección de *landmarks* para obtención de partes relevantes de la cara.

- **Face_recognition**: utiliza la librería dlib para reconocer y manipular caras.

- **Cmake**: Necesario para controlar el proceso de compilación de software.

Mientras que cmake podemos instalarla desde la propia ventana de Anaconda, dlib y face_recognition se instalaran utilizando CMD.exe prompt, que se realizará siguiendo los siguientes pasos:

1) Para ello deberemos de descargar la librería de github. Dicho archivo fue subido por el usuario “RvTechiNNovate” como solución a un problema de instalación.

    - https://github.com/RvTechiNNovate/face_recog_dlib_file

2) Ejecutar el comando pip install dlib-19.19.0-cp38-cp38-win_amd64.whl para instalar la librería dlib para Python 3.8

3) Ejecutar el comando pip installl face_recognition

4. Desarrollo del software de detección de emociones

En este capítulo detallaremos el desarrollo de un modelo capaz de clasificar imágenes en función de varias expresiones básicas en un rostro: expresión neutra, feliz, triste, ira y sorpresa. Todo ello teniendo en cuenta el denominado sesgo algorítmico mencionado en capítulos anteriores para resolver algunas de las negligencias que provoca la AI.

4.1 Conjunto de datos

Para la elaboración de este proyecto, se hizo uso de tres bases de datos distintas (CK+48 (38), Jonathan Oheix (39) y las heredadas del anterior proyecto realizado por Andrea Martínez) que nos proporcionan, después de haber realizado el filtrado de personas de piel oscuras, un total de 12298 imágenes, las cuales han sido reprocesadas para limitar el rostro, recortarlos y redimensionar la imagen a un tamaño de 48x48.

533 de estas imágenes pertenecen al estado ira, 554 pertenecen al estado sorpresa, 978 imágenes pertenecen a tristeza,5012 pertenecen a la emoción feliz y 5021 corresponden a la emoción neutral. Estas imágenes se han separado en dos bloques, el 80% se han utilizado como entrenamiento y el 20% restante para validación.

En las siguientes figuras se muestra unos ejemplos de las imágenes del dataset para cada tipo de emoción.

Figura 35. Ejemplos de fotos cabreados

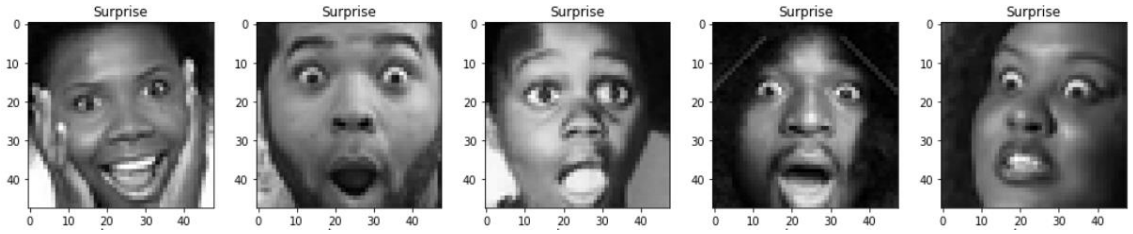


Figura 36. Ejemplos de fotos sorpresa

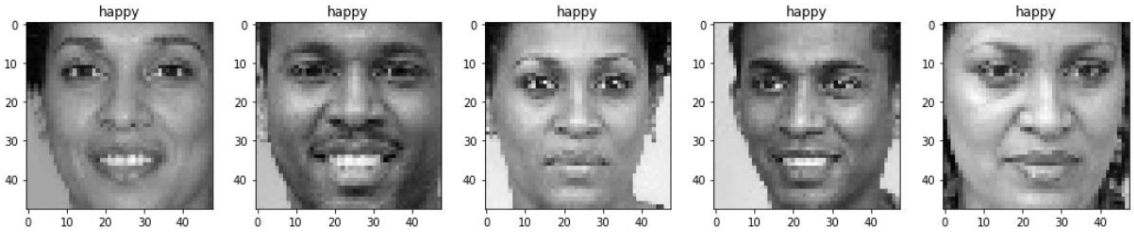


Figura 37. Ejemplos de fotos feliz

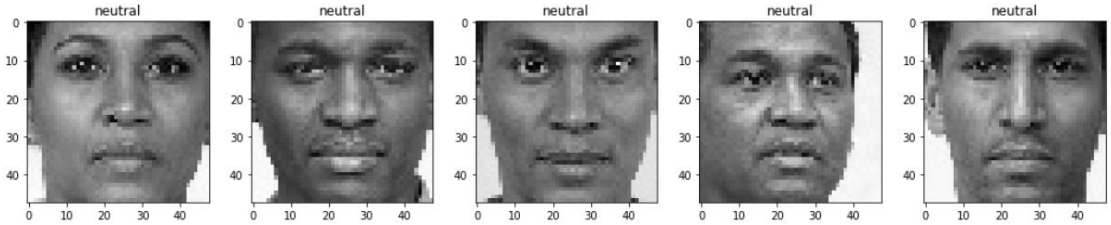


Figura 38. Ejemplos de fotos neutral

Figura 39. Ejemplos de fotos tristeza

Dividiremos el código en dos partes, la primera corresponderá a la obtención de imágenes y entrenamiento del modelo, y la segunda parte, que se corresponderá a las pruebas y los resultados.

4.2 Entrenamiento del modelo

Para realizar el entrenamiento del modelo se ha utilizado la herramienta Anaconda. Utilizaremos la GPU (*Graphic Processing Units*) de una NVIDIA GeForce GTX 1060, tal y como se puede apreciar en la siguiente imagen.

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 842286187061613929
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 14044779575191105410
physical_device_desc: "device: XLA_CPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 5060693856
locality {
  bus_id: 1
  links {
  }
}
incarnation: 18203335877876548353
physical_device_desc: "device: 0, name: NVIDIA GeForce GTX 1060, pci bus id: 0000:01:00.0, compute capability: 6.1"
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 5369660481366819748
physical_device_desc: "device: XLA_GPU device"
]
```

Figura 40. Modelo GPU

La tarjeta gráfica incorpora una unidad de procesamiento que nos permitiría la mejor distribución de la carga de procesos del computado, agilizando así el tiempo del entrenamiento.

Lo primero de todo es importar las librerías necesarias para poder llevar a cabo dicho proyecto.

```
import tensorflow as tf

import keras
from keras import backend as K
from keras import layers
from keras import regularizers

from keras.callbacks import EarlyStopping,ReduceLROnPlateau,ModelCheckpoint

from keras.layers import Conv2D, MaxPool2D, Flatten,Dense,Dropout,BatchNormalization
from keras.models import Sequential
from keras.optimizers import Adam,RMSprop,SGD,Adamax

from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
```

Figura 41. Emotion_Detection.ipynb fragmento 1

Tal y como vemos en la imagen anterior, las bibliotecas que utilizamos para este apartado son keras y Tensorflow: keras para definir las capas de la arquitectura de red y Tensorflow para comprobar si se ejecuta correctamente la GPU y el modelo.

En este punto, también podría añadirse la importación de la biblioteca drive de Google.colab si en vez de en Anaconda el programa se entrenara en Google Colab.

El siguiente paso es definir la ruta donde se encuentran las imágenes, en mi caso:

```
data_train_path='C:/Users/Franches/Desktop/TFE/fotos'
```

Figura 42. Emotion_Detection.ipynb fragmento 2

A continuación, definiremos algunos de los parámetros que utilizaremos tanto para obtención de las imágenes como para realizar el entrenamiento del modelo.

```
epochs = 100
batch_size = 128
img_size = 48
validation_split = 0.2
```

Figura 43. Emotion_Detection.ipynb fragmento 3

-Epoch: Cantidad de etapas de entrenamiento que se realizarán.

-Batch_size: Tamaño del lote de imágenes que se procesará en cada fase de entrenamiento.

-Img_size: Tamaño de la imagen.

-Validation_split: Porcentaje de los datos de entrenamiento que será destinado para la validación, el resto estará destinado para entrenamiento, debe de ser un número entre 0 y 1.

Al tener y utilizar pocas muestras, podemos llevar a cabo un aumento sobre los datos de entrenamiento. Con el siguiente ejemplo, podemos aplicar transformaciones simples para este objetivo.

```
train_datagen = ImageDataGenerator(width_shift_range = 0.1,
                                   height_shift_range = 0.1,
                                   horizontal_flip = True,
                                   rescale = 1./255,
                                   validation_split = validation_split
                                  )
```

Figura 44. Emotion_Detection.ipynb fragmento 4

- Realiza desplazamientos horizontales.
- Realiza desplazamientos verticales.
- Volteamos las entradas aleatoriamente de forma horizontal.
- Cambiamos la escala al factor 1/255.

- Realizamos la división entre el conjunto de entrenamiento y el de validación con el valor que hemos asignado antes.

A continuación, tomaremos la ruta del directorio establecido y generaremos los lotes de las fotos con las modificaciones realizadas. Como parámetros nuevos, tenemos que mencionar:

- Color_mode: Modo de color, estableceremos que son imágenes en escala de grises.

- Class_mode: Establecemos mediante *categorical* que las fotos irán etiquetadas

- Subset: Estableceremos a que subconjunto se refiere, si de entrenamiento o de validación.

```
train_generator = train_datagen.flow_from_directory(directory = data_train_path,
                                                    target_size = (img_size,img_size),
                                                    batch_size = batch_size,
                                                    color_mode = "grayscale",
                                                    class_mode = "categorical",
                                                    subset = "training"
                                                   )
```

Figura 45. Emotion_Detection.ipynb fragmento 5

```
validation_generator = train_datagen.flow_from_directory( directory = data_train_path,
                                                          target_size = (img_size,img_size),
                                                          batch_size = batch_size,
                                                          color_mode = "grayscale",
                                                          class_mode = "categorical",
                                                          subset = "validation"
                                                         )
```

Figura 46. Emotion_Detection.ipynb fragmento 6

Al ejecutar estas dos últimas instrucciones, el programa nos devolverá un *string* indicándonos la cantidad de imágenes que se han encontrado y el total de clases distintas que hay. Cada subcarpeta que exista en ese directorio será considerada como una clase nueva, de este modo se nos facilita la futura ampliación del proyecto.

Guardaremos los directorios de los subconjuntos de datos para usarlos en un futuro.

```
directory_train_generator=train_generator.directory
directory_validation_generator=validation_generator.directory
```

Figura 47. Emotion_Detection.ipynb fragmento 7

Generaremos una nueva variable (name_emotions); que usaremos tanto para la representación de imágenes (almacenaremos la cantidad de veces representada la emoción) como para obtener el orden de las clases, que depende de la forma que tiene de leer los subdirectorios en la carpeta).

```
name_emotions=train_generator.class_indices
print(train_generator.class_indices)
```

Figura 48. Emotion_Detection.ipynb fragmento 8

Representaremos una pequeña muestra de las fotos de ambos subconjuntos a modo de comprobación visual. Con este propósito, primero necesitaremos importar dos colecciones de la librería de matplotlib como son pyplot e image.

- Pyplot : Colección que nos permite representar figuras como si fuera en MATLAB.

- Image : Módulo que nos permite cargar una imagen, cambiar la escala y su visualización.

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
```

Figura 49. Emotion_Detection.ipynb fragmento 9

El objetivo de la siguiente parte es la representación gráfica de las distintas emociones obtenidas de la carpeta data_train_path. Obtendremos tanto la representación de imágenes del conjunto de entrenamiento como las imágenes del conjunto de validación; se explicará solo un código (el de conjunto de entrenamiento) debido a que la diferencia con el de conjunto de validación solo es el origen de las imágenes. El código es el siguiente:

```
fig = plt.figure(figsize=[18,18])
dic = dict.fromkeys(name_emotions,0)
contador=0
for j in range(len(train_generator.class_indices)):
  value=dic.get(list(name_emotions)[j])
  for i in range(train_generator.samples):

      data=train_generator.filenames[i]
      emotion =data.split("\\")
      if (list(dic)[j]==emotion[0]):
        if ((dic.get(list(name_emotions)[j])<5)):
          value=value+1
          dic.update({list(name_emotions)[j]:value})
          route=directory_train_generator+'\\'+data
          testim = mpimg.imread(route)
          ax = fig.add_subplot(len(train_generator.class_indices), 5, contador+1 )
          ax.imshow(testim,cmap = plt.get_cmap('binary_r'))
          ax.set_title(emotion[0])
          contador=contador+1
```

Figura 50. Emotion_Detection.ipynb fragmento 10

El primer paso es crear una figura de tamaño 18x18 para que puedan visualizarse correctamente las imágenes.

El segundo paso es establecer un valor de 0 a todos los contadores de las distintas emociones. El objetivo de esto es que solo queremos representar 5 imágenes de cada tipo.

El tercer paso es la creación de un doble bucle. El primer bucle irá recorriendo todos los elementos de la clase índices de la variable train_generator, con ello nos garantizaremos de que se repita el bucle tantas veces como emociones tengamos. Con el segundo for recorreremos todas las imágenes almacenadas en train_generator, obteniendo la ruta y dividiremos la ruta en dos mediante un Split("\\"). Si aun no hemos representado las 5 imágenes que pretendíamos entraremos en la última parte del código.

El cuarto paso trata de la representación de las imágenes. Para entrar a esta parte, tenemos que cumplir dos condiciones, la primera es que nos encontremos en la carpeta de la emoción que queremos representar. Para una mejor explicación, se abordará la misma mediante un ejemplo:

```
print(list(name_emotions)[0])

data=train_generator.filenames[2]
print(data)

emotion=data.split("\\")
print(emotion[0])
```

```
Angry
Angry\17496.png
Angry
```

Figura 51. Aclaración en la representación de imágenes

La primera emoción almacenada en el array se trata de Angry, la tercera foto almacenada en train_generator se encuentra en Angry\ y se llama 17496.png, puesto que solo nos interesa la carpeta en la que se encuentra lo dividiremos con un simple Split y nos quedaremos con la primera parte (la de la carpeta).

La segunda condición para entrar en esta parte es, como se ha mencionado anteriormente, que no se hayan representados más de 5 veces la emoción.

Una vez dentro, incrementaremos el valor y lo actualizaremos mediante un update. Ahora obtendremos la ruta completa para poder cargar la imagen mediante imread, la añadiremos al subplot, se le establecerá de titulo la emoción correspondiente y se incrementará la variable contador, que es la que indica la posición del subplot que debe de ocupar la imagen.

Antes de entrar a la parte del modelo es necesario saber la convención de formato de datos que seguirá Keras, si van los canales al principio o si van al final. Para ello, nos serviremos de ayuda de un Backend de Keras.

```
if K.image_data_format() == 'channels_first':
    input_shape = (1, 48, 48)
else:
    input_shape = (48, 48, 1)
```

Figura 52. Emotion_Detection.ipynb fragmento 11

Ahora ya tenemos los datos necesarios para la ejecución del modelo (el número de clases que se utiliza para crear la capa densa final e input_shape que se utiliza para crear la capa de entrada).

```
model = Sequential()

model.add(Conv2D(32, (3, 3), padding="same", input_shape=input_shape))
model.add(Activation("relu"))
model.add(BatchNormalization())

model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(128, (3, 3), padding="same",kernel_regularizer=regularizers.l2(0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization())

model.add(Conv2D(128, (3, 3), padding="same",kernel_regularizer=regularizers.l2(0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(512))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(len(train_generator.class_indices)))
model.add(Activation("softmax"))
```

Figura 53. Emotion_Detection.ipynb fragmento 12

Ahora comprobamos el resumen generado utilizando la función summary y compilamos el modelo utilizando un parámetro de pérdidas del tipo binary_crossentropy y un optimizador del tipo rmsprop.

Binary_crossentropy: Función de pérdida utilizada en tareas de clasificación binaria, al ser binaria solo responde con dos opciones, cero o uno.

Rmsprop: Busca mantener un promedio móvil de los cuadrados de gradientes.

También se ha utilizado el módulo utils de keras para poder guardar el modelo en nuestra carpeta.

```
keras.utils.plot_model(model,'model.png',show_shapes=True)
```

Figura 54. Emotion_Detection.ipynb fragmento 13

Antes de empezar el entrenamiento hemos declarado tres devoluciones de llamada (callbacks):

- -EarlyStopping: Detiene el entrenamiento cuando las perdidas dejan de mejorar.

- -ReduceLROnPlateau: Reduce la tasa de aprendizaje cuando considera que la tasa de precisión deja de mejorar.

- -ModelCheckpoint: Guardamos los pesos en una ruta previamente establecida. Se monitoriza la precisión y se guarda solo el modelo que considera mejor.

```
early_stopping = EarlyStopping(
    monitor='val_loss',
    min_delta=0.00008,
    patience=16,
    verbose=1,
    restore_best_weights=True,
)

lr_scheduler = ReduceLROnPlateau(
    monitor='val_accuracy',
    min_delta=0.0001,
    factor=0.4,
    patience=8,
    min_lr=1e-7,
    verbose=1,
)

model_checkpoint_callback = ModelCheckpoint(
    filepath=filepath,
    monitor='val_accuracy',
    mode='max',
    verbose=1,
    save_best_only=True,
)

callbacks = [
    early_stopping,
    lr_scheduler,
    model_checkpoint_callback,

]
```

Figura 55. Emotion_Detection.ipynb fragmento 14

La ruta en la que se guardan los pesos es la siguiente:

```
filepath='C:/Users/Franches/Desktop/TFE/weight/'+str(batch_size)+'-{epoch}-{val_accuracy}.hdf5'
```

Figura 56. Emotion_Detection.ipynb fragmento 15

Como se puede apreciar, el nombre del archivo dependerá de 3 variables: la primera es batch_size que indica el tamaño de bloque escogido para el entrenamiento; la segunda es epoch que indica la etapa donde se ha realizado ese valor de máximo de precisión; la tercera variable indica el valor que tiene ese momento la precisión. Solo batch_size es un valor fijo, puesto que lo establecimos a principio del código, tanto epoch como val_accuracy son valores dinámicos que pueden cambiar en cada etapa.

Una vez configurado el callbacks, ya podemos entrenar el modelo utilizando model.fit.

```
history = model.fit(x = train_generator,epochs = epochs,validation_data = validation_generator,callbacks = callbacks)
```

Figura 57. Emotion_Detection.ipynb fragmento 16

Utilizaremos como datos de entrenamiento los datos reservados para entrenamiento(train_generator), para datos de validación utilizaremos los datos reservados para validación (validation_generator), recordemos que los datos de validación no son utilizados para entrenar el modelo, simplemente se usa para conseguir la precisión de este. El número de etapas está definido por la variable epochs y el valor callbacks es la agrupación de las funciones EarlyStopping, ReduceLROnPlateau y ModelCheckpoint.

Los datos del modelo entrenado lo guardaremos en la variable history para sus posteriores representaciones gráficas.

```
Epoch 78/100
76/76 [==============================] - ETA: 0s - loss: 0.0908 - accuracy: 0.9148
Epoch 00078: val_accuracy improved from 0.91022 to 0.91063, saving model to C:/Users/Franches/Desktop/TFE/weight\128-78-0.910
6330275535583.hdf5
76/76 [==============================] - 25s 324ms/step - loss: 0.0908 - accuracy: 0.9148 - val_loss: 0.0981 - val_accuracy:
0.9106
Epoch 79/100
76/76 [==============================] - ETA: 0s - loss: 0.0902 - accuracy: 0.9170
Epoch 00079: val_accuracy did not improve from 0.91063
76/76 [==============================] - 8s 102ms/step - loss: 0.0902 - accuracy: 0.9170 - val_loss: 0.1160 - val_accuracy:
0.8928
Epoch 80/100
76/76 [==============================] - ETA: 0s - loss: 0.0897 - accuracy: 0.9152
Epoch 00080: val_accuracy did not improve from 0.91063
76/76 [==============================] - 11s 140ms/step - loss: 0.0897 - accuracy: 0.9152 - val_loss: 0.1059 - val_accuracy:
0.8941
```

Figura 58. Emotion_Detection.ipynb fragmento 17

Como podemos ver en la anterior imagen la mejor precisión obtenida sobre los datos de validación se ha producido en el paso 78 y tiene un valor del 91.06%.

Y ya para terminar, solo quedaría la representación gráfica de los valores obtenidos en el entrenamiento. Para ello usaremos matplotlib nuevamente.

```
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
fig.set_size_inches(12,4)

ax[0].plot(history.history['accuracy'])
ax[0].plot(history.history['val_accuracy'])
ax[0].set_title('Training Accuracy vs Validation Accuracy')
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('Epoch')
ax[0].legend(['Train', 'Validation'], loc='upper left')

ax[1].plot(history.history['loss'])
ax[1].plot(history.history['val_loss'])
ax[1].set_title('Training Loss vs Validation Loss')
ax[1].set_ylabel('Loss')
ax[1].set_xlabel('Epoch')
ax[1].legend(['Train', 'Validation'], loc='upper left')

plt.show()
```

Figura 59. Emotion_Detection.ipynb fragmento 18

Como podemos ver en la imagen siguiente, podemos ver el resultado del modelo establecido en el punto anterior y sus comparaciones entre entrenamiento y validación.

A la izquierda tenemos la gráfica que compara la precisión de entrenamiento y la precisión de validación, a la derecha tenemos la gráfica que compara las pérdidas del entrenamiento y las pérdidas de validación. Para ambas gráficas el eje X será el mismo, el número del paso y para el eje Y el valor será el de precisión, para el primer caso, y perdidas para el segundo.

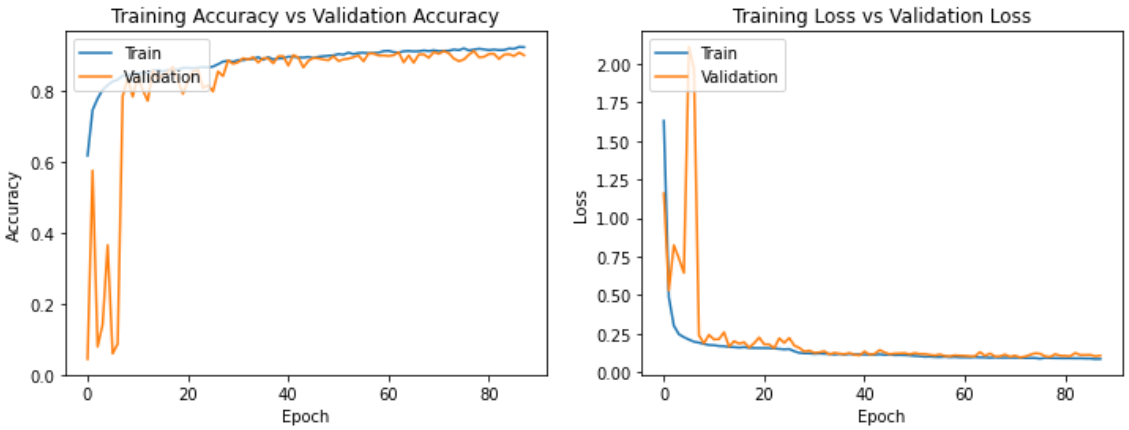


Figura 60. Emotion_Detection.ipynb fragmento 19

También se puede apreciar cómo se estabiliza tanto la precisión como las pérdidas a partir de la época 30 y con un buen resultado en ambos casos, este ha sido uno de los motivos principales por los que se ha elegido este algoritmo. Otro punto a mencionar es que, a pesar de que establecimos que eran 100 épocas, el algoritmo termina en la 88, esto es

debido a que establecimos previamente una parada anticipada para cuando las pérdidas dejaban de mejorar.

4.3 Pruebas y resultados

Para las pruebas vamos a utilizar el modelo entrenado en el punto anterior e imágenes sacadas de internet con un tamaño distinto al 48x48.

El código que nos ayuda a identificar la emoción, recuadrar el rostro con su etiquetado es el heredado por el proyecto anterior junto con unas modificaciones.

Para este paso necesitaremos los paquetes dlib y face_recognition que nos permitirá el reconocimiento facial. Esto, junto al método predict, nos dará la correspondiente emoción y podremos tratarla debidamente.

En primer lugar, se han encapsulado en un bucle if-else posibles errores como imagen no encontrada o la inexistencia de caras en la imágenes. Esto puede ser debido a dos razones. O bien porque no existe la cara o bien porque la cara está muy borrosa para poder detectar los puntos faciales.

El segundo lugar se ha dividido el código en dos partes, una si la imagen de entrada corresponde con un tamaño de 48x48 y otra parte para otro caso, esto se ha realizado así para que quede el recuadro visualmente más atractivo ya que, si se introducía una imagen distinta a 48x48 el texto era muy pequeño y costaba verlo.

Como antes solo disponíamos de dos emociones, con un simple if-else resolvíamos la predicción de la clase y le asignábamos un color. Como ahora disponemos de cinco emociones se ha realizado unas modificaciones en el código para que detecte las nuevas, incluso para que no sea necesario tantas modificaciones futuras en el caso de la adición de nuevas clases de emociones.

Esto se ha solucionado con emotion=(detected_emotions[predicted_class]), las emociones detectadas venían de la expresión:

```
class List(list):
    def push(self, x):
        self.append(x)

detected_emotions = List()
for j in range(len(name_emotions)):
    detected_emotions.push(str(list(name_emotions)[j]))
```

Figura 61. Emotion_Detection.ipynb fragmento 20

Se creará una clase List que nos proporcione la función de indexar un nuevo elemento, en este caso clases. Con ella obtendremos, mediante un bucle for, las emociones existentes en la variable name_emotions. Con este código se facilitará también una futura adición de nuevas emociones.

Para el caso del color se ha hecho de manera similar a la forma que hemos realizado en emotion.

```
color=(colors[predicted_class])
```

Figura 62. Emotion_Detection.ipynb fragmento 21

Los colores estarán previamente almacenados en la variable colors cuyos valores son los siguientes:

```
colors = [(0,0,255), (255,0,0), (0,255,0), (255,0,255), (100,100,100), (60,200,30)]
```

Figura 63. Emotion_Detection.ipynb fragmento 22

Cabe recordar que estos colores siguen la codificación BGR.

Antes de comenzar con los resultados para las distintas fotos de prueba, tenemos que mencionar que nos aparecerán, dependiendo del tamaño de la imagen, entre 3 y 5 print en la consola junto con la foto.

El primer print nos servirá como recordatorio, nos mostrará el nombre de las emociones entrenadas, el segundo print nos indicará los valores de las emociones después de haber realizado el predict. El tercer print nos informará de cuál es el valor de la emoción más probable así como de su porcentaje. Y por último, en el hipotético caso de que el algoritmo tuviera dudas entre el primer valor y el segundo (una relación de valor1/valor2 es menor que 20) nos aparecerá también por pantalla el segundo valor más alto así como también la emoción a la que se corresponde.

Para el caso que la imagen sea distinta de un tamaño 48x48 también nos aparecerá por pantalla la altura y la anchura de la imagen original.

A continuación, pasaremos a los resultados de las imágenes que hemos utilizado para la prueba.

- Ejemplo Angry:

  Para el ejemplo de enfado hemos obtenido los siguientes valores:

  {'Angry': 0, 'Surprise': 1, 'happy': 2, 'neutral': 3, 'sadness': 4}
  [[0.9825454 0.00333003 0.00129673 0.00143859 0.01138924]]

  Como podemos observar el modelo detecta que la foto pertenece a la emoción de enfado con una probabilidad del 98.25% , la segunda emoción posible se trataría de tristeza con un porcentaje del 1.1%

Figura 64. Ejemplo cabreado

- Ejemplo Surprise:

  Para el ejemplo de sorpresa hemos obtenido los siguientes valores:

  {'Angry': 0, 'Surprise': 1, 'happy': 2, 'neutral': 3, 'sadness': 4} [[2.73343013e-03 9.82770324e-01 1.53983404e-08 1.30599865e-05 1.44830942e-02]]

  La emoción más probable es la correspondiente a sorpresa con un 98.28%, seguida de tristeza con un valor de 1.45%

Figura 65. Ejemplo sorpresa

- Ejemplo Happy:

  El siguiente ejemplo que tenemos es el de felicidad. Para esta foto hemos tenido los siguientes resultados:

  {'Angry': 0, 'Surprise': 1, 'happy': 2, 'neutral': 3, 'sadness': 4} [[8.7769957e-05 4.3953734e-04 9.9827766e-01 8.7961473e-04 3.1546195e-04]]

La emoción con mayor porcentaje sería de felicidad, con un valor del 99.82%, la segunda emoción más probable se correspondería con neutral con un valor 0.088%

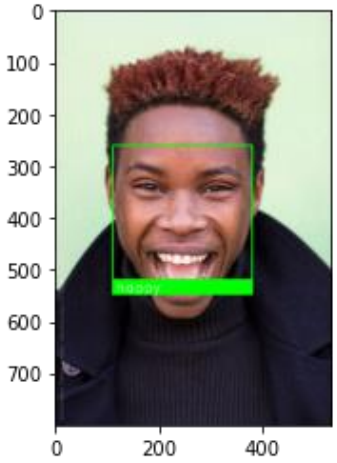


Figura 66. Ejemplo Feliz

- Ejemplo Neutral:

  Para el ejemplo de neutral hemos tenido un resultado más similar.

  {'Angry': 0, 'Surprise': 1, 'happy': 2, 'neutral': 3, 'sadness': 4}
  [[0.19203481 0.09477359 0.00055691 0.38166106 0.33097357]]

  Como podemos ver en el resultado anterior, el mayor porcentaje se obtiene en la emoción neutral con un valor del 38.17%, seguido de tristeza con un 33.1% y también se encuentra cerca enfado, con un 19.2%

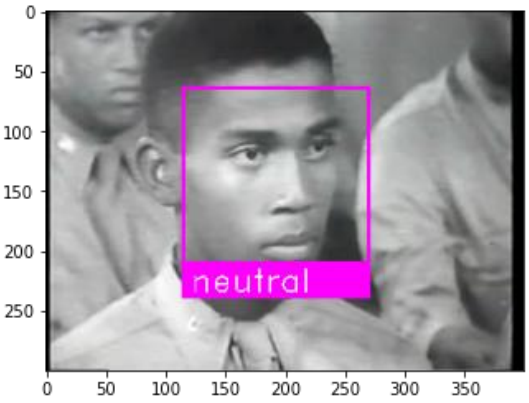


Figura 67. Ejemplo neutral

- Ejemplo Sadness:

  Para el caso de tristeza hemos tenido los siguientes resultados:

  {'Angry': 0, 'Surprise': 1, 'happy': 2, 'neutral': 3, 'sadness': 4}
  [[0.0418949 0.00453055 0.00442774 0.00506747 0.9440794]]

  La emoción predominante en este caso es el de tristeza, con un 94.41%, muy lejos queda la siguiente emoción más probable, enfado, con un 4.19%

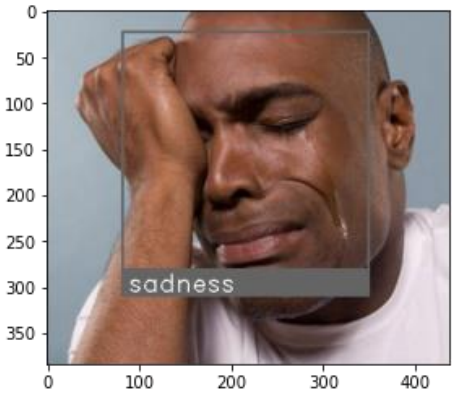


Figura 68. Ejemplo tristeza

- Ejemplo de emociones mixtas

  Para el siguiente caso se ha buscado una foto de una persona, en la que podemos vislumbrar tanto la emoción triste como enfadada. Lo notamos en los labios fruncidos y levemente hacia fuera (signos de tristeza),en las cejas hacia abajo y ojos ligeramente cerrados (signos de enfado). El resultado es el siguiente:

  {'Angry': 0, 'Surprise': 1, 'happy': 2, 'neutral': 3, 'sadness': 4}
  [[4.5953739e-01 2.9126497e-03 2.1592926e-04 1.7955083e-03 5.3553855e-01]]

  Como era de esperar, las emociones más probables son tanto tristeza, con un 53.55%, como enfado, con un 45.95%

Figura 69. Ejemplo emociones mixtas

5. Conclusiones

Este proyecto se ha centrado en el estudio de algoritmos de Deep Learning como método para su posterior uso en el reconocimiento de emociones en personas de piel más oscura. Para ello se parte del trabajo realizado en el Trabajo Fin de Estudios de Andrea Martínez.

Las emociones que se pretenden detectar son, aparte de las heredadas de Feliz y Neutral, Sorpresa, Tristeza y Enfado. Inicialmente, se han tratado de analizar y resumir las técnicas y métodos que se han desarrollado para el reconocimiento de emociones.

Tras el desarrollo y las correspondientes pruebas de este proyecto, se pueden obtener las siguientes conclusiones:

1. En primer lugar, con la identificación del problema, se ha iniciado una búsqueda de imágenes.

2. Se ha estudiado los distintos conjuntos de datos de imágenes para poder distinguir patrones en cada emoción con el fin de poder catalogarlos correctamente, después de ser obtenidos.

3. Se han estudiados diferentes arquitecturas Deep Learning con el fin de encontrar el mejor algoritmo que se adapte a nuestro proyecto.

4. Se ha establecido como mejor opción una arquitectura VGGNet para la clasificación de imágenes.

5. Una vez se ha entrenado la red, hemos obtenido unos valores de precisión de 0.9106 y unas pérdidas de 0.0908.

6. Se ha comprobado, utilizando imágenes de Internet, la efectividad del modelo en personas de color.

El objetivo para el futuro deberá centrarse en seguir aumentando la cantidad de expresiones del modelo, seguir aumentando la base de datos recopilada. Una vez aumentada la base de datos y la cantidad de expresiones, será más factible mejorar la precisión y poder detectar mejor las micro expresiones faciales.

Los conocimientos adquiridos durante la realización de este proyecto se pueden dividir en dos bloques, el primer bloque a nivel académico y un segundo bloque a nivel social.

A nivel académico:

- Manejo de Python y el uso de librerías de alto nivel

- Estudio de las bases teóricas de Deep Learning así como de las arquitecturas más utilizadas en este campo.

- Manejo de herramientas para la realización del proyecto, tanto en local utilizando Anaconda como en la nube utilizando Google Colab, esta última de gran utilidad si no se posee ninguna tarjeta gráfica para acelerar el proceso ya que usaremos una proporcionada con Google.

- Conocimiento, instalación y utilización de GPUs en Python para el desarrollo de algoritmos de Deep Learning.

A nivel social:

- Conocimiento del sesgo algorítmico.

- Estudio de reconocimiento de emociones utilizando la AI, ventajas y desventajas.

A nivel personal, este proyecto me ha aportado conocimiento en el campo de la inteligencia artificial, concretamente en el apartado de aprendizaje supervisado. Esta tecnología posee un amplio mercado y se encuentra en expansión, con lo que ha sido interesante tener una experiencia en este campo. La mayor dificultad ha residido en la obtención de las imágenes, puesto que actualmente los dataset de personas de color están muy limitados. Es por esta razón que se han producido grandes escándalos (40) (41).

Referencias

1. **Kimaldi. [En línea] https://www.kimaldi.com/blog/biometria/reconocimiento_facial/#:~:text=es%20el%20objetivo%3F-,El%20reconocimiento%20facial%20es%20una%20soluci%C3%B3n%20biom%C3%A9trica%20que%20emplea%20un,funci%C3%B3n%20de%20sus%20caracter%C3%ADsticas%20fisiol%C3%B3gicas.&.**

2. Rosebrock, Adrian. Facial landmarks with dlib, OpenCV, and Python. [En línea] 3 de Abril de 2017. https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/.

3. Nilsson, Patricia. Expansion. [En línea] 12 de Marzo de 2018. https://www.expansion.com/economia-digital/innovacion/2018/03/12/5a9e813822601d70088b4625.html.

4. Solares, Claudia. Neuromarketing. [En línea] Noviembre de 2017. https://neuromarketing.la/2017/11/inteligencia-artificial-en-neuromarketing/.

5. Marco-Garcia, S., y otros. Neurologia. [En línea] 1 de Septiembre de 2019. https://www.neurologia.com/articulo/2019047.

6. Rodríguez, Héctor. [En línea] 9 de Febrero de 2021. https://www.nationalgeographic.com.es/ciencia/inteligencia-artificial-emocional-para-maquinas-empaticas_16304.

7. Vincent, James. Theverge. [En línea] 6 de Junio de 2018. https://www.theverge.com/2018/6/6/17433482/ai-automated-surveillance-drones-spot-violent-behavior-crowds.

8. Ng, Alfred. CNET. [En línea] 11 de Agosto de 2020. https://www.cnet.com/news/in-china-facial-recognition-public-shaming-and-control-go-hand-in-hand/#:~:text=%22China%20uses%20facial%20recognition%20to,Mike%20Pompeo%20on%20March%2011..

9. Vincent, James. TheVerge-Emotion-Recognition. [En línea] 25 de Julio de 2019 . https://www.theverge.com/2019/7/25/8929793/emotion-recognition-analysis-ai-machine-learning-facial-expression-review.

10. Crawford, Kate. Nature. [En línea] 6 de Abril de 2021. https://www.nature.com/articles/d41586-021-00868-5.

11. Gutiérrez, Miren. Eldiario. [En línea] 7 de Febrero de 2021. https://www.eldiario.es/tecnologia/sesgos-genero-algoritmos-circulo-perverso-discriminacion-linea-vida-real_129_7198975.html.

12. Abnewswire. [En línea] 8 de Abril de 2021. https://www.abnewswire.com/pressreleases/emotion-detection-and-recognition-market-growing-at-a-cagr-113-key-player-microsoft-apple-google-tobii-intel_537491.html.

13. Naciones Unidas Derechos Humanos. [En línea] 27 de Noviembre de 1978. https://www.ohchr.org/SP/ProfessionalInterest/Pages/RaceAndRacialPrejudice.aspx.

14. Martinez, Naroa y Matite, Helena. Theconversation. [En línea] 10 de Agosto de 2020. https://theconversation.com/discriminacion-racial-en-la-inteligencia-artificial-142334.

15. Wasicek, Armin. [En línea] 11 de Octubre de 2018. https://www.sumologic.com/blog/machine-learning-deep-learning/.

16. Rosenblatt, F. *The perceptron. "A probabilistic model for information storage and organization in the brain".* 1958.

17. Mwandau, Brian y Nyanchama, Matunda. *Investigating Keystroke Dynamics as a Two-Factor Biometric Security.* 2018.

18. *Learning representations by back-propagating errors.* Rumelhart , David E. 533, California : Nature, 9 de Octubre de 1986, Vol. 323.

19. Neuralmagic. [En línea] 08 de Junio de 2019. https://neuralmagic.com/blog/history-gpus/.

20. Redaccion APD. [En línea] 4 de Marzo de 2019. https://www.apd.es/que-es-machine-learning/#:~:text=Machine%20Learning%20o%20Aprendizaje%20autom%C3%A1tico,de%20datos%20en%20su%20sistema..

21. Calvo, Diego. [En línea] 20 de Julio de 2017. https://www.diegocalvo.es/red-neuronal-convolucional/convolucion/.

22. MYO NeuralNet. [En línea] 17 de Febrero de 2020. http://makeyourownneuralnetwork.blogspot.com/2020/02/calculating-output-size-of-convolutions.html.

23. Saxena, Abhineet . [En línea] 29 de Junio de 2016. https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/.

24. Rana, Kartikeya. [En línea] 3 de Abril de 2020. https://ai.plainenglish.io/pooling-layer-beginner-to-intermediate-fa0dbdce80eb.

25. Amidi, Afshine y Amidi, Shervine. [En línea] https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks.

26. Bhattacharyya, Siddhartha. [En línea] Enero de 2011. https://www.researchgate.net/figure/Commonly-used-neural-network-activation-functions-a-Binary-threshold-b-Bipolar_fig1_236268473.

27. OmarAdmin. [En línea] Agosto de 2017. https://forums.fast.ai/t/lesson-2-using-batch-normalization-after-non-linearity-or-before-non-linearity/4817.

28. Oreilly. [En línea] https://www.oreilly.com/library/view/machine-learning-for/9781786469878/252b7560-e262-49c4-9c8f-5b78d2eec420.xhtml.

29. Chatterjee, Aditya. [En línea] https://iq.opengenus.org/evolution-of-cnn-architectures/.

30. Wei, Jerry. Towardsdatascience. [En línea] 3 de Julio de 2019. https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951.

31. Zeiler, Matthew D. y Fergus, Rob. Visualizing and Understanding Convolutional Networks. *Computer Vision – ECCV 2014 .* 2014, págs. 818-833.

32. Tsang, Sik-Ho. [En línea] 19 de Agosto de 2018. https://medium.com/coinmonks/paper-review-of-zfnet-the-winner-of-ilsvlc-2013-image-classification-d1a5a0c45103.

33. Simonyan, Karen y Zisserman, Andrew . *Very Deep Convolutional Networks for Large-Scale Image Recognition.* 2014.

34. Sahu, Anurag . Quora. [En línea] 2020. https://www.quora.com/What-is-the-VGG-neural-network.

35. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2015.7298594.

36. Guo, Zhiling & Chen, Qi & Wu, Guangming & Xu, Yongwei & Shibasaki, Ryosuke & Shao, Xiaowei. (2017). Village Building Identification Based on Ensemble Convolutional Neural Networks. Sensors. 17. 2487. 10.3390/s17112487.

37. He, Kaiming, y otros. Deep Residual Learning for Image Recognition. [En línea] 10 de Diciembre de 2015. https://arxiv.org/abs/1512.03385.

38. *The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression.* Lucey, Patrick, y otros. San Francisco, CA, USA : s.n., 2010.

39. Oheix, Jonathan. [En línea] 3 de Enero de 2019. https://www.kaggle.com/jonathanoheix/face-expression-recognition-dataset.

40. Hern, A. [En línea] 12 de Enero de 2018. https://www.theguardian.com/technology/2018/jan/12/google-racism-ban-gorilla-black-people.

41. VozPopuli. [En línea] 7 de Octubre de 2019. https://www.vozpopuli.com/economia_y_finanzas/google-paraliza-reconocimiento-facial-estafa-contratar-personas-sin-hogar_0_1288971251.html.