Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

Área de Ingeniería Telemática

Emotions Edge: Reconocimiento de emociones usando un SBC y un TPU

D. Prieto Sánchez Ignacio

TUTOR: D. Corcoba Magaña Víctor

FECHA: Enero de 2021

Índice de Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

El avance tecnológico de las últimas décadas ha creado una revolución. Vivimos una era donde dispositivos y tecnología son indispensables en casi cualquier ámbito de la vida cotidiana. Puede verse como una verdadera revolución tecnológica si se compara con lo que existía 200 años atrás. Hoy en día, representa tanto para nosotros que puede ser difícil detenerse a estudiar a qué velocidad crece e innova.

Observando análisis y estudios acerca de estos avances, es sorprendente el crecimiento exponencial que describen. Esto implica una velocidad de crecimiento que se incrementa con el paso del tiempo, naciendo unos siglos atrás y llegando a ser hoy, los pilares sobre los que toda nuestra sociedad está construida.

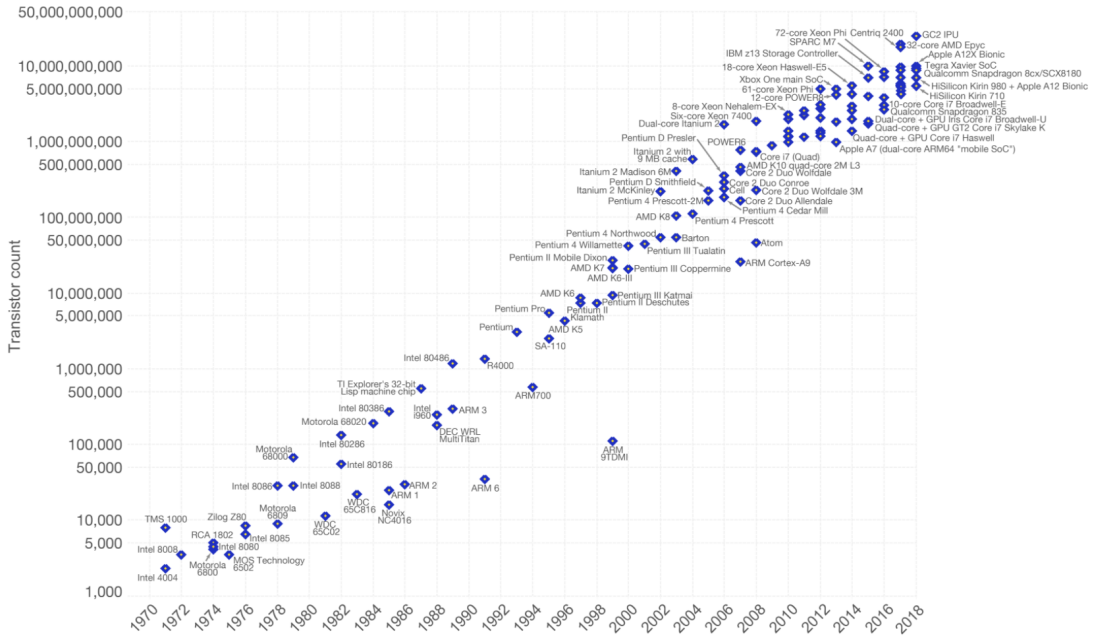


Figura 1.1.- Operaciones en coma flotante por segundo de los procesadores. Fuente: World in Data [9]

En la figura 1.1 se observa un ejemplo en el ámbito de los procesadores, puede analizarse el incremento de la potencia de los procesadores mediante el número de operaciones de punto flotante realizadas por segundo.

Dentro de este concepto encontramos muchos sectores que se encuentran en pleno rendimiento y avance. Las comunicaciones, el automovilismo, la robótica o la inteligencia artificial (IA), son algunos de los sectores que describen este desarrollo tan pronunciado [10].

El mercado tecnológico genera unas ganancias cuantiosas siendo los productos más deseados y cotizados por la población. Mejorar y traer a los clientes los últimos dispositivos o servicios es la motivación que acelera la velocidad de crecimiento y fomenta aún más los avances. Una de las claves, es la pretensión por abarcar más ámbitos de la vida cotidiana de las personas. Es evidente como los dispositivos inteligentes inundan nuestras casas, espacios de trabajo y lugares públicos.

Nace aquí el concepto de Internet de las cosas (IoT): capacidad que poseen los dispositivos de la vida cotidiana para interconectarse a través de internet [11].

Asistentes de voz, robots aspiradoras o automóviles con capacidades asombrosas entran en el mercado con una gran aceptación por parte la población. Planteando una búsqueda del nexo común que tienen estos conceptos, es fácil deducir que hacen la vida más sencilla a las personas.

La inteligencia artificial, adquiere vital importancia en la descripción y el desarrollo de estos productos. Es una idea quizás un tanto abstracta que ha adquirido un gran peso, protagonizando muchas noticias, polémicas o incluso miedos.

1.1.- Motivaciones

Desde el lanzamiento de los asistentes inteligentes de voz y la domótica del hogar, he estado interesado en los dispositivos y funciones que desempeñaban. Explotando todas las funcionalidades que proporcionaban y probando todas las actualizaciones,

comencé a visualizar utilidades y productos que aún no existían. Me planteaba funciones verdaderamente útiles para mí, pero lamentablemente aún no habían sido desarrolladas por las compañías del sector.

Al mismo tiempo, realicé prácticas en una empresa que tenía proyectos para los coches eléctricos. Me fascinaban las capacidades y mejoras que incluían estos vehículos, además de lo positivo que resultaban para el ecosistema. Los proyectos estaban orientados a las estaciones de carga instaladas en la vía pública, con el propósito de crear redes capaces de gestionarlas mediante un sistema común.

Orientar este trabajo a una simulación del proyecto anterior fue mi pretensión inicial, pero desbordaba los objetivos y propósitos para un proyecto de fin de Grado. Mezclando la inteligencia artificial y el mundo del automóvil, surgió la posibilidad de crear un programa ejecutable en un dispositivo embarcado. El tutor responsable de este trabajo, planeaba realizar un proyecto donde estas motivaciones encajaban a la perfección. Su objetivo era incorporar a un simulador de conducción un sistema capaz de medir las emociones que presentaba el conductor a tiempo real. Gracias a esta simulación podría comenzar un estudio acerca de la influencia de las emociones en la conducción.

Me fascinó la idea de emplear la Inteligencia Artificial para elaborar el programa y además ejecutarlo en un dispositivo más pequeño con el objetivo de incluirlo en un automóvil. Una de las principales motivaciones para mi, era realizar un trabajo con proyección futura y encontré exactamente eso en la propuesta del tutor. Él pretendía incluir mi diseño en el futuro simulador, obteniendo provecho de mi trabajo y permitiéndome ayudarle en su futura investigación. Si su trabajo se desarrolla exitosamente, su objetivo es publicarlo en una revista científica.

1.2.- Objetivos

El objetivo principal del proyecto es desarrollar un programa que indique en tiempo real, la emoción que presenta una persona. Se pretende ejecutar el programa en una placa Raspberry Pi de forma que pueda llegar a ser un sistema embarcado, es decir, incluirlo en

el equipamiento de un automóvil. El tutor de este trabajo forma parte de una investigación acerca de la lectura de emociones en los automóviles. En ella un simulador de conducción pretende valorar la influencia que tienen las emociones y la personalidad en la circulación. Por ejemplo, un estado emocional agresivo o triste del conductor puede ser susceptible a producir un accidente. El programa desarrollado en este trabajo, pretende ser utilizado en este simulador, proporcionando la información emocional que requiere.

La predicción de la emoción a partir del rostro se realizará mediante inteligencia artificial, por lo que un primer objetivo ha de ser el estudio de las redes neuronales, los tipos de entrenamientos y las API bajo las que están construidos. El proyecto se sitúa en un marco descrito por el Machine Learning (aprendizaje automático) y el Deep Learning (Aprendizaje profundo).

Se realizará un estudio de la biblioteca de código abierto Tensorflow desarrollada por Google. Bajo este sistema de aprendizaje automático se creará un programa en Python orientado a la creación de redes neuronales y su entrenamiento. Se probarán diferentes metodologías presentes en este ecosistema, pudiendo ser contrastadas las ventajas y desventajas de cada una. A la par de la creación de las redes neuronales, se pretende dotar al script de Python con funcionalidades útiles para ser ejecutado de una forma más eficiente y segura. Como se describirá más adelante, los entrenamientos son procesos en muchas ocasiones largos y con requisitos computacionales altos, lo que plantea el objetivo de conseguir una eficiencia durante dicho proceso.

Los entrenamientos de las redes neuronales otorgarán modelos que podrán ser ejecutados en otros dispositivos, y realizar así la función deseada. Con la construcción del modelo se pretende además generar datos acerca de su precisión para realizar comparativas. Otro objetivo, será plantear herramientas que aceleren estos entrenamientos y bajo qué máquinas podrán ejecutarse.

Tras construir el modelo que realizará las predicciones, el objetivo es lograr un correcto funcionamiento del programa en un dispositivo menos potente como es la Raspberry-Pi 4. Al mismo tiempo, se estudiará el funcionamiento de estos pequeños ordenadores de bajo coste y sus posibles funcionalidades remotas. Con dicho dispositivo se logra un mayor

alcance del proyecto, permitiendo su ejecución en base a la abstracción del entrenamiento inicial, realizándose este en otra máquina más potente. El enfoque descrito, plantea un camino interesante con la filosofía de Edge Computing que será contemplado.

Se ensamblará un prototipo del dispositivo para ser utilizado en el simulador, componiéndose de una cámara acompañada de un panel LED capaz de mostrar las emociones en tiempo real. Otro de los objetivos, será trabajar con el panel LED así como la programación de su electrónica de control.

Los resultados obtenidos en las simulaciones, podrían ilustrar si es posible introducir esta funcionalidad en un automóvil. De esta forma, el conductor sería alertado de que se encuentra en un estado no apto para la conducción, pudiendo ser peligroso para los pasajeros y el entorno que lo rodea. Este sistema, con la eficiencia necesaria, podría salvar vidas.

2. Planificación

En cuanto a la planificación del proyecto, se describirán las distintas etapas en orden cronológico, especificando el tiempo de desempeño para cada una de ellas en el diagrama de Gantt, 2.1. Se distinguen dos entornos de trabajo sobre los que se desarrollará el proyecto.

2.1.- Entornos de trabajo

El **entorno de ejecución** donde se pondrá en funcionamiento el programa.

- **Planteamiento**: en la primera etapa, se selecciona la idea y se define con criterios claros. Entre muchas opciones, se elige el dispositivo donde se ejecutará el proyecto y qué modelo es el más adecuado, la Raspberry-Pi 4.
- **Investigación del entorno**: se estudia el funcionamiento de este ordenador de placa reducida. Se completa la instalación del sistema operativo y funcionalidades útiles a lo largo del proyecto, como SSH o VNC. Además, se seleccionan los accesorios necesarios para el dispositivo: refrigeración, cámara y panel LED. Al finalizar esta etapa, el entorno está listo para ejecutar programas y ser controlado de forma remota.

El **entorno de elaboración del modelo**, donde se realizarán los entrenamientos y pruebas de rendimiento.

- **Investigación de la inteligencia artificial**: se inicia el trabajo con un análisis de los tipos de inteligencia artificial y aprendizaje profundo. Comprender sus diferencias sitúa el marco de trabajo para seleccionar el mejor lenguaje de programación y las API necesarias.
- **Investigación de TensorFlow**: se estudia la API para conocer su funcionamiento a nivel genérico y poder plantear las posibles opciones para su aplicación. Vinculada a esta, se incluyen otras librerías que deberán utilizarse en este marco de desarrollo.

- **Selección del dataset**: se selecciona un conjunto de datos que contenga caras de personas etiquetadas con una emoción, para ser utilizado en los entrenamientos.
- **Construcción de la red neuronal**: se plantean diferentes configuraciones y tipos estudiando la compatibilidad con el objetivo planteado. Tipos de neuronas, número de capas o funciones disponibles son algunos de los ajustes seleccionados en esta etapa.
- **Entrenamiento y recolección de datos**: se ejecutarán diferentes entrenamientos y se mejorará la velocidad de éstos gracias a las herramientas de procesamiento con GPU. Además, se incorporará la herramienta Tensorboard para el estudio a tiempo real de los resultados que vaya generado el entrenamiento.
- **Pruebas de eficiencia de los modelos**: tras los entrenamientos según diferentes estructuras neuronales y parámetros de aprendizaje, se realiza una comparativa prestando especial interés a la precisión y a las pérdidas asociadas a cada entrenamiento.
- **Incorporación del modelo en la Raspberry-Pi 4**: finalmente, se prueba el programa para la identificación de las emociones en el dispositivo ligero utilizando el modelo entrenado con anterioridad. Se configura para reducir la carga computacional y así conseguir proporcionar una experiencia fluida al usuario.

La elaboración de la memoria comenzará tras la elección del marco de trabajo y se desarrollará hasta la finalización de la última etapa.

Planificación del proyecto	
Etapa	**Tarea**
1. Planteamiento	*Definición de objetivos y marco de trabajo*
	Comparativa de placas disponibles
2. Investigación del entorno	*Selección de accesorios y adquisición*
	Instalación del sistema operativo
	Configuración de acceso remoto
3. Investigación Inteligencia Artificial	*Estudio teórico*
	Elección del marco de trabajo
4. Investigación de Tensorflow	*Lectura de un libro para aprendizaje*
	Prueba de ejemplos prácticos
5. Selección del dataset	*Búsqueda del conjunto de datos*
6. Construcción de la red neuronal	*Adaptación de los datos para ser utilizados*
	Elaboración del programa de entrenamiento
7. Entrenamiento y recolección de datos	*Ejecución de entrenamientos*
	Configurar los sistemas de proyección de datos
8. Pruebas de eficiencia de los modelos	*Adaptación del programa para mejorar su eficiencia*
	Comparativa de los datos obtenidos
9. Incorporación en la Raspberry-Pi 4	*Adaptación del programa de predicción*
	Pruebas de funcionamiento final
10. Documentación	*Elaboración de la memoria*

Tabla 2.1.- Tabla de las tareas realizadas en cada etapa

2.2.- Diagrama de Gantt

A continuación, se describen los procesos mediante un diagrama de Gantt, 2.1. Se sitúa la fecha de inicio el 1 de julio y la de finalización, el 31 de octubre.

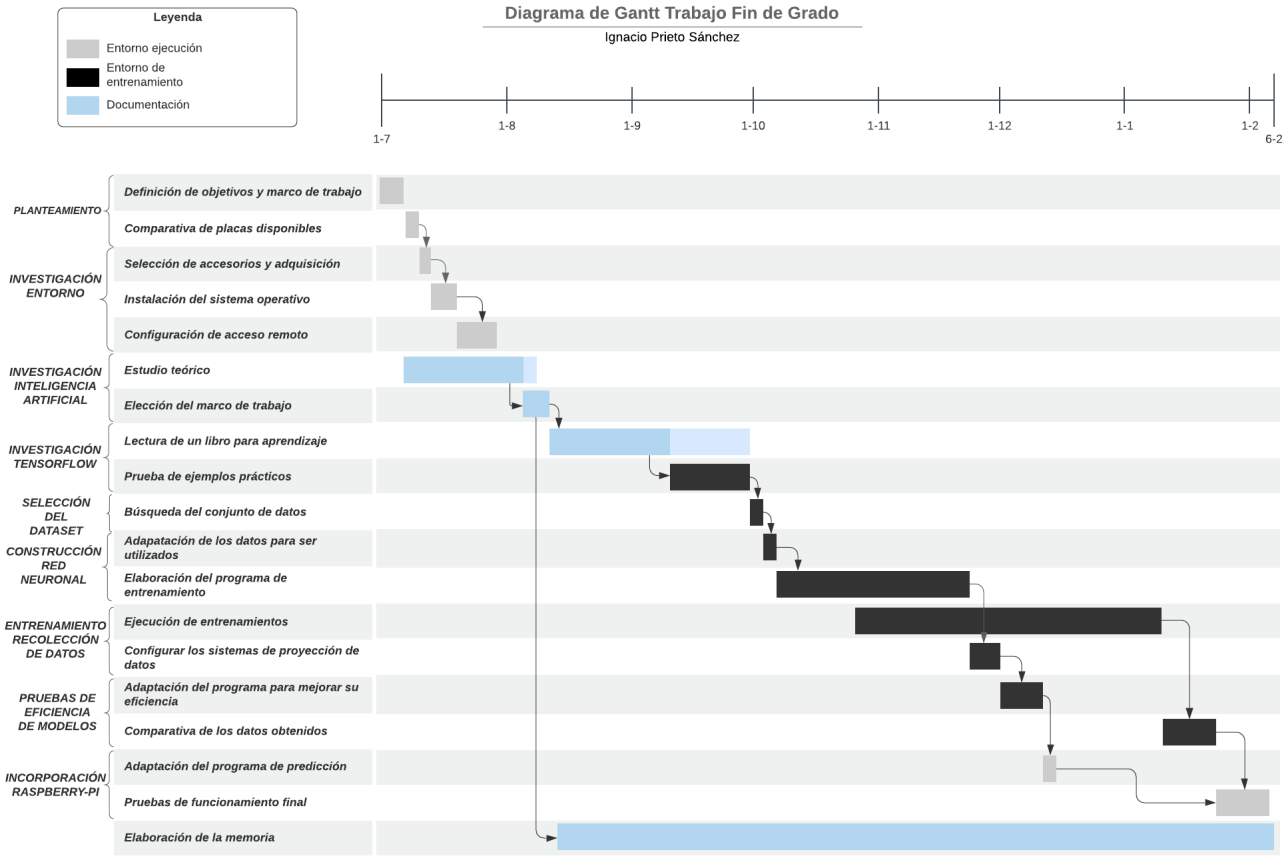


Figura 2.1.- Diagrama de Gantt del proyecto

3. Estudios y análisis previos

3.1.- Estado del arte

Para abordar la tarea de reconocimiento de emociones se plantean dos objetivos principales, la lectura de una cara en una imagen y el reconocimiento de la emoción.

3.1.1.- Identificación facial

La identificación de rostros es un problema muy habitual tratado en numerosos artículos. Se entiende por identificación de rostros reconocer una o más caras en una imagen o vídeo. Existen diferentes algoritmos muy efectivos que permiten solventar con eficiencia este primer dilema del proyecto. A pesar de ello, deberá seleccionarse el más adecuado para los efectos prácticos del proyecto. La librería OpenCV [12], está desarrollada por Intel y es considerada una de las visiones artificiales más eficientes. Incluye algoritmos de Machine Learning que permiten identificar múltiples elementos. Para nuestro proyecto, puede ser de gran utilidad para identificar los rostros de las imágenes que captura la web cam.

La biblioteca Dlib [13], también se emplea en algunos proyectos de reconocimiento de rostros ya que con esta herramienta, es posible trazar puntos en las caras de una imagen. Con este método, se pueden medir las relaciones (distancias o posicionamientos) existentes entre los puntos de un rostro. Conocer esta información de una cara puede ser útil para asociarla a una clase. También emplea Machine Learning para efectuar los reconocimientos.

3.1.2.- Identificación de emociones

Existen bastantes publicaciones relacionadas con el reconocimiento emocional. Los métodos utilizados son muy variados así como la fuente de extracción de los datos. Estudiando algunas publicaciones interesantes, se encontraron proyectos como [14]. En

este, plantean reconocimiento de emociones a partir de la voz. Mediante una señal del habla obtenían un acierto del 63 %. También emplearon métodos PCA para interpretar emociones en imágenes y obtuvieron un acierto del 53,4 %. La eficiencia que conseguían fusionando los dos métodos se incrementaba, siendo de interés buscar si un algoritmo de redes neuronales podría mejorar esa tasa de acierto.

También se estudiaron publicaciones con estrecha relación al proyecto donde se ejecutaban sus predicciones en una Raspberry Pi. En el artículo [2], se trabaja con una Raspberry Pi 3 para capturar imágenes mediante una cámara web. Para la detección de caras emplean la técnica de Viola-Jones [4]. Este algoritmo permite detectar rostros mediante cuatro etapas de selección de características Haar, que se basa en clasificadores en cascada. Tras detectar un rostro, se guarda la imagen con la cara eliminando la información que no es útil. En el preprocesamiento de las imágenes, se pasa a tonos grises y se adapta el tamaño. El resultado será pasado por un filtro Sobel que elimina el ruido. La extracción de características de la imagen se realiza a partir de puntos geométricos fijados en el rostro.

En el artículo [5], utilizan redes neuronales convolucionales profundas para abordar el reconocimiento de las emociones. Resulta interesante que emplearon técnicas de aumento para incrementar la base de datos. De esta forma, puede balancearse la cantidad de datos que se conocen de cada clase. Emplean características de bajo nivel como son la textura, los colores o formas para conseguir información en las primeras capas. Las capas más profundas extraen datos significativos pudiendo finalmente categorizar la emoción presente en la imagen. Los rendimientos de prueba que obtienen son superiores a los métodos tradicionales. Comparan la efectividad del modelo según el número de capas consiguiendo una precisión pico del 75,84 %.

3.1.3.- Influencia emocional en la conducción

La conducción es un proceso muy automatizado que efectuamos día a día. Al haberse convertido en un proceso tan cotidiano, no tenemos una percepción de los diferentes estilos de conducción que podemos presentar. Conducir hacia el trabajo, saber que llegas tarde a un evento o simplemente, tener un mal día, alteran nuestro comportamiento en la carretera.

Existen ya estudios donde se demuestra que los conductores, no son conscientes de la influencia que está teniendo su estado emocional mientras circulan. La falta de control emocional, se resaltó en un estudio efectuado por MAFRE, donde un 84,5 % de los conductores, consideraba que no tenía un correcto control sobre sus emociones [15].

Al volante, tenemos automatizado el análisis de las carreteras, la observación de los otros vehículos... pero se olvida algo muy importante, la percepción de uno mismo. La conclusión clave de estos análisis se encuentra en ese aspecto, analizamos todo el entorno olvidando analizarnos a nosotros mismos.

Cuando el estado emocional no es adecuado, los conductores tienden a tomar decisiones precipitadas, que acaban resultando erróneas. En estos estudios se destaca la importancia de la percepción de uno mismo, lo que es un punto de desarrollo para este proyecto.

3.2.- Planteamiento de los dispositivos

Durante el estudio inicial del proyecto, se pretendía orientar el diseño a un dispositivo con inteligencia artificial con cierto grado de portabilidad. Para hacerlo posible, una de las claves era seleccionar una placa dotada de capacidad para la ejecución del programa y una buena portabilidad. Al mismo tiempo, debería proporcionar escalabilidad para futuras mejoras. Actualizaciones en tiempo real del modelo o recolección de datos en la nube podrían ser funciones interesantes para las que el dispositivo debería estar preparado, cobrando gran importancia la conectividad que proporciona.

Se estudiaron diferentes opciones que cumplían los criterios anteriormente mencionados:

- **Google Coral**: placa destinada al desarrollo con capacidad de un pequeño ordenador. Utiliza un chip Edge TPU que está enfocado a la inteligencia artificial. Además, existe un accesorio capaz de acelerar la inferencia de Machine Learning, el Accelerator USB coral. Este dispositivo se conecta a la placa consiguiendo mejoras del rendimiento, por ejemplo, incrementar los FPS en modelos de clasificación de imágenes. Es compatible con Windows 10, MacOS y Debian Linux, lo que permite su uso en la Raspberry. Esta opción resulta mucho más costosa, tanto la placa como el acelerador triplican el precio de otras opciones.

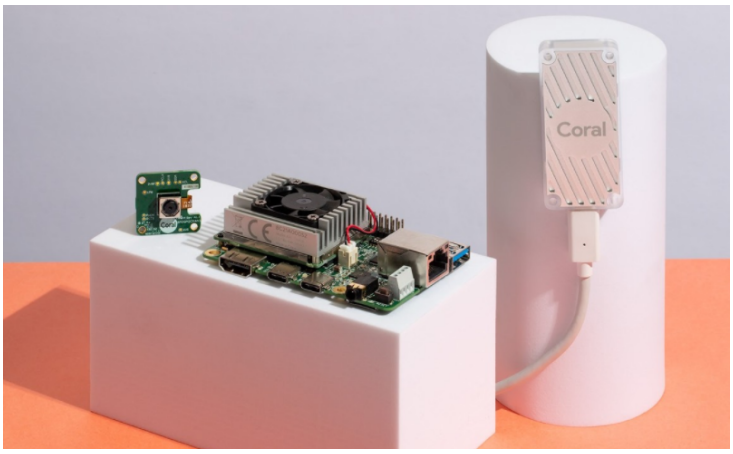


Figura 3.1.- Placa Google Coral y Accelerator USB. Fuente: The Verge [16]

- **Raspberry-Pi 4**: es una placa con funcionalidad de un miniPc. Es un dispositivo muy extendido, por lo que una gran comunidad de usuarios la precede. En concreto, el modelo referido para este proyecto, posee 4 GB de memoria RAM, una tarjeta gráfica Broadcom VideoCore VI y un procesador que trabaja a 1.5GHz de 64-bit. Posee conectividad de red mediante WiFi 802.11ac, Bluetooth 5.0 y un puerto RJ-45. Incluye además puertos USB, salidas de vídeo micro HDMI y audio jack 3.5 mm. El almacenamiento se proporciona a partir de una microSD ya que carece de almacenamiento interno.

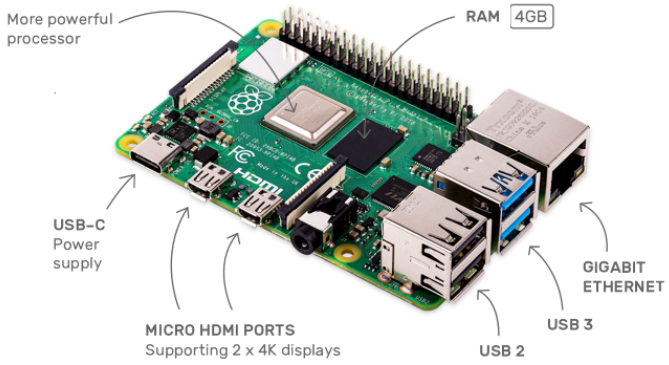


Figura 3.2.- Placa Raspberry-Pi 4 B. Fuente: Web Raspberry [17]

3.3.- MiniPC: Raspberry-Pi 4

Está placa fue seleccionada para la realización del proyecto gracias a su bajo coste. Además, la gran comunidad de usuarios facilita tanto la búsqueda de información como la resolución de futuros errores. Acompañando a la placa, se emplean elementos extra para su protección y refrigeración. Se instalan disipadores de calor sobre los componentes que más se calientan durante el funcionamiento. Para proteger la pieza completa, se realiza el montaje de una carcasa protectora como se aprecia en la figura 3.3. Tras conectar el ventilador que se encuentra en la parte superior, la temperatura de la placa se mantiene en una franja correcta que le proporcionará mayor durabilidad.

Figura 3.3.- Carcasa protectora con sistema de refrigeración

3.3.1.- Sistema operativo y funcionamiento

De forma predeterminada la placa no dispone de un sistema operativo instalado. Dado que no posee memoria interna, deberá cargarse la información necesaria en una microSD para la instalación y el arranque. El sistema operativo utilizado es Raspberry Pi Os, del que se hablará en la sección 3.3.1.1. Desde la página oficial de Raspberry [18], están disponibles tres versiones del sistema operativo:

- **Raspberry Pi OS (32-bit) Lite**: la versión menos pesada, destinada a utilizarse únicamente mediante la terminal.
- **Raspberry Pi OS (32-bit) with desktop**: incluye una interfaz gráfica para utilizarse mediante escritorio.
- **Raspberry Pi OS (32-bit) with desktop and recommended software**: proporciona interfaz gráfica junto con algunos programas funcionales. Es la instalación más pesada.

La instalación que más se ajusta a este proyecto es la de escritorio sin software predeterminado. De esta forma, evitamos programas que no serán utilizados pero tendremos acceso al escrito para facilitar algunas instalaciones y poder modificar código

de forma más visual. Para escribir la imagen en la tarjeta microSD, se utilizará el software gratuito recomendado, BelenaEtcher [19]. Compatible con Windows, MacOS y Linux, este programa permite flashear la memoria y copiar la imagen directamente desde el archivo en formato zip.

3.3.1.1.- Raspberry Pi Os

Este sistema operativo, más conocido como Rasbian, es una distribución de GNU/Linux que está basado en Debian. Su principal enfoque es el aprendizaje y desarrollo para estudiantes y docentes. La elección de este sistema se debe a su gran optimización para la placa utilizada. Característico por ser simple, el sistema ofrece en la interfaz gráfica configuraciones sencillas, por ejemplo, la resolución, la activación de interfaces SSH o VNC...

3.4.- Software y otras funcionalidades

Además de la instalación del sistema operativo, serán de gran utilidad el software que complementa el entorno de ejecución del programa. Se describe a continuación el lenguaje de programación y el intérprete de código empleado.

3.4.0.1.- Python

Python es el lenguaje de programación empleado en el proyecto. Característico por ser interpretado, dinámico y multiplataforma, es uno de los lenguajes más extendidos con utilidad en múltiples campos. Gracias a la posibilidad de importar módulos, encontramos diferentes clases y funciones para cada proyecto, optimizando así el trabajo. [20].

En cuanto al estilo del código se refiere, brinda la posibilidad de utilizar diferentes paradigmas de programación. Tanto la programación orientada a objetos, como la funcional o imperativa pueden llevarse a cabo mediante Python. La versión utilizada en el proyecto es la 3.7.9, compatible con la última versión de Tensorflow, módulo que permite desarrollar redes neuronales.

3.4.0.2.- Visual Studio Code

Visual Studio Code es un editor de código fuente con multitud de funcionalidades. Depuración del código, interpretación de la sintaxis o autocompletado inteligente, son algunas de las utilidades que aporta este software. Está desarrollado por Microsoft, pero puede utilizarse en Linux y MacOs además de en Windows.

Aunque la mayor parte del proceso de elaboración del código se realiza en el PC, es muy útil para ejecutar y depurar los scripts cuando deban ser adaptados a la Raspberry. La depuración del código es una herramienta fundamental para la detección de errores, gracias a ella, es posible monitorizar la ejecución del programa incluso línea a línea si fuera necesario.

Otra característica importante de este programa es que ofrece las funcionalidades a partir de extensiones. De esta forma, pueden añadirse elementos adicionales que faciliten la tarea de programación y optimicen el software. Una extensión con significativa importancia es Remote-Development [21]. Gracias a ella se permite desarrollar programas de forma remota en otras máquinas.

3.4.0.3.- Virtual Network Computing

VNC, computación virtual en red, es un programa orientado al control remoto de un dispositivo. Utiliza un sistema de cliente y servidor, donde el cliente puede observar y controlar el escritorio del equipo servidor. Una de sus utilidades es supervisar el funcionamiento de la Raspberry de forma gráfica. También otorga la posibilidad de revisar la imagen captada por la cámara durante la ejecución del programa.

La conectividad entre cliente y servidor no depende del sistema operativo, lo que permite controlar el sistema Raspberry Pi Os desde un equipo Windows.

3.5.- Protocolos y conexiones

Con el propósito de dotar al proyecto de una mayor aplicación, se ha prestado especial atención al desarrollo remoto. Como ya ha sido mencionado, tanto la supervisión gráfica como la ejecución de una terminal remota, otorgan al dispositivo la capacidad de ser gestionado desde cualquier lugar. Aplicando la configuración de conexión inicial, el dispositivo podría ser colocado en el lugar deseado, consiguiendo efectuar toda su configuración y puesta en marcha a partir de la conexión de red.

Al mismo tiempo, facilita la actualización del algoritmo de predicción sin necesidad de un mantenimiento físico. En las próximas secciones se expone el software utilizado, así como los protocolos participantes en el proceso y su configuración.

3.5.1.- SSH: Secure Shell

El protocolo SSH permite el acceso remoto desde un equipo con rol de cliente, a otro con rol de servidor. Es un protocolo de la familia de Internet que trabaja en la capa de aplicación. Se crea un canal de comunicación seguro entre los dos equipos, permitiendo además la transferencia de archivos simulando una conexión FTP.

Los paquetes enviados mediante SSH están cifrados, de esta forma, si fueran interceptados la información no sería usurpada. La seguridad que aporta este protocolo es una ventaja respecto a los anteriores como Telnet. El cliente es el encargado de iniciar la comunicación y configurar los parámetros de la conexión.

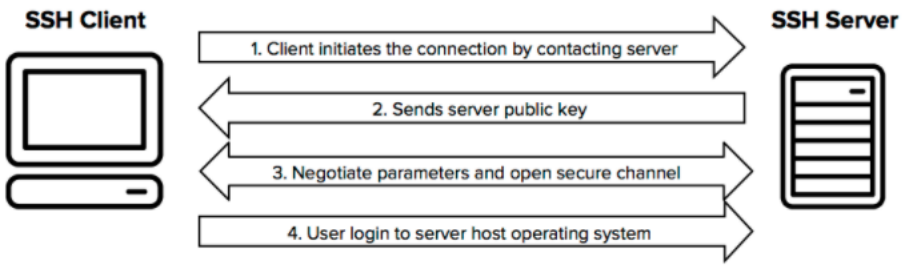


Figura 3.4.- Diagrama de conexión protocolo SSH. Fuente: Web SSH Protocol [22]

Para garantizar una mayor seguridad y añadir más niveles de protección, el protocolo SSH permite diferentes configuraciones para la aceptación de una conexión. Para iniciar la comunicación entre cliente y servidor, se utilizan dos sistemas combinados:

- **Cifrado simétrico**: es el utilizado en el intercambio de información tras haber sido aceptada la conexión. Durante toda la conexión, la información será cifrada conforme a unos parámetros establecidos en una negociación inicial denominada algoritmo de intercambio de claves. Tras este proceso, la información será cifrada acorde a los acuerdos alcanzados y únicamente podrá descifrarse si estos se conocen. Esto significa que interceptar un paquete no sirve de nada ya que se desconocen los acuerdos iniciales.
- **Cifrado asimétrico**: este sistema se utiliza únicamente para garantizar que sólo el cliente y el servidor originales participan en el algoritmo de intercambio de claves. Este cifrado asimétrico garantizará que los parámetros del cifrado simétrico no son interceptados por un usuario fraudulento. En el proceso intervienen dos tipos de claves; pública y privada. Como su propio nombre indica, la clave privada nunca se incluye en los paquetes mientras que la pública se distribuye sin compromiso. La clave pública y privada están vinculadas entre si. De forma que, lo que se cifra con la clave pública, solo puede descifrase con la clave privada vinculada a dicha clave pública que lo encriptó. De esta forma, cifrar utilizando la clave pública es un proceso unidireccional, ya que sólo el equipo conocedor de la clave privada podrá descifrarlo.

Particularmente, en este proyecto se configura la Raspberry y el PC mediante este protocolo. El objetivo es configurar los equipos para que puedan actuar tanto de cliente como de servidor. El PC, como ya ha sido mencionado, podrá gestionar o reparar la Raspberry remotamente. También la Raspberry tendrá la capacidad para ejecutar un entrenamiento derivando su procesamiento al PC. Esta última funcionalidad, se realiza gracias a la herramienta Visual Studio Code explicada en la sección 3.4.0.2.

Se generarán el par de claves, pública y privada en el equipo cliente. La clave pública deberá ser introducida en el archivo “authorized_keys” del equipo servidor. La clave

privada, se mantendrá en el cliente protegida, no siendo nunca compartida. Ahora el servidor reconoce al cliente como usuario de confianza, facilitando la conexión sin necesidad de una contraseña. Esta configuración se repite alternando los roles para permitir las conexiones en ambos sentidos.

Ajeno al usuario, el servidor debe cerciorarse de que el cliente es el auténtico dueño de la clave pública. Al tratarse de una clave que no se mantuvo protegida, cualquier usuario podría tenerla. Sin embargo, ningún usuario que no sea el autentico, tendrá la clave privada.

Para comprobar la autenticidad del cliente, el servidor plantea un desafío encriptado con la clave pública del cliente de confianza. Dicha clave pública ha sido escrita en el archivo "authorized_keys" como ya se mencionó. Si el usuario que intenta conectarse es el auténtico, podrá descifrar el desafío gracias a su clave privada. Tras responder con la solución del desafío, el cliente ha demostrado al servidor su identidad ya que jamás un usuario fraudulento hubiera podido descifrar el desafío.

Con el propósito de agregar más niveles de seguridad, pueden bloquearse los accesos que no procedan de un cliente de confianza o limitar la operativa que el cliente puede efectuar.

3.5.2.- WOL: Wake on LAN

Con propósito de aumentar la conectividad y funcionalidad del proyecto, también se dota al PC con la capacidad de ser encendido remotamente. De esta forma, si se desea ejecutar un entrenamiento podrá ordenarse desde cualquier lugar sin tener que acudir de forma presencial a la ubicación de la máquina para encenderla.

Para el funcionamiento del protocolo, es necesario que la máquina esté conectada a Internet vía Ethernet. También existe un formato para redes inalámbricas pero no es lo habitual. Este protocolo permite que un equipo se encienda a partir de la recepción de un paquete denominado magic packet. Tras configurar esta función en la BIOS y en el sistema operativo, cuando el equipo esté apagado, se mantendrá recibiendo paquetes en bajo consumo.

El magic packet debe contener la dirección MAC del equipo destino además de un número identificativo de la tarjeta de red. Solo un paquete con estas características producirá el arranque de la máquina. Existen múltiples aplicaciones que pueden generar estos paquetes introduciendo previamente los parámetros necesarios.

3.5.3.- DNS: Domain Name System

Es un protocolo de la capa de aplicación ampliamente extendido. Permite la asociación de nombres de dominio a direcciones IP numéricas. En este proyecto se dará cierta conectividad a través de Internet a los dos equipos como se explicará en la sección 3.6. Será de utilidad asociar dominios a las dos direcciones IP mediante un proveedor gratuito. Se emplea la compañía No-IP [23], que proporciona un cliente para actualizar la dirección IP global en caso de que cambie.

Dado el objetivo académico de este proyecto, se prueban ciertas aplicaciones que aportan funcionalidad. Estas pueden suponer un riesgo de seguridad para casos reales. Para el funcionamiento real del programa no son estrictamente necesarias estas características.

3.6.- De LAN a Internet

Todas las funciones descritas en los apartados anteriores funcionan para la red de área local, LAN. La comunicación entre los dispositivos dentro de una misma red es posible gracias al router, que establece las rutas para los paquetes. Bajo el contexto académico se pretende adaptar las funcionalidades ya descriptas en la sección 3.7, para que funcionen vía Internet. De esta forma, no será necesario que los dispositivos se mantengan en la misma ubicación y bajo la misma red local.

Para conseguir que los paquetes viajen a través de Internet y sean recibidos por el equipo destino, debe configurarse el router de la compañía. Además, las IP que poseen los dispositivos dentro de la red local deben ser fijas. Esto es necesario ya que el router, bajo configuración, redirigirá un tipo de paquetes hacia una dirección IP concreta. De forma predeterminada, las direcciones de los dispositivos cambian al conectarse y desconectarse,

ya que son asignadas dinámicamente. Debe reservarse una dirección estática para los dispositivos, consiguiendo que nunca otro equipo sea direccionado a partir de esa IP.

3.6.1.- Redirección de puertos

Es el proceso mediante el cuál un usuario externo a la red local tiene acceso a un puerto de una dirección IP privada. La redirección de puertos, también denominada tunelado de puertos, se realiza en la configuración del router. Para ello, mediante un navegador se accede a la dirección del router que habitualmente es: 192.168.1.1. Introduciendo el usuario y contraseña para administrarlo, se puede acceder a una interfaz gráfica que facilita la configuración.

Posee una sección reservada para introducir entradas en una tabla con las redirecciones de puertos deseadas. Por ejemplo, en el caso del PC se necesitarán dos, ya que se desean utilizar los protocolos WOL y SSH, explicados en la sección 3.7.

Figura 3.5.- Interfaz gráfica de configuración router Vodafone

En la figura 3.7, se observan las dos redirecciones al PC con dirección IP privada: 192.168.0.28. Terminado este proceso, el PC podrá ser encendido desde cualquier lugar con la construcción del magic packet correcta y controlado mediante SSH únicamente por el usuario de confianza. Para un proyecto académico, la redirección de puertos aporta una funcionalidad útil, pero como ya fue mencionado, ha de prestarse especial cuidado

para casos reales ya que puede crear brechas de seguridad importantes. Existen otras alternativas como VPN, que proporcionan mayor seguridad para entornos que lo requieran.

3.7.- PC con potencia gráfica

Uno de los objetivos del proyecto es conseguir ejecutar el programa ligero en un dispositivo de baja capacidad. Este concepto ya está planteado para aplicarse a la Raspberry como se viene describiendo en las secciones anteriores. Puede surgir aquí la cuestión de por qué se utilizan dos dispositivos pudiendo, aparentemente, ser desarrollado y ejecutado el programa únicamente en la Raspberry.

La causa reside en la inteligencia artificial que el programa proporciona. La potencia y funcionalidad no recae en ofrecer un resultado al usuario sino en la precisión de estas predicciones. Es aquí donde puede incorporarse el concepto de inteligencia artificial. No es el hecho de ofrecer un resultado sino el método mediante el que se consiguió. El proceso mediante el cual se obtienen estos resultados se denomina inferencia y se describirá con mucho mayor detalle en próximos capítulos.

La Raspberry puede utilizar el denominado modelo de inteligencia artificial para realizar inferencias y lograr un resultado. Sin embargo, no sería capaz de crear el modelo. La creación del modelo es un proceso computacionalmente costoso que además, en el ámbito de este proyecto, incorpora procesamiento de imágenes lo que supondría un trabajo inasumible para un dispositivo de bajo rendimiento.

Por tanto, es necesario la utilización de un ordenador con mucha mayor capacidad. Entrenar el modelo supone una iteración de procesos, lo que da como resultado una tarea lenta. Aún así, la capacidad y potencia del equipo que lo ejecuta, supone una diferencia drástica en este desempeño. Para este proyecto se emplearon dos equipos diferentes, sus características se detallan en las siguientes tablas 3.1 y 3.2.

Función	Componente	Métrica
Procesador	AMD Ryzen Threadripper 1920X	4 GHz
RAM	Corsair Vengeance 3200 MHz	32 GB
Tarjeta gráfica	NVIDIA GEFORCE RTX 2070 SUPER	Núcleos CUDA: 2560
Placa base	Asus Rog Strix X399-E	
Sistema operativo	Windows 10 Home	64 Bits

Tabla 3.1.- Características del PC destinado al entrenamiento del modelo

Función	Componente	Métrica
Procesador	Intel Core i9-10900K	5.30 GHz
RAM	Corsair Vengeance 3200 MHz	32 GB
Tarjeta gráfica	NVIDIA GEFORCE RTX 3070	Núcleos CUDA: 5888
Placa base	Asus Rog Strix z490-E	
Sistema operativo	Windows 10 Home	64 Bits

Tabla 3.2.- Características del segundo PC 2 destinado al entrenamiento del modelo

De todos los componentes, destacan para el proyecto la memoria RAM y la tarjeta gráfica. Gracias a la RAM pueden procesarse mayor cantidad de imágenes en un mismo lote sin llenar la memoria. La tarjeta gráfica es el motor principal del entrenamiento, acelera sustancialmente el proceso. En la sección 5.6.1, se detalla ampliamente el funcionamiento de este componente.

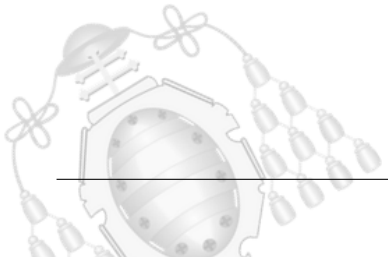


Figura 3.6.- PC con GPU NVIDIA para uso de núcleos CUDA

Figura 3.7.- PC 2 con GPU NVIDIA para uso de núcleos CUDA

4. Inteligencia Artificial

El concepto de Inteligencia Artificial es utilizado habitualmente para describir máquinas o servicios, volviéndose en ocasiones difuso o abstracto. En muchos ámbitos y sectores comienza a aparecer tecnología puntera relacionada con la IA. En ocasiones, su significado se vuelve más difuso complicando la compresión de lo que estas estructuras suponen. En esta sección se introducirá una definición concreta y se desarrollarán diferentes aspectos de interés para el proyecto.

4.1.- Introducción a la Inteligencia Artificial

Se puede definir la Inteligencia Artificial como la capacidad de una máquina para realizar tareas comúnmente consideradas propias de un ser "inteligente". Este término, es habitualmente utilizado para describir proyectos enfocados en desarrollar sistemas con capacidades aparentemente propias de un humano. Razonamiento, identificar significados, aplicar generalizaciones o utilizar experiencias pasadas para casos futuros, son algunas de las características propias de la inteligencia humana.

Simplificando el concepto, la Inteligencia Artificial se crea con la pretensión de lograr que una máquina adquiera características propias de la inteligencia humana. El concepto de inteligencia es extenso y con gran carácter abstracto. Enfocado a la tecnología, se orienta a algunos rasgos concretos: razonamiento, percepción, capacidad resolutiva de problemas y aprendizaje.

Uno de los científicos que participo en el nacimiento de la Inteligencia Artificial fue Alan Turing (1912-1954). En los inicios de este concepto, se pretendía lograr ver a la máquina con capacidades similares a las del ser humano, deseando que estas llegaran a pensar algún día. Pasados 66 años, los sueños de Turing y otros científicos del sector no han sido cumplidos. Los sistemas de aprendizaje y evolución de las máquinas resultaron ser más complejos de lo que imaginaban sus fundadores.

4.1.1.- La prueba de Turing

Las máquinas del siglo XX, no podían compararse con las que hemos logrado crear hoy en día. Sin embargo, Turing planteó ya en aquel entonces una prueba curiosa capaz de evaluar una Inteligencia Artificial. Incluso a día de hoy puede ser complicada de superar, ya que evalúa el grado de naturalidad de las respuestas que otorga una máquina.

La prueba se realiza mediante un proceso con 3 entidades participantes. Dos de los participantes serían personas mientras que el tercero sería una máquina aplicando Inteligencia Artificial. Una de las personas adopta el rol de entrevistador, que interrogará a las otras dos entidades. Este entrevistador sabe que uno de los participantes no es humano, por lo que su misión será identificar de cuál se trata.

Los entrevistados se separan del interrogador y contestarán a las preguntas de forma textual mediante un monitor. Tras 5 minutos de conversación, si el interrogador no lograba identificar a la máquina y a la persona, la Inteligencia Artificial habría superado la prueba.

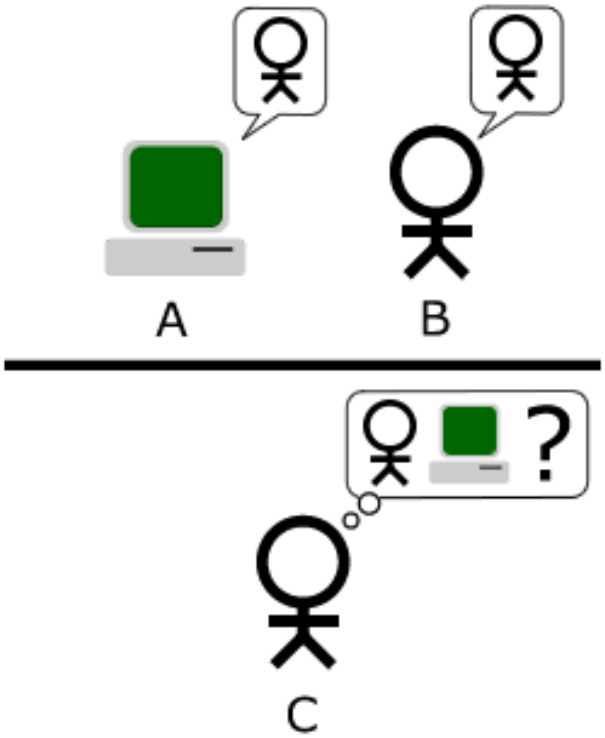


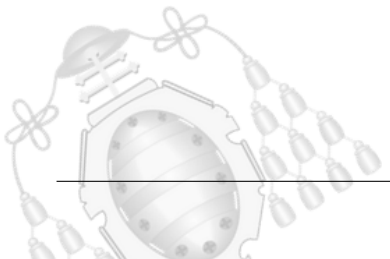
Figura 4.1.- Prueba de Turing para Inteligencia Artificial. Fuente: SearchEnterpriseAI [24]

4.2.- Machine Learning

El concepto del aprendizaje es clave para el funcionamiento de una Inteligencia Artificial. El mayor peso de estos algoritmos recae en la toma de decisión que puede relacionarse directamente con los conocimientos y el proceso para adquirirlos que realiza la máquina. El dilema del aprendizaje se vuelve el verdadero enfoque de trabajo. Los documentos científicos o libros técnicos habitualmente emplean los términos de Machine Learning o Deep Learning, pasando a segundo plano el concepto abstracto de Inteligencia Artificial.

El Machine Learning, conocido también como aprendizaje automático, es la ciencia que trabaja en la programación de máquinas para dotarlas de la capacidad de aprendizaje utilizando datos. Los algoritmos predecesores programaban a las máquinas de forma que ante un evento esperado A, se realizara una acción B conocida. Aparentemente, un programa que interpreta unas respuestas y nos guía a una solución puede parecer inteligente, pero puede no serlo. Tener en cuenta todas las entradas posibles para otorgar una salida es poseer conocimiento, no aprendizaje.

Un sistema de aprendizaje automático utilizará un conjunto de datos para entrenamiento denominado dataset. Cada entrada del conjunto de datos es una instancia de entrenamiento. El conjunto de entrenamiento influye directamente en la calidad del aprendizaje. El sistema presenta bastantes semejanzas con el aprendizaje humano. Por ejemplo, un bebé aprende una lengua gracias al conjunto de datos que le proporciona el entorno. Cada sonido que emite otra persona con intención de comunicar algo, es una instancia más para el conjunto de datos. Cuando el sonido “no”, ha sido escuchado en múltiples ocasiones, los bebes son capaces de relacionarlo con el rechazo y comenzar a utilizarlo. Es evidente que ningún algoritmo de Machine Learning actual puede equipararse a la inteligencia humana pero abre un debate sobre a cuánto pueden evolucionar.



Se plantea un ejemplo para diseñar un algoritmo de detección de SPAM, concepto extraído de [6]. Bajo un diseño tradicional se siguen las siguientes fases:

1. Observación de características del SPAM: búsqueda de palabras habitualmente utilizadas en el SPAM, por ejemplo: oportunidad, oferta o ahorro. Pueden tenerse en cuenta también la inclusión de muchas imágenes o el número de destinatarios del correo.
2. Creación de un algoritmo: se diseña el algoritmo capaz de analizar los correos electrónicos. Además deberá implementarse una lista de reglas para la decisión. En ella se registran todas las entradas que se catalogan como inapropiadas.
3. Testeo: se deberá probar el programa para comprobar su correcto funcionamiento. En caso de ser impreciso, deberá regresar al paso dos donde pueden introducirse más reglas.

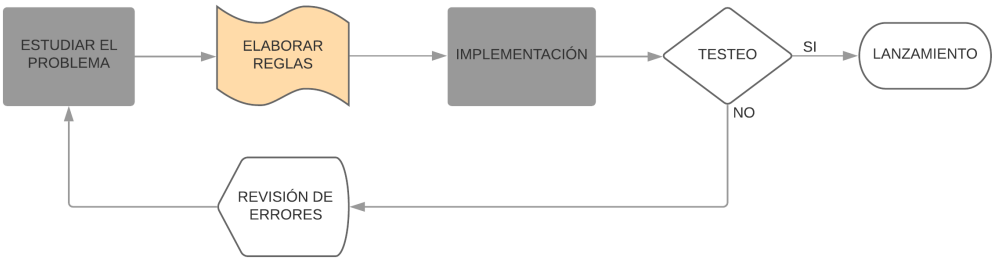


Figura 4.2.- Diseño clásico filtro contra el SPAM

En la figura 4.2, puede observarse el diagrama de flujo de la sección del diseño explicada. El proceso de añadir reglas es muy poco eficiente. Nunca podrán tenerse en cuenta todas las palabras posibles relacionadas con el SPAM, además de que el lenguaje cambia con el tiempo. Un algoritmo clásico debería ser actualizado eternamente, ya que también las empresas de marketing intentarían evitar esas palabras para eludir el filtro.

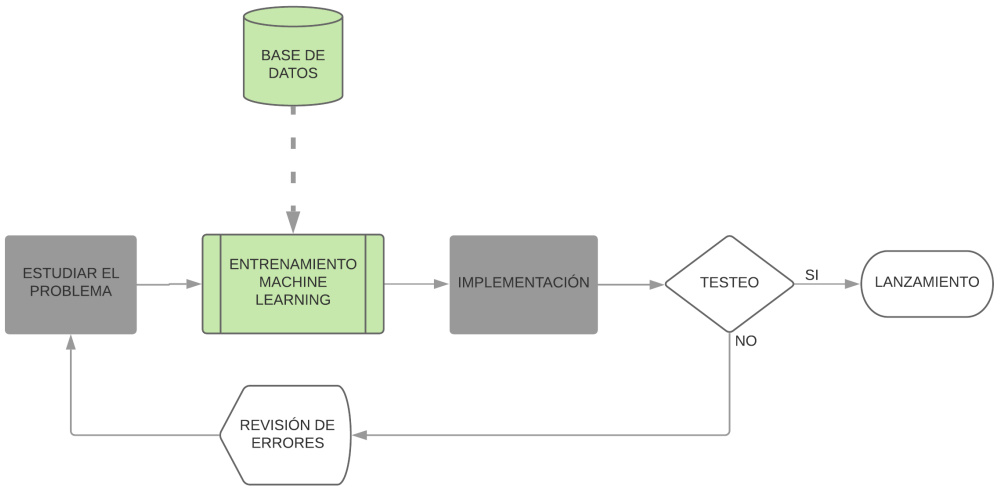


Figura 4.3.- Diseño basado en Machine Learning filtro contra el SPAM

En la figura 4.3 se observa un planteamiento del problema enfocado en Machine Learning. En este se analizaría cada correo de SPAM en busca de patrones comunes. Cada filtrado realizado permitiría al algoritmo aprender parámetros nuevos para interceptar cada vez mejor los correos. El cambio del lenguaje o de remitente producirá un cierto engaño temporal al filtro, pero tras analizar algunas entradas, volvería a encontrar los nuevos patrones comunes mejorando aún más su eficiencia.

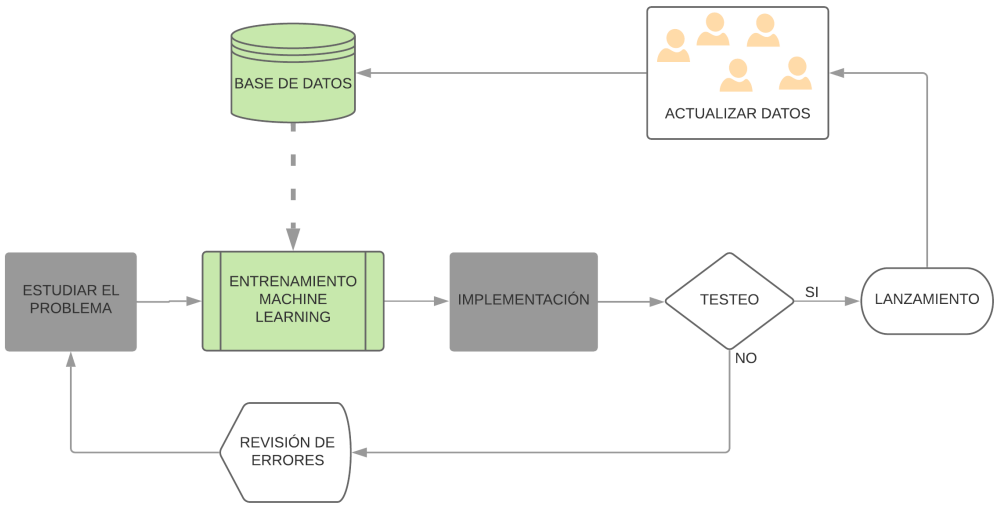


Figura 4.4.- Diseño basado en Machine Learning adaptación automática filtro contra el SPAM

Finalmente, puede visualizarse en la figura 4.4 como el filtro adquiere mucho mayor potencial si el resto de usuarios proporcionan a una base de datos común todos los parámetros recolectados. De esta forma, el dataset incrementa considerablemente su tamaño, permitiendo al modelo realizar un entrenamiento más completo. El algoritmo consigue un estado de automatización que supera con creces un diseño clásico como el planteado al inicio.

4.2.1.- Tipos de Machine Learning

Las características de los algoritmos de aprendizaje automático son muy diversas. Con el propósito de clasificarlos, pueden agruparse bajo las siguientes características:

- Autosuficiencia en el entrenamiento: el aprendizaje puede requerir en mayor o en menor medida la asistencia humana para realizarse. Esta característica se refiere generalmente a la forma de entrada de datos, si el algoritmo solo es capaz de aprender a partir de datos etiquetados por personas o si, por el contrario, puede incluir en el entrenamiento instancias que el mismo etiquetó.
- La forma de aprendizaje: algunos algoritmos se mantienen en constante aprendizaje mientras que otros deben iniciar un proceso de entrenamiento que producirá el nuevo conocimiento.
- Método para la predicción: una predicción de una entrada nueva puede realizarse mediante de la memorización previa de las anteriores instancias o habiendo creado un modelo que permita generalizar a todos los casos.

En relación a la autosuficiencia de los modelos, pueden distinguirse los siguientes algoritmos:

4.2.1.1.- Aprendizaje supervisado

Los algoritmos de aprendizaje supervisado se entrenan con un conjunto de datos que está etiquetado con la solución deseada para cada instancia. Todos los tipos de aprendizaje

presentan ventajas y desventajas, resultando muy eficientes para algunos casos y poco eficaces para otros.

Para obtener buenos resultados con el aprendizaje supervisado es esencial que el conjunto de datos esté bien etiquetado y tenga las dimensiones apropiadas. El algoritmo localiza el problema, la solución y las características importantes. Tras iterar sobre el conjunto de datos, encuentra relaciones entre las etiquetas indicadas por las personas y las características de los datos de entrada. De esta forma, al finalizar el entrenamiento, el algoritmo produce un modelo. En él, se establece la importancia de cada posible característica que puede presentar la entrada desconocida con vistas a decidir qué etiqueta le corresponde.

Un conjunto de datos pequeño no logrará un aprendizaje correcto ya que puede ser complicado generalizar. Si por ejemplo se entrena el filtro anti SPAM con únicamente 10 correos publicitarios, el resultado posiblemente sea muy particularizado a esos casos. Sin embargo, si el conjunto de entrenamiento posee 10.000 instancias, las detecciones del filtro serán mucho más generalizadas y, por tanto, eficientes.

Tras crear un modelo capaz de clasificar los datos, el algoritmo puede continuar descubriendo nuevas características y relaciones, mejorando así tras su implementación. En Machine Learning, se denomina inferencia al proceso de predecir una solución utilizando el modelo creado.

Los algoritmos de regresión lineal, regresión logística, árboles de decisión o redes neuronales utilizan aprendizaje supervisado. Se detallarán sus características en el capítulo 4.2.2.

Matemáticamente, el aprendizaje supervisado busca reducir el riesgo empírico [25]. Durante el aprendizaje el algoritmo busca una función g a partir del conjunto de datos de entrenamiento donde cada instancia es un punto (x, g(x)). Los datos del dataset son variables aleatorias con independencia entre sí y cuya distribución de probabilidad p, es desconocida.

La función de pérdida L, viene dada por la expresión:

$$L : Y \times Y \rightarrow \mathbb{R}^{\geq 0} \tag{4.1}$$

En la que Y representa el dominio de la función g. La expresión L(z,y) representa la pérdida que se produce en la predicción al obtener el valor z a partir de g cuando el correcto es y.

La siguiente expresión representa el riesgo de una función f que se describe a partir de la esperanza de la función L:

$$R(f) = \sum_{i} L(f(x_i), g(x_i))p(x_i) \tag{4.2}$$

Como p, que representaba la distribución de probabilidad, es continua, es posible sustituir el sumatorio por una integral. Para minimizar el riesgo, el objetivo será hallar una función f* de entre un conjunto de funciones, de forma que R(f*) sea mínima. La búsqueda de la función f* se denomina en estadística minimización empírica del riesgo.

Como g viene representado por un conjunto de entrenamiento finito, uno sólo puede realizar una aproximación del riesgo. Por ejemplo:

$$\widetilde{R}_n(f) = \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) \tag{4.3}$$

4.2.1.2.- Aprendizaje no supervisado

En los modelos de aprendizaje no supervisado, a diferencia del anterior, el conjunto de datos no está etiquetado por humanos. El algoritmo será capaz de aprender sin la necesidad de que un humano le aporte las soluciones de los datos. Este formato donde el conjunto de datos está sin etiquetar, se asemeja más a cómo se encuentran los problemas en los casos reales. Además, en muchas circunstancias, etiquetar los datos puede ser un proceso muy tedioso, incluso inasumible. Con el propósito de afrontar este tipo de situaciones, se diseñan los algoritmos de aprendizaje no supervisados [26].

El objetivo es conseguir que el algoritmo sea capaz de identificar patrones e información por sí solo de forma que le permita agrupar los datos con características comunes. Los algoritmos de aprendizaje no supervisado habitualmente presentan las siguientes técnicas:

- **Clustering**: se denomina cluster a un grupo o conjunto de datos. Este método consiste en dividir el conjunto de datos y organizarlo en grupos que presentan características semejantes. De esta forma, el algoritmo detecta patrones comunes entre los datos y es capaz de identificar que existen relaciones entre ellos. Los datos no están etiquetados, pero tras analizar todo un conjunto de datos, el algoritmo los habrá organizado en n grupos que considera que representan lo mismo. El proceso de clustering puede ser más favorable en unas situaciones que en otras ya que de un mismo conjunto de datos pueden existir varias organizaciones posibles. Seleccionando la técnica de división o el número de grupos deseado puede encaminarse el proceso al reparto esperado.
- **Detección de anomalías**: tras analizar los datos, es posible que aparezcan instancias atípicas que no pertenecen a ningún grupo. La capacidad de que el algoritmo detecte estas anomalías pude ser en sí mismo una utilidad para identificar elementos no deseados dentro de un conjunto de datos.
- **Reglas de asociación**: al minar un conjunto de datos, además de identificar los grupos, el algoritmo puede identificar una relación entre elementos de diferentes grupos. Este método, es verdaderamente útil para el marketing y la publicidad. Por ejemplo, dentro de un conjunto de datos de ventas, puede detectar que existe una relación clara entre los clientes que compraron una lámpara y una bombilla. Tras comprar un producto A, la asociación de reglas sugiere mostrar publicidad del producto B ya que es altamente probable que sea adquirido.
- **Reducción de dimensionalidad**: facilita filtrar un conjunto de datos para eliminar la información irrelevante. Descompone el conjunto de datos para extraer las características que se desean. Por ejemplo, si se pretende analizar caras de personas, posiblemente en el conjunto de imágenes aparezcan muchos otros elementos innecesarios. El algoritmo extrae las características determinantes que influirán en la solución desechando el resto.

4.2.1.3.- Aprendizaje semi supervisado

También existe la posibilidad de combinar las dos versiones de aprendizaje anteriores. Muchas de las situaciones donde un algoritmo de aprendizaje automático pretende resolver un problema, no pueden completarse con una única técnica. Por ello, en este tipo de casos, una pequeña parte de los datos se presentarán etiquetados mientras que el resto no lo estarán.

Por ejemplo, si se desea identificar a una persona en un conjunto de fotos. El algoritmo habrá agrupado todas las imágenes donde esta aparece pero necesita que, al menos, una de las instancias esté etiquetada para darle nombre al grupo. De esta forma, el elemento anónimo que se identificaba, pasa ahora a ser una persona con identidad.

4.2.1.4.- Aprendizaje reforzado

Este tipo de aprendizaje enseña al algoritmo denominado agente, a tomar buenas decisiones y alcanzar el resultado correcto. El sistema de aprendizaje se efectúa a partir de recompensas ante las acciones correctas y castigo ante los errores. Para alcanzar un resultado, el algoritmo ha de tomar numerosas decisiones dentro de un espacio de acción que se le plantea en cada paso.

En el proceso de conseguir la solución, el agente realizará acciones correctas que desencadenarán un aumento de la recompensa. También tomará decisiones equivocadas que producirán una reducción en la recompensa siendo así castigado por el error. En el aprendizaje, influyen muchos elementos complejos como el tiempo o los recursos. También ante una decisión poco eficiente o que plantea un camino lento, el agente será castigado.

Por ejemplo, puede desarrollarse un bot para jugar al tres en raya. El agente será el bot que juega contra un rival. El tablero del juego será el ambiente del problema, la acción el movimiento de la ficha y la recompensa, la victoria de la partida. En la figura 4.5 se observa que el bot realiza una acción incorrecta que causa la derrota. Como repercusión de la acción incorrecta, se restan puntos de recompensa.

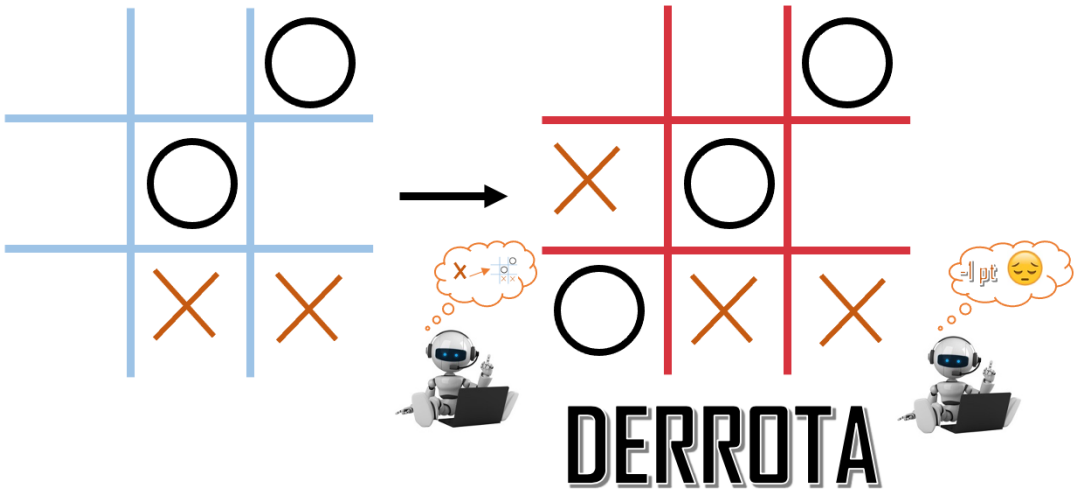


Figura 4.5.- Aprendizaje reforzado acción errónea

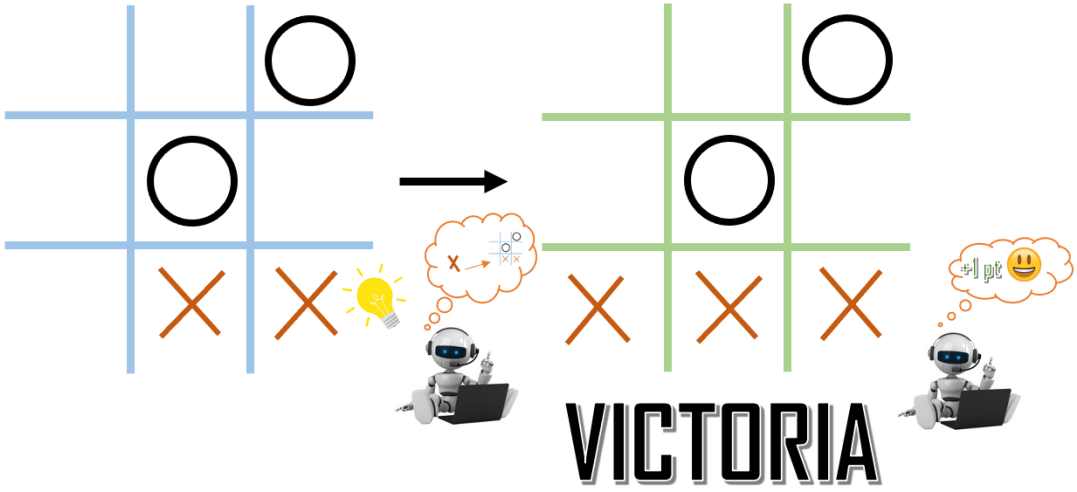


Figura 4.6.- Aprendizaje reforzado acción correcta

En la figura 4.6, el bot sabe que el camino tomado en la primera partida ante esa situación fue penalizado. Por lo tanto decide tomar un camino nuevo, que produce la victoria y un incremento de la recompensa.

4.2.1.5.- Aprendizaje multitarea

En los algoritmos de aprendizaje multitarea, el aprendizaje es múltiple. Diferentes tareas de aprendizaje son resueltas al mismo tiempo. Esta estrategia facilita al algoritmo utilizar

los puntos comunes y las diferencias entre las tareas logrando un mejor resultado final. Los modelos creados bajo entrenamiento de multitarea son mucho más precisos que si hubieran sido entrenados por separado.

Por ejemplo, el filtro anti SPAM basado en aprendizaje automático. Si se aplica un algoritmo de multitarea, el resultado es mucho más eficiente. El entrenamiento individual de un modelo que podría realizar un único cliente es poco efectivo. Sin embargo, si se realiza un entrenamiento múltiple entre todos los clientes que utilizan el filtro, los resultados serán significativamente mejores.

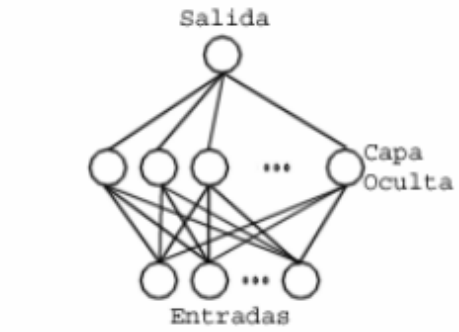


Figura 4.7.- Aprendizaje simple. Fuente: Proyecto Aprendizaje Multitarea [27].

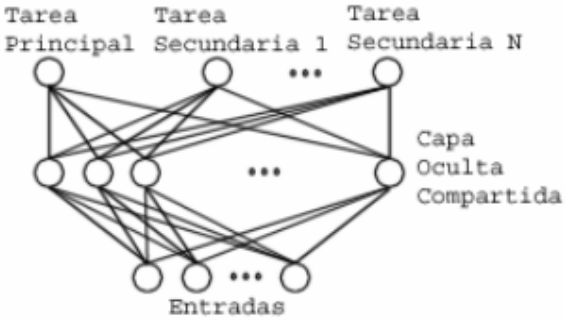


Figura 4.8.- Aprendizaje multitarea. Fuente: Proyecto Aprendizaje Multitarea [27].

En la figura 4.7, se observa el esquema de una red neuronal para aprendizaje simple. Existe una única tarea que desempeñar por lo que la salida es única. Las capas intermedias entre las entradas y la salida, están destinadas únicamente a resolver una tarea. Sin embargo, en la figura 4.8 que representa el aprendizaje multitarea, las salidas son múltiples. Las

capas intermedias, son compartidas para resolver diferentes tareas. La información que aporta resolver las tareas secundarias, marca el camino que debe construirse para alcanzar la solución principal.

4.2.2.- Algoritmos de Machine Learning

En la sección 4.2.1, se describieron los diferentes enfoques que pueden tomar los algoritmos de aprendizaje automático. En esta sección se describirán los algoritmos más habituales para el aprendizaje.

4.2.2.1.- Algoritmo de regresión

El algoritmo de regresión pertenece al grupo de aprendizaje supervisado. Como su nombre indica, es una extensión del concepto estadístico de la regresión aplicado al aprendizaje. El caso más simple de regresión, es aquel donde sólo existe una variable dependiente y una independiente con relación lineal. Para dicho caso, la solución buscada es una recta aproximada que pueda modelar la relación existente entre las dos variables.

Para construir la regresión lineal, el algoritmo empleará un elemento clave en el Machine Learning, el error cuadrático medio (ECM). Esta métrica es utilizada en estadística y representa el promedio de los errores cometidos al cuadrado. En el caso del aprendizaje automático, será la función clave que utilizará el algoritmo para conocer la eficiencia de su modelo y mejorarlo.

Por ejemplo, puede plantearse un problema básico donde se pretende conocer la nota de un examen, evaluado sobre 20, respecto a los días dedicados a su estudio. En la figura 4.9, pueden observarse los puntos naranjas que representan la nota en el eje vertical, en función de los días que emplearon para estudiar. La recta buscada se rige por la siguiente expresión:

$$y = w_0 + w_1 * x_1 \tag{4.4}$$

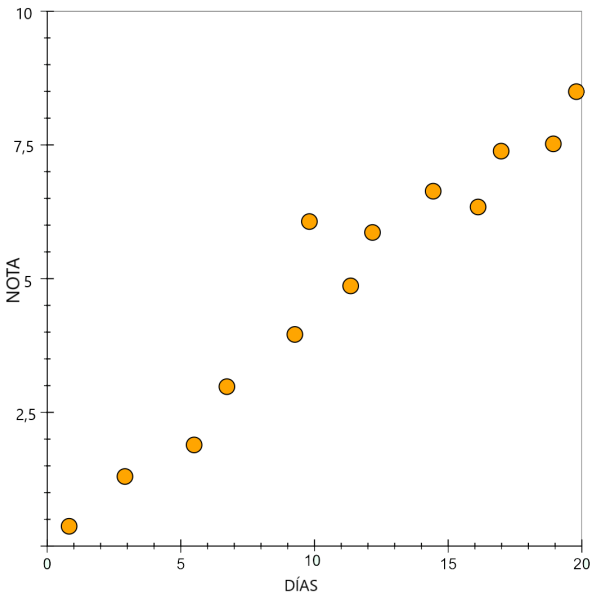


Figura 4.9.- Puntos del conjunto de datos

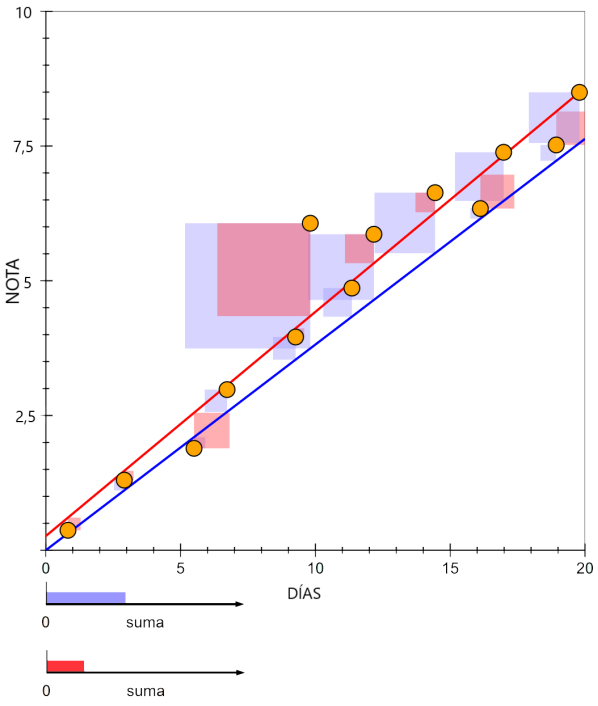


Figura 4.10.- Rectas de regresión

En la figura 4.10, puede observarse en color rojo la recta que minimiza el error y en color azul una recta que no ha conseguido minimizarlo. El color rojo es un buen modelo que hará mejores predicciones que el modelo azul.

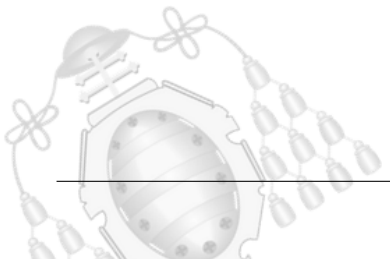
El error cuadrático de cada punto se representa con un cuadrado que incrementa su área al encontrarse más alejado de la recta. Estas áreas añaden mayor penalización a los puntos que se encuentran más alejados. La labor del algoritmo será hallar aquella recta que ofrece una suma mínima de los errores. Ambas rectas son modelos que el algoritmo de aprendizaje automático podría construir. A partir de estas rectas, pueden predecirse futuras notas a según el tiempo de estudio.

Probablemente no sea una novedad para el lector un cálculo de regresión lineal, pero este ejemplo es útil para introducir el concepto de minimización del error, pues es la métrica que se emplea durante los entrenamientos de muchos modelos de aprendizaje automático.

Este ejemplo de dos dimensiones, obtiene como solución una recta. Podría tenerse en cuenta también la capacidad intelectual del alumno, representada como x_2, lo que daría lugar a un plano como solución. También el humor del profesor que lo corregirá x_3, lo motivado que se encuentre el alumno x_4 o el tipo de letra utilizado para los enunciados x_5. Para n variables, la ecuación podría tener el siguiente aspecto:

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + ... w_n x_n \tag{4.5}$$

Esta solución pasa a representar un hiperplano en espacios multidimensionales, que obviamente no puede ser imaginado de forma gráfica. La complejidad de la ecuación, comienza a incrementarse, siendo incluso computacionalmente complicada de resolver de forma analítica. Los modelos que pretenden solucionar un problema real no pueden ser resueltos de forma analítica dada su extensión. Con este ejemplo, se ha introducido de forma simplificada lo que es un modelo. Los cálculos reales que suceden en los entrenamientos son tratados habitualmente como una caja negra donde se conoce la entrada y la salida, sin interpretar los cálculos intermedios. En futuros capítulos se explicará esquemáticamente la metodología utilizada para estos cálculos intermedios.



4.2.2.2.- Algoritmo bayesiano

También conocido como algoritmo de Naive Bayes o bayesiano del ingenuo, es un algoritmo que emplea el método de clasificación basándose en el teorema de Bayes. El atributo de ingenuo se utiliza por el uso de una suposición falsa.

Se trata de uno de los algoritmos más sencillos, ya que se basa en una clasificación a partir de la probabilidad. Una de las hipótesis de partida que realiza el algoritmo es asumir que todas las características influyentes en el resultado son independientes entre sí. Esto generalmente no es cierto, los datos reales sí presentan dependencia entre sí, pero asumir esta independencia permite realizar las operaciones. En el caso de no asumirse las operaciones resultarían imposibles de resolver. Inesperadamente, el algoritmo consigue realizar buenas predicciones aún partiendo de esa hipótesis incierta [28].

El teorema de Bayes adaptado a clases y datos enuncia:

$$P(C|D) = \frac{P(D|C)P(C)}{P(D)} \tag{4.6}$$

Donde:

- C es la hipótesis de una clase dado un conjunto de ellas.
- P(C) es la denominada probabilidad a priori, probabilidad de que la hipótesis C sea cierta.
- P(D) es la probabilidad de D con independencia de la hipótesis.
- $P(C|D)$ es la probabilidad de la hipótesis C dado los datos D.
- $P(D|C)$ es la probabilidad de los datos D dada como cierta la hipótesis C.

Por ejemplo, puede diseñarse un filtro anti SPAM basado en el algoritmo de bayesiano. Primeramente se identifican dos categorías: normal (N) o SPAM (S). De todo el conjunto de datos, el algoritmo realiza un histograma con la frecuencia en la que aparece cada palabra en cada categoría. De esta forma, se obtiene:

$P(N) = \frac{15}{20} = 0,75$
$P(\text{"}querido\text{"}|N) = \frac{16}{40} = 0,4$
$P(\text{"}amigo\text{"}|N) = \frac{12}{40} = 0,3$
$P(\text{"}otras\text{"}|N) = 0,3$

En el conjunto de datos, se encuentran 15 correos de 20, que están etiquetados como normales. En el histograma realizado de todas las palabras, se conoce la frecuencia con la que aparece la palabra "querido" y "amigo". Así mismo, existen otras palabras que no interesan para este ejemplo. Si realizamos lo mismo para los correos SPAM:

$P(S) = \frac{5}{20} = 0,25$
$P(\text{"}querido\text{"}|S) = \frac{3}{30} = 0,1$
$P(\text{"}amigo\text{"}|S) = \frac{15}{30} = 0,2$
$P(\text{"}otras\ palabras\text{"}|S) = 0,7$

En el conjunto de datos, se encuentran 5 correos de 20, que están etiquetados como SPAM. Se conoce la frecuencia de las palabras a analizar: "amigo" y "querido". Al igual que antes, el resto de palabras no son de interés.

Ahora se recibe la combinación de palabras "Querido amigo", el algoritmo procede a calcular la probabilidad que asigna a cada etiqueta:

- $P(Normal) * P(\text{"}querido\text{"}|N) * P(\text{"}amigo\text{"}|N) = 0{,}09$
  este valor será proporcional a $P(N|\text{"}querido\ amigo\text{"})$ y corresponde a la categoría NORMAL
- $P(SPAM) * P(\text{"}querido\text{"}|S) * P(\text{"}amigo\text{"}|S) = 0{,}03$
  este valor será proporcional a $P(S|\text{"}querido\ amigo\text{"})$ y corresponde a la categoría SPAM

Dada que la probabilidad de la categoría NORMAL es mayor, el algoritmo predecirá que este correo no es SPAM. Cabe destacar que en este tipo de algoritmo bayesiano nunca se incluye probabilidad 0 para una característica. Si se introdujera un 0, la probabilidad resultante sería siempre 0.

4.2.2.3.- Algoritmo de árbol de decisión

Los algoritmos de árbol de decisión utilizan aprendizaje automático supervisado. Su principal enfoque es para la resolución de problemas donde interviene la clasificación y la regresión. Este algoritmo también se denomina CART (Classification and regression tree).

El sistema de funcionamiento de este algoritmo puede compararse con un diagrama de flujo. De esta forma, cada nivel del diagrama puede plantearse como una pregunta, que será respondida con si o no. Tras atravesar varios niveles se obtendrá una respuesta final.

El árbol estará formado por nodos de pruebas que representan un atributo o característica. Las ramas que unen los nodos son reglas de decisión y los nodos de decisión o finales representan un resultado. La variable más representativa se denomina nodo raíz y realiza la primera división de los datos.

Para afrontar un problema con este algoritmo, debe realizarse una selección de atributos. La búsqueda de características que mejor dividan los datos aumenta la eficiencia. Estas características o atributos se convertirán en nodos de decisión que irán dividiendo los datos en subconjuntos más pequeños. Tras iterar todos los datos, se irán añadiendo nuevos atributos, finalizando el proceso cuando todos los datos puedan clasificarse con el árbol actual. Alcanzada esa situación no existen nuevos casos a contemplar.

Este proceso de selección de los atributos es una heurística, que busca dividir los datos de la forma más eficiente posible.

4.2.2.4.- Algoritmo de clasificación K-Means

Los algoritmos de aprendizaje automático de clasificación emplean métodos para agrupar los datos según características comunes. Estos grupos reciben el nombre de cluster y han sido introducidos en la sección 4.2.1.2. Existen diferentes algoritmos con un funcionamiento similar, pero para esta sección se ha seleccionado el K-Means. Se trata de un algoritmo de aprendizaje automático no supervisado que plantea agrupar los puntos en grupos.

Para ejecutar el algoritmo, debe indicarse el número de grupos que se desean identificar dentro del conjunto de datos. Estos grupos representan la solución ya que ubicar un nuevo dato en un grupo es, en esencia, establecer la etiqueta que le corresponde.

En cada grupo tomará gran importancia un elemento denominado punto central o centroide. Estos puntos centrales, actúan como referencia representativa de un grupo. Un elemento cerca del centroide, estará dentro de su campo lo que permite considerarlo parte del grupo.

La distancia entre un centroide y un elemento a clasificar se mide a partir de la distancia cuadrada euclídea que puede expresarse como:

$$d(x,y)_2 = \sum_{j=1}^{m}(x_j - y_j)^2 = ||x-y||_2^2 \tag{4.7}$$

Donde se representa la distancia entre x e y, en un plano de m dimensiones con j como cada dimensión. Para conocer la precisión de las agrupaciones se emplea la suma de los errores cuadrados. Este término se denomina inercia de los clusters y viene dado por la siguiente expresión:

$$SEE = \sum_{i=1}^{n}\sum_{j=1}^{k} w^{(i,j)}||x^i - \mu^j||_2^2 \tag{4.8}$$

“Donde (j) es el centroide del cluster j, y w(i,j) es 1 si la muestra x(i) está en el cluster j y 0 en caso contrario.” Fuente del razonamiento matemático [29].

El proceso de construcción del modelo sigue los siguientes pasos:

1. Al comenzar el proceso, el algoritmo debe conocer el número de clases que se desean encontrar dentro del conjunto de datos. Los centroides de cada grupo son inicializados aleatoriamente.
2. Cada instancia del conjunto de datos se asocia a un centroide. Como el centroide representará el núcleo de un grupo, una instancia pertenecerá a aquel grupo cuyo centroide este más próximo.
3. Se calcula el error cuadrático, es decir la inercia de los clusters, SEE.
4. Los centroides son actualizados, el nuevo centroide se ajusta para ser el verdadero centro del conjunto de datos que representa. Esto se realiza minimizando el error cuadrático de los datos.
5. Se repite el proceso desde el paso 2 con el fin de mejorar la agrupación en cada iteración.

Cabe destacar que este algoritmo no es útil para todos los conjuntos de datos. Además, la elección del número de grupos es determinante en su efectividad. Por lo tanto, trabajar con conjuntos de datos donde se conoce el número de los grupos presentes repercutirá positivamente en el proceso de agrupación.

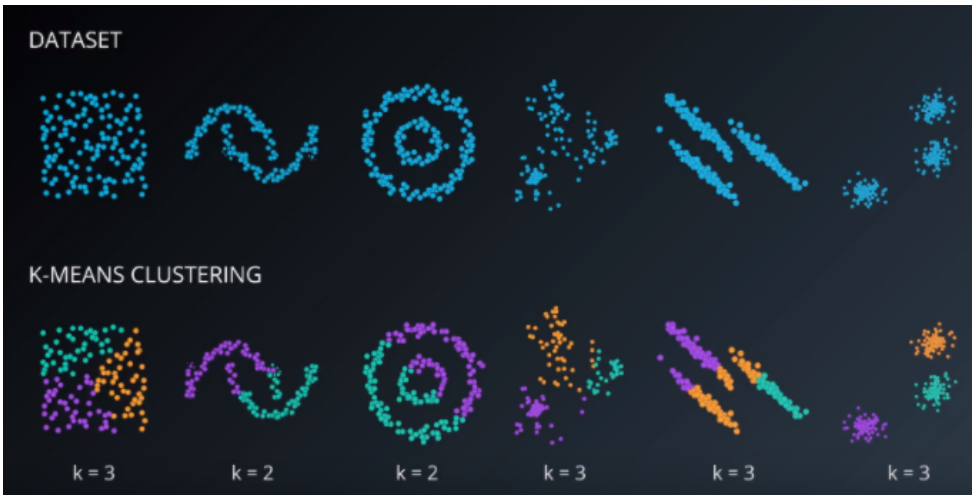


Figura 4.11.- Efectividad del algoritmo k-means según la distribución de los datos [29]

En la figura 4.11, se observa como se realiza la agrupación de los datos en k grupos, según la distribución que presenten. Los datos con tendencia circular presentan una mayor eficiencia, pues el sistema basado en centroides tiene esa naturaleza.

4.2.2.5.- Redes Neuronales

Las redes neuronales son un enfoque de Machine Learning que emplea una estructura inspirada en el cerebro humano. Las señales de entrada son recibidas por neuronas que interpretarán el estímulo recibido y responderán con otros impulsos hacia otras neuronas. La esencia del funcionamiento de la red neuronal artificial, es la misma que las respuestas nerviosas que se producen en el cerebro humano. El proceso de aprendizaje consiste en conseguir que las neuronas adapten los impulsos para que la red completa pueda representar un comportamiento concreto.

Las redes neuronales más simples, que se catalogan dentro del Machine Learning, poseen tres niveles. Las entradas, una capa de neuronas y las salidas.

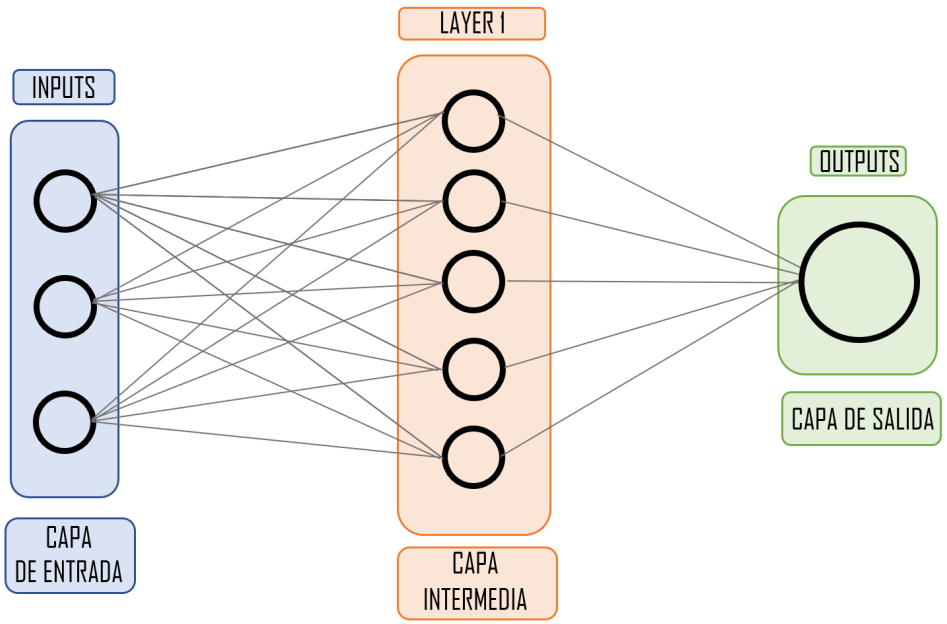


Figura 4.12.- Estructura red neuronal simple

Como se puede apreciar en la figura 4.12, la capa intermedia puede contener varias neuronas para construir el comportamiento deseado. Las capas de entrada y salida solo

sirven como formato de los datos. Para comprender el funcionamiento de estas redes, primero se explicará la unidad básica de procesamiento: la neurona.

Mediante las conexiones de entrada de la neurona, se reciben los valores de entrada con los que realizará un cálculo interno produciendo una salida. El término neurona sirve para facilitar la comprensión del sistema porque, en realidad, solo se trata de una función que genera una salida a partir de las entradas. Cada entrada representa una variable que formará parte de la operación. Dichas entradas tienen un peso asociado que corresponde a la ponderación de cada variable en la expresión. En cuanto a lógica se refiere, los pesos que se asocian a cada entrada corresponden con la importancia que tienen de cara a generar un resultado. En este punto nace el concepto del aprendizaje de la neurona. Tras efectuar entrenamientos la neurona alterará los pesos de cada entrada para conseguir la salida esperada en cada caso [7].

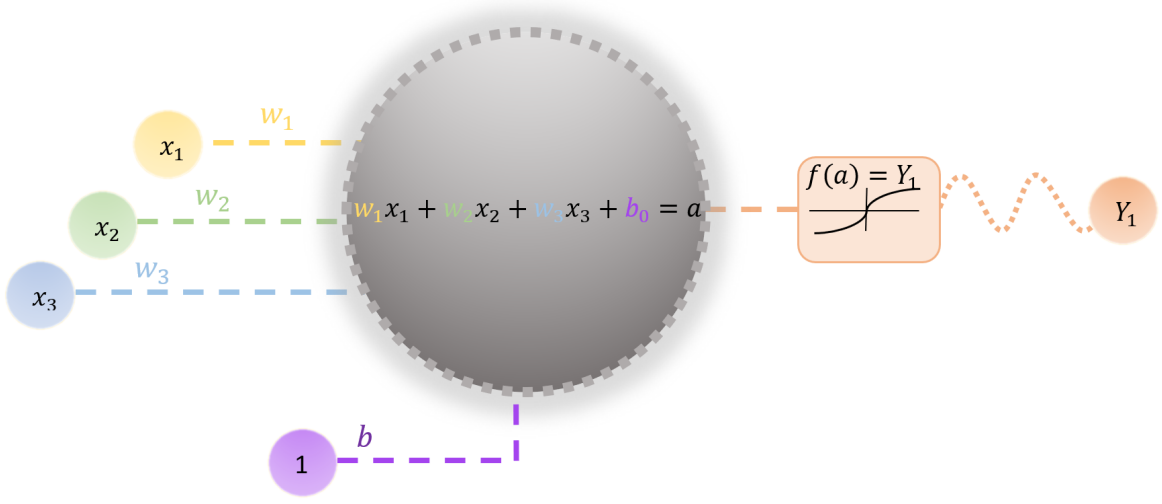


Figura 4.13.- Estructura de una neurona

En la figura 4.13, se ha representado esquemáticamente una neurona. Las entradas se definen mediante $x_1, x_2, x_3 ... x_n$, donde n representa el número de entradas totales.

Cada entrada posee asociado un peso $w_1, w_2, w_3 ... w_n$, que multiplicarán a cada entrada en la operación interna. Además, se encuentra un término independiente b denominado sesgo o "bias". Este término permanece invariante modificándose únicamente con el aprendizaje.

Se asociará el valor 1 a la entrada del sesgo para representar que no depende de una entrada.

La activación a para 3 entradas se corresponde con la expresión:

$$a = w_1 x_1 + w_2 x_2 + w_3 x_3 + b_0 \tag{4.9}$$

De forma general para n entradas puede expresarse como:

$$a = \sum_{i=1}^{n} w_n x_n + b_0 \tag{4.10}$$

La activación a corresponde a una relación lineal que ha de ser transformada mediante una función de activación f. De esta forma el output de la neurona, Y_1 puede ser escrito como:

$$Y_1 = f(a) \tag{4.11}$$

Tras haber especificado la neurona de forma particular, puede desarrollarse las ecuaciones para una red neuronal. Cada neurona j recibe una entrada $p_j(t)$ correspondiente a las neuronas predecesoras. Cada neurona tendrá por tanto una activación $a_j(t)$ que dependerá de un parámetro temporal discreto y un sesgo b_j.

Mediante la función de activación f, se calculará la a activación correspondiente al instante $t+1$ dando lugar a:

$$a_j(t+1) = f(a_j(t), p_j(t), b_{0_j}) \tag{4.12}$$

Donde la función de salida Y_j viene dada por la expresión:

$$Y_j(t) = f_{out}(a_j(t)) \tag{4.13}$$

Por tanto puede escribirse la función de propagación como:

$$p_j(t) = \sum_i Y_j(t) w_{ij} + b_{0_j} \tag{4.14}$$

4.2.2.6.- Funciones de activación

Las funciones de activación son uno de los elementos más importantes de las neuronas. Ya se detalló el funcionamiento de una neurona de forma aislada y su posible combinación con otras mediantes las entradas y la salida.

Con una única neurona puede modelarse un comportamiento muy simple, por ejemplo, activar la salida únicamente cuando las dos entradas también estén activas. Este comportamiento corresponde a una puerta lógica AND. Gráficamente existe una recta frontera que separada los posibles valores de salida. Aquellos que se encuentren por debajo de la recta corresponderán a (0,0), (0,1) o (1,0). Mientras que el valor (1,1), se encontrará en la zona superior a esta. Lo mismo puede realizarse con la puerta lógica OR [30].

El problema apareció cuando se pretendía modelar la puerta XOR con una única neurona. El antecesor a las neuronas, el perceptrón actuaba como una función umbral para ofrecer un resultado como ocurría con las puertas lógicas AND y OR. Sin embargo, resultaba imposible modelar el comportamiento de la puerta XOR, pues no podía ser descrito mediante un único umbral.

Conseguir modelar la puerta XOR, abrió la posibilidad de emplear varias neuronas y utilizarlas en diferentes capas. Gracias a esto, comportamientos mucho más complejos podían ser representados mediantes estas redes .

El único inconveniente era que las neuronas modelaban mediante regresión lineal. Matemáticamente, concatenar muchas operaciones de regresión lineal da como resultado una nueva regresión lineal. Es decir, toda la red neuronal planteada formada por muchas neuronas podría simplificarse a una sola. Surgió en este punto la necesidad de que la salida de las neuronas no fuera una simple regresión lineal. Las salidas deberían sufrir una distorsión para crear comportamientos no lineales, nacen aquí las funciones de activación.

Como se indica en la ecuación 4.11, la combinación lineal a se deforma mediante la función de activación f. Existen principalmente 5 funciones de activación que aportan deformaciones diferentes:

- **Sigmoide**: la función Sigmoide convierte la activación a a un valor comprendido entre (0,1). Los valores de gran magnitud saturarán a 1 cuando sean positivos y a 0 si son negativos. También se conoce como función lógica debido a sus características. Se utiliza habitualmente en la última capa. Gráficamente se representa como la figura 4.14.

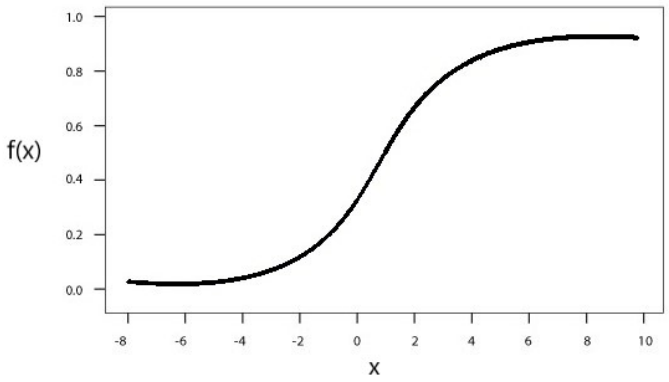


Figura 4.14.- Función de activación Sigmoide. Fuente: Anlytic Steps [31]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.15}$$

- **Tangente Hiperbólica**: la Tangente Hiperbólica distorsiona la activación a al rango (-1,1). Posee un buen rendimiento en redes neuronales recurrentes. Gráficamente se representa como la figura 4.15.

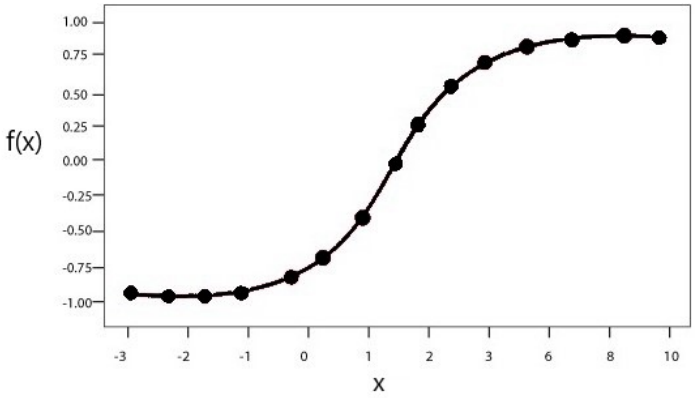


Figura 4.15.- Función de activación Tangente Hiperbólica. Fuente: Anlytic Steps [31]

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{4.16}$$

- **ReLU**: se denomina Unidad Lineal Rectificada, ReLU, ya que transforma en 0 todos los valores negativos. Su rango va de 0 a infinito y ofrece un buen rendimiento en redes neuronales convolucionales. Gráficamente se representa como la figura 4.16.

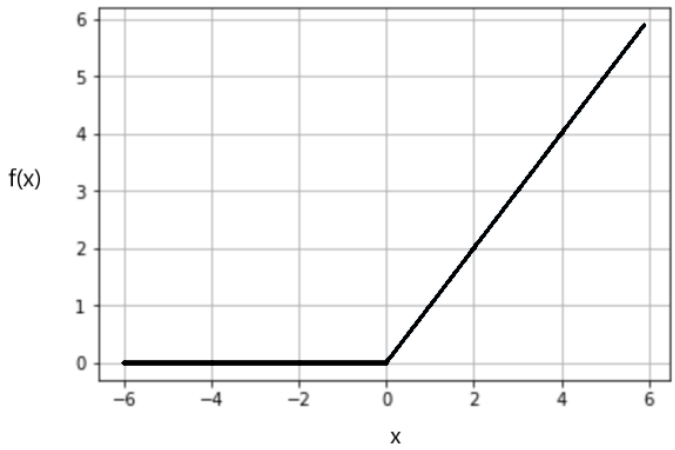


Figura 4.16.- Función de activación ReLU. Fuente: Anlytic Steps [31]

$$f(x) = \begin{cases} 0 & \text{si x} < 0 \\ x & \text{si } x \geq 0 \end{cases} \tag{4.17}$$

- **Leaky ReLU**: es una modificación de la función ReLU que permite los valores negativos pero acentúa los positivos. Si el valor es negativo, su valor de salida será el mismo, sin embargo, si es positivo se multiplica por un coeficiente a. Gráficamente se representa como la figura 4.17.

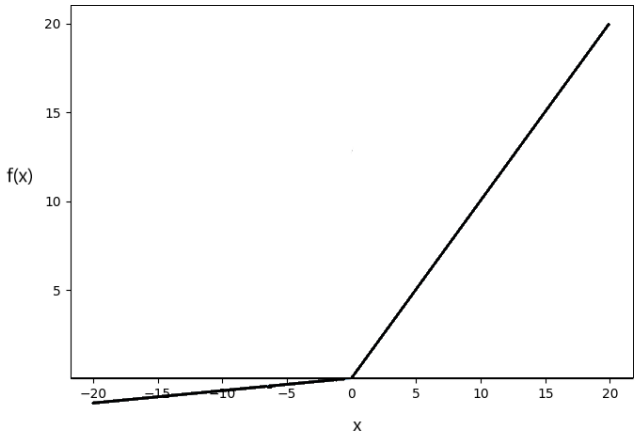


Figura 4.17.- Función de activación Leaky ReLU. Fuente: Anlytic Steps [31]

$$f(x) = \begin{cases} x & \text{si x} \leq 0 \\ ax & \text{si } x > 0 \end{cases} \tag{4.18}$$

- **Softmax**: esta función transforma las salidas como probabilidad. Se emplea habitualmente en la última capa de forma que expresa la probabilidad asociada a cada salida. El sumatorio de todos los valores es 1. Matemáticamente representa una función exponencial normalizada. La entrada de la función es z que representa un vector de k dimensiones. La salida $f(z)$ corresponde a un vector que aporta una probabilidad asociada a cada categoría. Gráficamente se representa como la figura 4.18.

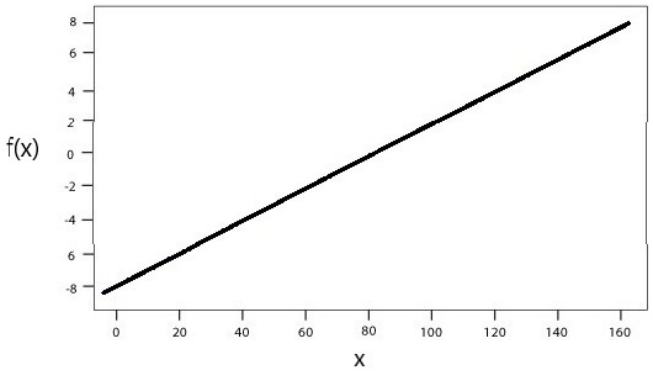


Figura 4.18.- Función de activación Softmax. Fuente: Anlytic Steps [31]

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \tag{4.19}$$

4.3.- Deep Learning

Tras haber explicado la neurona como unidad de procesamiento y su estructura básica, puede introducirse el concepto de Aprendizaje Profundo, Deep Learning. La línea estructural que separa el Machine Learning del Deep Learning son las capas de las redes neuronales. La última estructura de Machine Learning explicada se correspondía con una red neuronal con 3 capas, una capa de entrada, una intermedia y una de salida.

Cuando se comenzaron a añadir más de una capa intermedia de neuronas, el término adoptó esa transformación a Aprendizaje Profundo. La capa intermedia, que antes podía representarse con un capa de neuronas, ahora se denominarán capas ocultas. Estás capas ocultas corresponden a diferentes neuronas de diferentes capas conectadas entre sí ya sea de forma secuencial o alternada.

Se dice que una estructura de red neuronal es secuencial cuando cada capa está conectada únicamente a la capa posterior y anterior. También existen estructuras donde las neuronas de cada capa pueden retroalimentarse y tener conexiones con cualquier nivel de la estructura, pero son menos habituales.

Aumentar el número de neuronas repercute en el número de parámetros de la red, esto ya sucedía en los modelos de Machine Learning con una capa. Sin embargo, aumentar además el número de capas produce un impacto mucho mayor en el número de parámetros. Esto se debe a que cada neurona ha de estar conectada a todas las neuronas de la capa anterior y posterior.

En una estructura simple, el número de neuronas de la capa input y output, era bastante bajo. Para una de estas redes, añadir una neurona en la capa intermedia suponía crear i enlaces de input nuevos, siendo i el número de neuronas de la capa de entrada, y o enlaces de output, siendo o el número de neuronas de la capa de salida.

Añadir una nueva capa supone crear i enlaces nuevos correspondientes al input de la capa oculta anterior y o enlaces nuevos que corresponde al output a la siguiente capa oculta. Como cada capa incluye varias neuronas, estos enlaces se multiplican por el número total de neuronas de la capa. Además, al tratarse de capas ocultas, i y o serán de ordenes mucho mayores en comparación con los casos simples.

En resumen, estas capas adquieren el nombre de ocultas dada la complejidad de sus conexiones y el elevado número de parámetros. Entender la lógica de estas conexiones resulta imposible y se abstrae a operaciones matemáticas para modelar un comportamiento.

4.3.1.- RNA: Perceptrón multicapa

Dentro del Deep Learning, como ya se venía introduciendo, las redes neuronales extienden el número de capas intermedias. El conjunto de capas intermedias entre la capa de entrada y salida se denominan capas ocultas. Se considera más profunda una red neuronal cuantas más capas ocultas posea. Perceptrón es el nombre que le dio Frank Rosenblatt a la estructura de la primera neurona artificial. Es por eso que estas redes profundas también son conocidas como modelos Perceptrón multicapa.

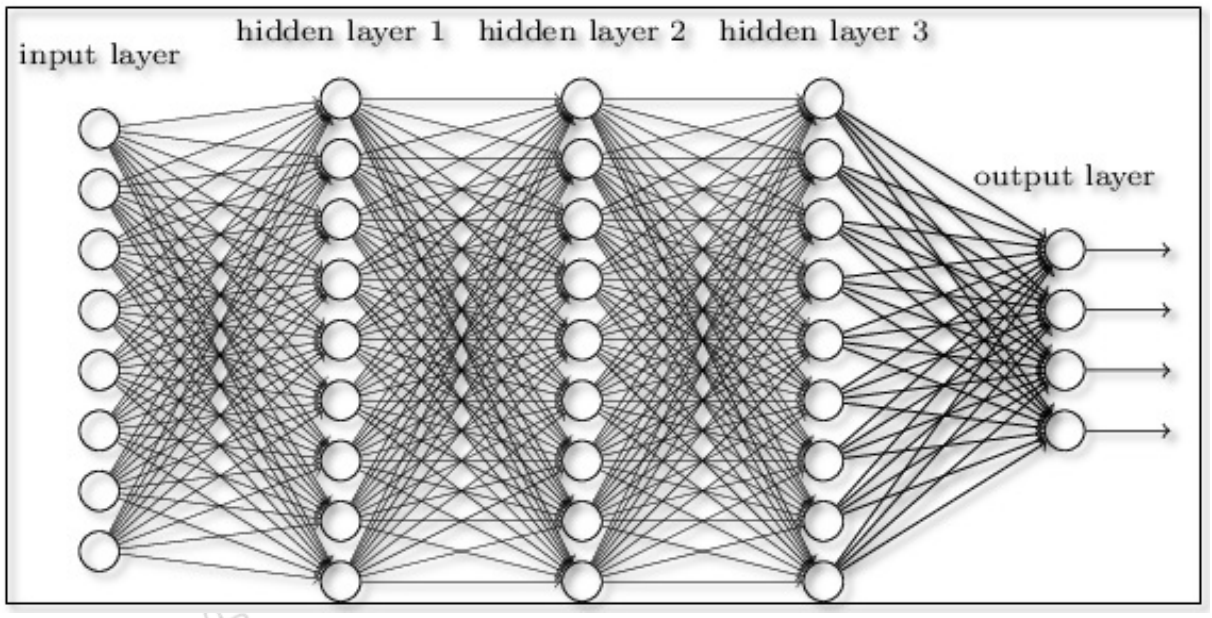


Figura 4.19.- Estructura de una red neuronal profunda. Fuente: Inside BigData [32]

En la figura 4.19, se observa un ejemplo estructural de este tipo de redes.Se pueden visualizar tres capas ocultas con 9 neuronas en cada una. De forma habitual se emplean muchas más neuronas en cada capa, aún así, el número de conexiones que aparenta la fotografía dista mucho del modelo sencillo explicado en la sección 4.2.2.5 anterior. En los casos trabajados para este proyecto, aplicando únicamente redes neuronales, se podrían llegar a alcanzar 200.000 parámetros ajustables diferentes en los entrenamientos.

4.3.1.1.- Aprendizaje

Durante la construcción de un modelo basado en redes neuronales se identifican dos fases, entrenamiento y validación. Inicialmente, todos los parámetros y sesgos de la estructura tendrán un valor aleatorio. Es mediante el proceso de entrenamiento donde esos valores se irán ajustando para conseguir buenos resultados.

En la creación de la estructura existen numerosos hiperparámetros que alteran las formulas empleadas o ciertos mecanismos de actuación. En esta sección se describirá el sistema de aprendizaje de forma general sin incidir demasiado en fórmulas matemáticas demasiado abstractas que podrían cambiar según la estructura.

En el modelo influirá un elemento clave denominado optimizador. Podría equipararse a un responsable de las modificaciones de los valores con el propósito de optimizarlos, de ahí su denominación. La red neuronal nutre su aprendizaje interpretando la base de datos e intentando predecir el resultado de cada entrada.

El entrenamiento se controla mediante una unidad de tiempo denominada época. Una época corresponde a una lectura completa de la base de datos destinada a entrenamiento. Durante la primera época, la red aportará sus primeras soluciones a cada entrada. Como cabe esperar, serán predicciones en su inmensa mayoría incorrectas pues se trata de una resolución basada en el azar. Tras el transcurso de una primera época, el modelo puede evaluarse a sí mismo observando cuál era la solución para cada caso.

Si se trata por ejemplo de identificar emociones a partir de un rostro, el modelo juzgará sus aciertos y errores. Generalmente, para ofrecer un resultado final se emplea la función

de activación Softmax, ya explicada en la sección . Mediante este tipo de función no solo se puede evaluar si un resultado es correcto o no, además se puede cuantificar el grado de confusión. Como los resultados están expresados como probabilidad, cuanto más cercano a 1 esté el porcentaje de la categoría correcta, más eficaz es el modelo. Por el contrario, para los valores incorrectos, el modelo debería haber aportado un 0 % para el resto de categorías.

Estos casos, por supuesto, son en un contexto ideal. Es prácticamente imposible que el modelo muestre un resultado con tanta certeza. Sin embargo, es un método que permite orientar a la red neuronal hacia el aprendizaje.

Entra aquí el concepto del error, es una tendencia habitual prestar demasiada atención a los errores. Para la red neuronal sucede lo mismo, se juzga mediante una función de error. Existen diferentes tipos, pero como patrón común, evalúan cada neurona output asignándole un error. Recolectando los errores de todas las neuronas output, se aporta el error global.

La función de error, al margen de su extensión y complejidad, no deja de ser una función. Por tanto para cada combinación de todos los parámetros de la red, la función error toma un valor. Si reducimos el problema a 2 dimensiones, es sencillo representar la función de error. Podría por ejemplo, representarse como muestra la figura 4.20.

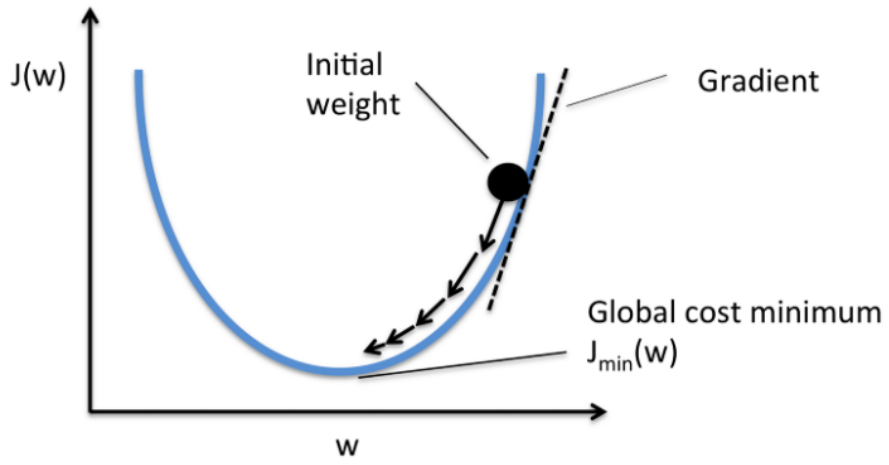


Figura 4.20.- Función de error, ejemplo 2 dimensiones. Fuente: Analyses of DL [33]

Como puede observarse, se trata de una función convexa. Caracterizada por tener un único mínimo global, esta parábola es un buen ejemplo para optimizar. Las dimensiones reales de la función de error están muy alejadas de esta imagen en 2D. Esto implica que es computacionalmente inasumible calcular todos los puntos y representarla. Pongámonos entonces en la cima de una cordillera montañosa, o en uno de los puntos de mayor valor de la figura 4.20.

El punto etiquetado como “Initial weight“ representa el error enorme que está cometiendo la función durante la primera época. Comienza ahora la fase dos, modificar los valores de los parámetros. El optimizador no conoce el punto mínimo de la función, pero si puede calcular el vector gradiente que le indica la dirección en la que crece la función. Si altera los valores para moverse en la dirección opuesta estará reduciendo el error. De esta forma, el modelo va transformándose a lo largo de las épocas minimizando la función de error.

4.3.1.2.- Backpropagation

El término de backpropagation o propagación hacia atrás, se refiere al algoritmo que se emplea para llevar a cabo el aprendizaje. Su funcionamiento se basa en el descenso del vector gradiente correspondiente a la función de pérdidas.

En las capas de salida se genera una señal de error que se propaga hacia atrás alterando los pesos de cada neurona. Es decir, se realiza el recorrido inverso. La modificación que se realiza a cada rama depende de su influencia en el resultado. Es decir, se valora qué rama fue mayor desencadenante del error, para aplicarle una mayor corrección.

Mediante este proceso, a lo largo del entrenamiento, las neuronas de las capas intermedias comenzarán a ajustarse reconociendo patrones influyentes en el resultado final.

4.3.2.- Redes Neuronales Convolucionales

Las CNN, convolutional neural network, son un tipo de red neuronal profunda con principal enfoque en la interpretación de imágenes. La visión computacional se basa en este tipo de estructuras, capaces de trabajar con imágenes. Estas redes utilizan aprendizaje supervisado para mejorar su eficiencia.

Dadas sus características, es la estructura más adecuada para el problema de la detección de emociones. Los datos de entrada se corresponden con caras y la predicción ha de basarse en esas imágenes. Las redes convolucionales son una versión regularizada de el modelo perceptrón multicapa.

Para comprender el funcionamiento de estas estructuras se debe introducir el concepto de las capas convolucionales. Estas capas consisten en aplicar un filtro convolucional a las imágenes con el fin de crear otras nuevas imágenes que representen un nuevo mapa de características. El entrenamiento es muy eficaz ya que adapta los filtros volviéndolos más efectivos. Otros algoritmos tradicionales de clasificación de imágenes requieren de mucho más tiempo para programar sus filtros. En cuanto a procesamiento de imagen se refiere, las CNN han abierto un camino de posibilidades a la visión computacional. Mediante estas estructuras, los filtros se ajustan automáticamente a las necesidades de los datos.

Las capas convolucionales requieren de una función de activación. La causa de ello se encuentra en la linealidad de la operación. Ejecutar sucesivos filtros convolucionales supondría una única operación lineal y ya se describieron anteriormente las limitaciones de esto [3]. Las neuronas correspondientes a la capa input recogerán la información de cada píxel. En este punto, puede surgir la cuestión de por qué no puede afrontarse la interpretación de imágenes mediante redes neuronales simples. Lo cierto es que sí podrían, las neuronas podrían leer la información de cada píxel e intentar encontrar relaciones entre ellos para ofrecer resultados. El problema estaría al probarlo, donde observaríamos la poca eficiencia de esa estructura.

Para la visión humana es complicado de asimilar, pero más allá del valor de cada píxel, existe una interpretación de la profundidad que no se puede obtener analizando de forma

separada cada píxel. Además de otras alteraciones, una red neuronal simple no podría nunca interpretar eficientemente imágenes.

Las capas convolucionales se encuentran dentro del conjunto de capas ocultas y emplean comúnmente la función de activación ReLU. Dada la estructura secuencial, el orden continúa siendo la capa de entrada, las capas ocultas y posteriormente una posible red neuronal simple que desemboca en la capa de salida. El formato de entrada de una CNN corresponde con un vector de 4 dimensiones tal que: (n, h, w, c) donde "n" representa el índice de la imagen, "h" y "w" las dimensiones y "c" el canal de color. La salida resultante tras aplicar un filtro de convolución se denomina mapa de características y podrá ser la entrada de un nuevo filtro. En la figura 4.21, se observa un ejemplo del flujo de funcionamiento de una red convolucional.

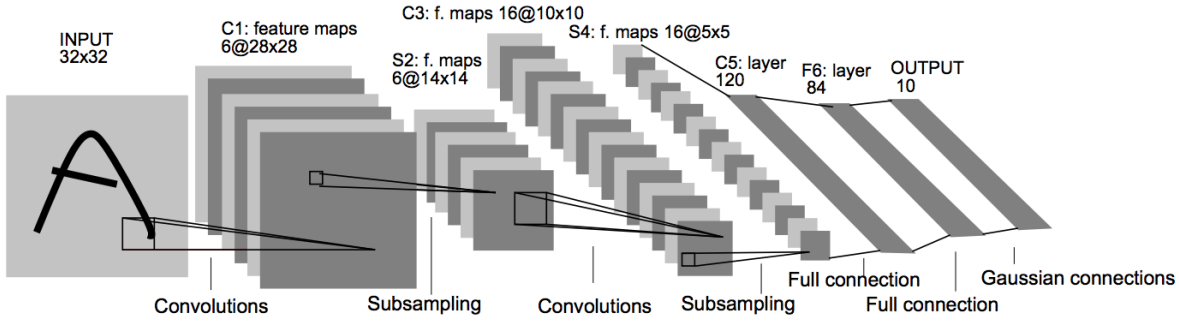


Figura 4.21.- Ejemplo de una estructura CNN. Fuente: LeNet-5 Architecture [34]

Dentro de las estructuras habituales que conforman las redes convolucionales se distinguen los siguientes elementos:

- **Capas convolucionales**: capas donde se aplican filtros convolucionales, también denominados kernel. Aplicar un filtro a toda una imagen consiste en desplazar la matriz por todos los píxeles generando una imagen nueva denominada mapa de características. Los píxeles representan la información de intensidad de color mediante un número. Por tanto, el proceso de aplicar un filtro consiste en multiplicar cada píxel por el valor correspondiente en esa posición de la matriz. Tras multiplicar cada píxel por su coeficiente se suman y da como resultado el valor de un nuevo píxel.

Tras generar un nuevo píxel, el filtro se desplaza. El mapa de características estará formado por los píxeles resultantes del desplazamiento del filtro por toda la imagen. Las dimensiones de los filtros son definidos previamente como un hiperparámetro.

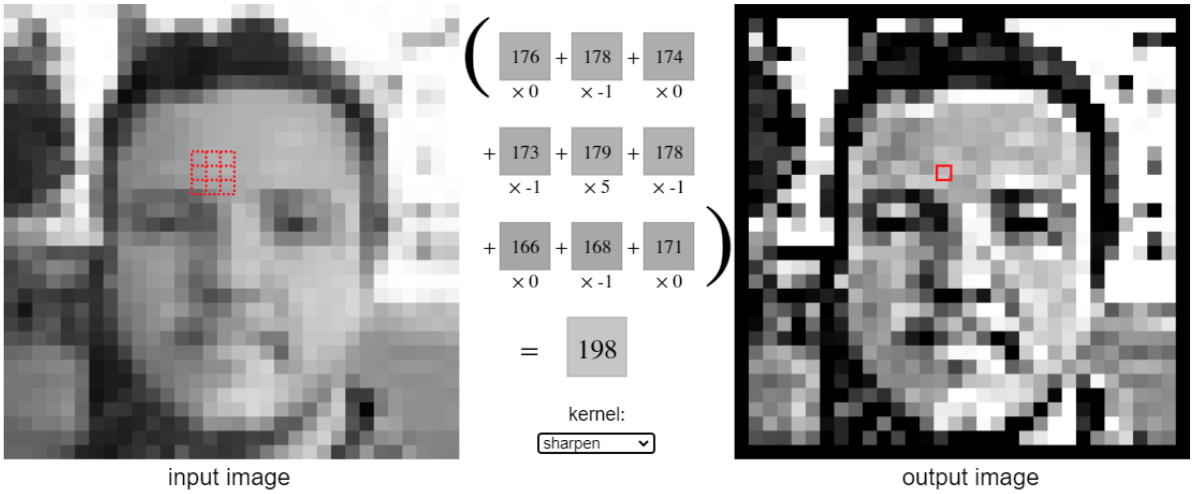


Figura 4.22.- Efecto de aplicar un filtro a una imagen. Fuente: Simulador Kernel [35]

En la figura 4.22, se emplea un simulador para observar el resultado de aplicar un filtro preestablecido denominado "sharpen". Este filtro consigue agudizar las formas de la figura. Durante el aprendizaje, los filtros se regularán de forma automática para obtener información característica. Aunque este filtro parece muy útil, podría no serlo para el modelo. Es por eso que el aprendizaje de las capas convolucionales reside en adaptar los kernels.

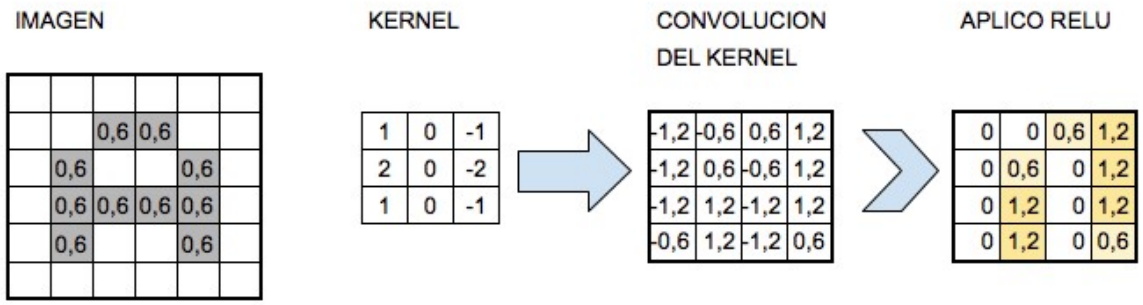


Figura 4.23.- Efecto de aplicar filtro convolucional. Fuente: Aprende Machine Learning [36]

En la figura 4.23, se observa otro ejemplo donde resulta interesante ver como después de aplicar el filtro, se pasa el resultado por la función de activación.

- **Pooling**: se trata de una transformación o agrupación de la imagen, de ahí su denominación. El efecto que se produce es la reducción de la escala, es decir, reducir la cantidad de información. La reducción puede efectuarse siguiendo diferentes métodos de selección, pero el más habitual es el MaxPooling. Esta agrupación consiste en simplificar utilizando el valor del píxel más alto de cada grupo. Esta técnica se aplica principalmente para reducir la carga de trabajo de la red y evitar además el sobre ajuste. Se ha comprobado que aplicar MaxPooling después de las capas de convolución acelera los entrenamientos y produce buenos resultados de predicción. La otra técnica es AveragePooling, que es similar solo que hace una media aritmética de los valores en vez de seleccionar el mayor.

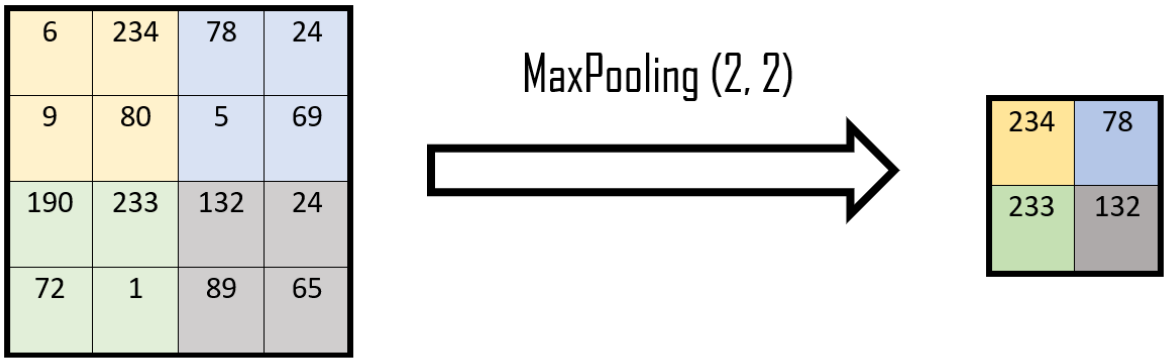


Figura 4.24.- Efecto de aplicar MaxPooling

En la figura 4.24, se observa el proceso de reducción de un fragmento de imagen. De un conjunto de 8x8 píxeles se obtiene uno de 2x2

- **Dropout**: la función Dropout permite desactivar en cada iteración del entrenamiento las neuronas con una probabilidad p. De esta forma, se evita el sobre ajuste y que los filtros particularicen el conjunto de datos en lugar de generalizarlo.
- **Flatten**: cuando la imagen haya pasado por todas las capas de convolución y MaxPooling, la información de salida continuará manteniendo un formato matricial. Para facilitar el tratamiento de estos datos, la función "Flatten()" aplana la información en un vector más extenso.

Tras pasar las capas convolucionales la información pasa a una red neuronal multicapa. De esta forma, ahora las neuronas podrán interpretar un mapa de características más fructífero. El nuevo formato de la información les permite identificar las relaciones y efectuar un entrenamiento más eficiente. Un posible esquema para este proyecto en particular sería el de la figura 4.25.

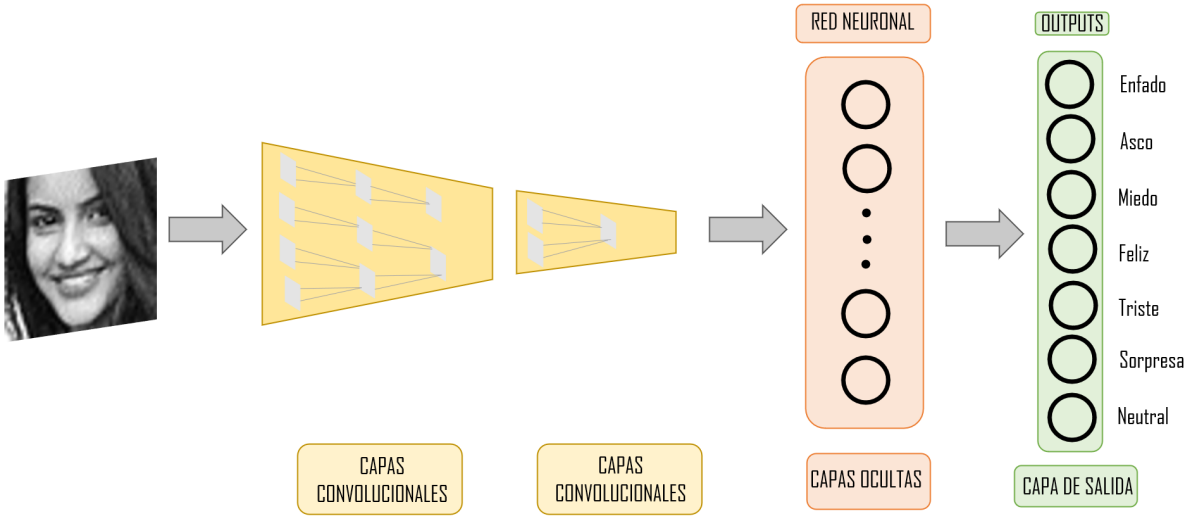


Figura 4.25.- Estructura del modelo completo esquematizado por capas

Para obtener otra visión del funcionamiento de una red convolucional es interesante el ejemplo de las siguientes figuras 4.26 y 4.27. Mediante un simulador 3D fue elaborada una posible visión del funcionamiento de una red neuronal convolucional en el espacio.

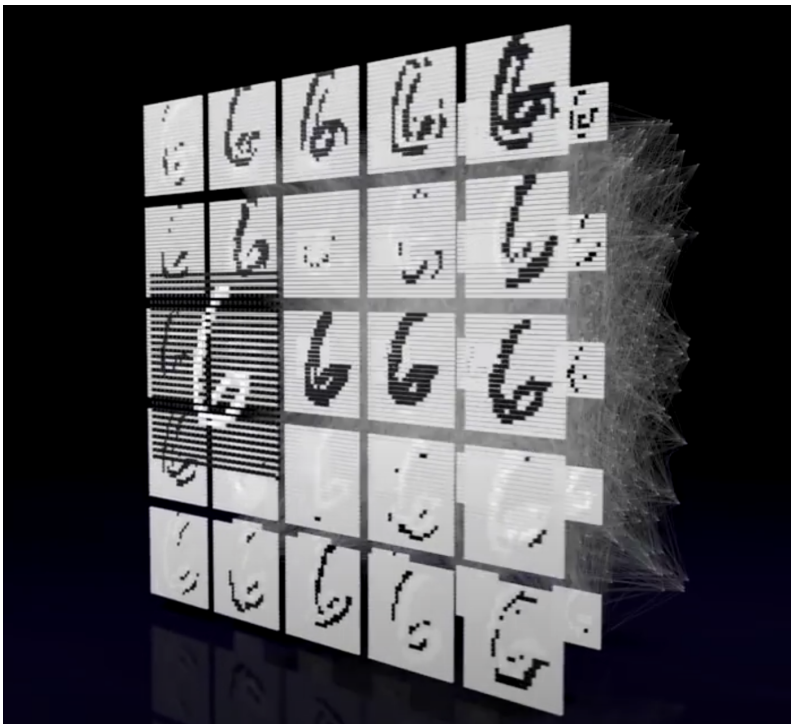


Figura 4.26.- Simulador de una CNN vista ángulo input. Fuente: CyberControls [37]

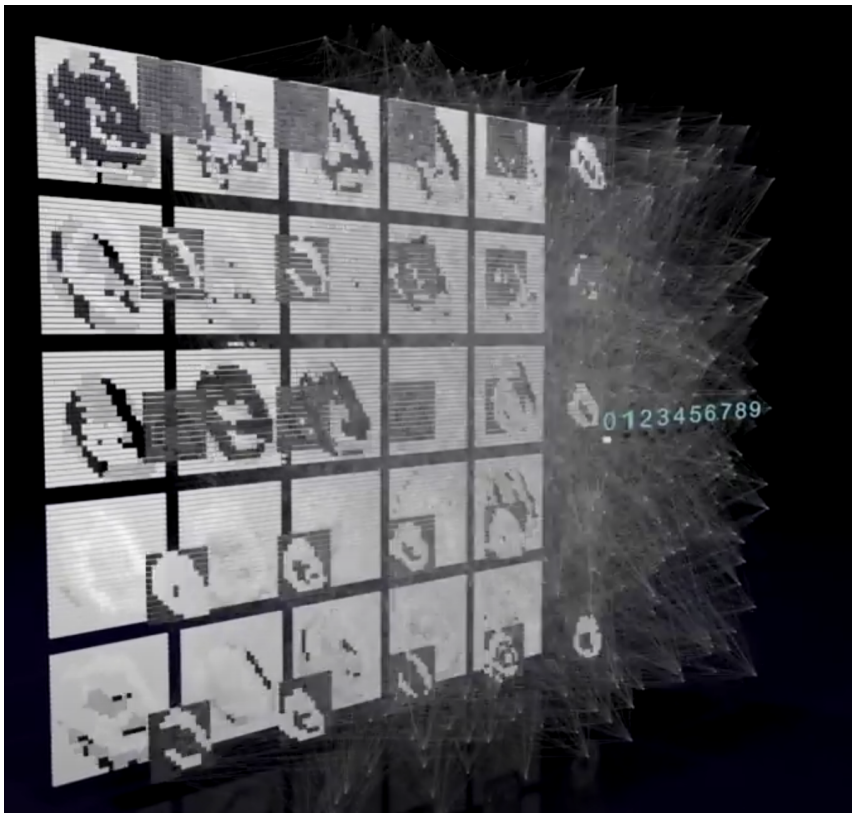


Figura 4.27.- Simulador de una CNN vista ángulo output. Fuente: CyberControls [37]

5. Metodología de trabajo

Para comenzar a crear modelos de predicción, independientemente de su estructura, es necesario un conjunto de datos para poder ejecutar el proceso de aprendizaje. Los datos son un elemento fundamental para el éxito de cualquier inteligencia artificial. La potencia de las redes neuronales, se debe a la cantidad y calidad de los datos con los que fue entrenada. Tal es así que Google, una de las compañías más grandes del mundo, establece su API TensorFlow como código abierto. Facilitan las herramientas para que cualquier usuario pueda construir sus propias estructuras. Si una de las empresas líder en inteligencia artificial como es Google, hace públicas sus herramientas de programación, es evidente que no representa una debilidad que puedan aprovechar sus competidores.

Las estructuras tienen importancia, pero son los datos los que otorgan la potencia. Relacionado con esto, se puede encontrar el vínculo entre la inteligencia artificial y el Big Data. Se denomina Big Data a los conjuntos de información o combinaciones de estos, cuyas dimensiones, complejidad y velocidad de crecimiento dificultan su gestión o procesamiento mediante métodos convencionales como bases de datos. Para proyectos de aprendizaje, se emplean bases de datos pequeñas y sencillas de manipular ya que son suficientes para aprender sobre estructuras de redes neuronales y su aprendizaje. [38]

5.1.- Dataset

Para el entrenamiento de la red neuronal se emplea una base de datos elaborada por Aaron Courville y Pierre-Luc Carrier para proyectos de investigación, Face Emotion Recognition. El dataset [8] contiene 35.888 entradas, representando cada una un rostro mediante una imagen. Las fotografías muestran únicamente rostros, es decir, no existen más elementos a considerar en la imagen. Las fotografías tienen una resolución muy baja, 48x48 píxeles. Además todas se encuentran en blanco y negro lo que compone un dataset muy poco pesado. La información que aporta el conjunto de datos es de tan sólo 294 MB, gracias a ello, podrá ser sencillo de gestionar y facilitará el trabajo de investigación. Dado el poco

peso del dataset, es posible utilizar un archivo de extensión csv, que contendrá toda la información de las imágenes organizada. Un archivo CSV, es un archivo de texto donde los campos están separados por diferentes caracteres para ser interpretados como una tabla [39].

Estos datos aportan la información necesaria para realizar un trabajo de introducción a la inteligencia artificial y procesado de imágenes. Las predicciones que se obtendrán, así como su eficacia, serán producto de un aprendizaje de 48x48 píxeles. Puede el lector plantearse la potencia que podría llegar a tener una inteligencia artificial entrenada con mucha más información en cada entrada. El aumento de la cantidad de información produce un gran impacto en los tiempos de procesado así como en los requisitos de los equipos donde se ejecutan. Al mismo tiempo logrará modelos mucho más complejos, capaces de producir resultados muy precisos. Encontrar el equilibrio entre coste computacional, tiempo y eficacia de los entrenamientos, es un campo de estudio para los ingenieros de datos.

Como ya se ha mencionado, para este trabajo, el formato de imágenes poco pesadas sirve para abordar el correcto enfoque de los objetivos.

5.1.1.- Formato de los datos

En la figura 5.1, puede visualizarse el formato de la tabla que representa el archivo csv.

	A	B	C
1	emotion	pixels	Usage
2	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110	Training
3	0	151 150 147 155 148 133 111 140 170 174 182 154 153 164 1	Training
4	2	231 212 156 164 174 138 161 173 182 200 106 38 39 74 138	Training
5	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 19 43 52 13 26	Training
6	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84 115 127 137 142	Training
7	2	55 55 55 55 55 54 60 68 54 85 151 163 170 179 181 185 188	Training
8	4	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 108 118 130 13	Training

Figura 5.1.- Fragmento del Dataset interpretado por Excel

Cada fila de la tabla representa una entrada del dataset. Para poder realizar un entrenamiento supervisado de la red neuronal, todas las entradas utilizadas deben estar etiquetadas con la solución esperada. Esto ya fue detallado en la sección 4.2.1.1.

La columna central representa una imagen que contiene un rostro. Al tratarse de una imagen en blanco y negro, solo se requiere un número comprendido entre 0 y 255 para cada píxel. Resulta más simple que el formato más habitual RGB, que requiere de tres números para representar el color de un píxel. Dado que cada imagen tiene una resolución de 48x48, cada fotografía tiene únicamente 2304 píxeles. La dimensión es ínfima comparada con una imagen en full hd o 4k, que contienen 2.073.600 y 8.294.400 píxeles, con formatos de color que incluyen más información a cada píxel. Por tanto, 2304 números separados por un espacio definen cada imagen.

La primera columna representa la etiqueta solución que acompaña a cada rostro. Las soluciones posibles son 7 emociones catalogadas mediante un código numérico:

- 0: Enfadado
- 1: Asqueado
- 2: Asustado
- 3: Feliz
- 4: Triste
- 5: Sorprendido
- 6: Neutral

La tercera columna del dataset, representa información acerca del uso esperado de cada imagen, ya sea entrenamiento o pruebas.

Para probar cualquier modelo de inferencia de forma precisa, es necesario dividir el conjunto de datos en dos categorías, pruebas y entrenamiento. Durante el aprendizaje profundo, los pesos de las neuronas se irán adaptando para obtener la solución esperada. Iterarán repetidas veces por el conjunto de datos catalogado como entrenamiento, es decir, todo el conocimiento adquirido solo provendrá de está categoría de imágenes.

Por otra parte, la categoría de pruebas será utilizada repetidamente para proporcionar información de la eficiencia del modelo ante imágenes que nunca antes ha interpretado. Gracias a dividir el conjunto de datos, el modelo podrá mejorar utilizando las imágenes de entrenamiento reduciendo el error que produce ante estas. Al mismo tiempo, se testea con la categoría de pruebas para mostrar una eficiencia más realista, ya que la iteración sobre esta categoría no participa en el aprendizaje. Conocer la precisión ante imágenes que el modelo nunca ha interpretado, permite valorar si el modelo proporciona soluciones genéricas o particularizadas al conjunto de datos [40].

Los modelos de inferencia resultantes, pueden verse como un alumno que debe aprender conocimiento para enfrentarse a unos problemas. Existen alumnos que centran el aprendizaje en repetir los métodos de los ejercicios que ya han visto resueltos. Dado que su aprendizaje es demasiado específico, fracasan ante problemas nuevos, sin embargo, pueden llegar a mostrar un falso éxito si se repite un problema ya visto.

Por otro lado, el alumno que adquiere un conocimiento general, sin caer en lo particular, probablemente logrará el éxito ante nuevos problemas.

5.1.2.- Características de las imágenes

Como ya se mencionó en la sección anterior, las imágenes tienen una resolución de 48x48 píxeles. Tal y como se aprecia en la figura 5.2, la fotografía no es de la calidad que habitualmente se utiliza. A pesar de ello, cuando la imagen aporta características claras como en este ejemplo de un rostro feliz, la red neuronal se nutrirá de ello.

Figura 5.2.- Ejemplo rostro 48x48 píxeles del dataset

Para conseguir construir el modelo de inferencia, todas las imágenes de entrada tienen la misma resolución. Por tanto, cuando el modelo realice predicciones, solo utilizará información correspondiente a 48x48 píxeles. Para las pruebas, se podrán utilizar todo tipo de imágenes así como las que toma la cámara a tiempo real. Sin embargo, antes de pasar el rostro al modelo, la imagen se verá transformada a la resolución reducida.

De esta forma, es esperable que el modelo presente limitaciones ante cambios sutiles o rostros caracterizados por una emoción poco representativa. Al reducir la dimensión de la imagen, esa información se perderá y el gesto no será perceptible ni siquiera por alguien que visualice la fotografía.

5.1.3.- Distribución de las imágenes

Previamente a efectuar los entrenamientos, se realiza un estudio acerca de la distribución del dataset. La cantidad de imágenes que se dispongan de cada categoría provoca un impacto directo en el modelo. La excesiva presencia de una categoría puede provocar un

desajuste en la imparcialidad del modelo. En otras palabras, la red neuronal adquiere la tendencia hacia ese resultado siempre por las mismas vías. Para evitar esta tendencia a particularizar las soluciones, existen herramientas muy útiles que agregan factores de aleatoriedad y modificaciones artificiales a los datos. Estas herramientas se detallarán más adelante.

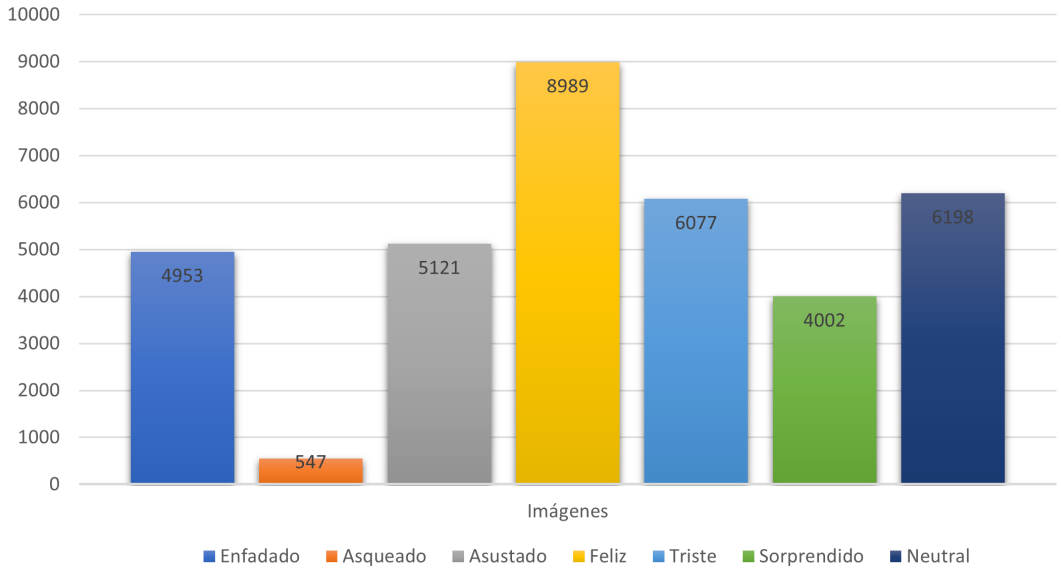


Figura 5.3.- Distribución de las imágenes para cada emoción

Previamente, también se realiza un estudio de la distribución de los datos en el dataset. En la figura 5.3, se observa la cantidad de imágenes disponibles para cada emoción. Claramente existe una desproporción para la categoría de "asqueado", así como un exceso en la categoría de "feliz". Esto ha de tenerse en cuenta en la etapa de pruebas para proporcionar soluciones y mejoras.

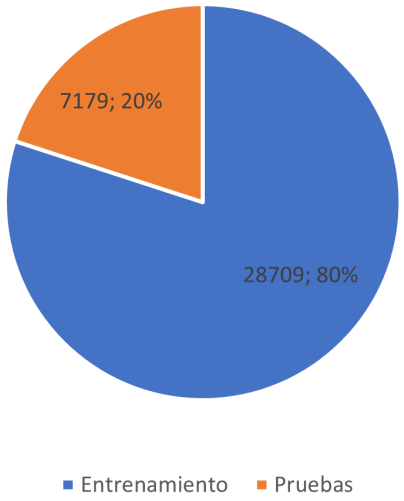


Figura 5.4.- Distribución de las imágenes según su función

En la figura 5.4, se muestra la cantidad de imágenes que se destinarán a entrenamiento y a pruebas según la recomendación del dataset.

5.1.4.- Eficiencia en las etiquetas

Se definen las emociones como un conjunto de reacciones fisiológicas que experimenta un individuo respondiendo a estímulos externos [41]. Clasificar emociones resulta en algunas ocasiones una tarea abstracta, siendo difícil seleccionar una única etiqueta. Se muestran en las siguientes figuras, ejemplos extraídos de la base de datos para cada emoción.

Figura 5.5.- Imágenes etiquetadas como enfado

Figura 5.6.- Imágenes etiquetadas como asco

Figura 5.7.- Imágenes etiquetadas como miedo

Figura 5.8.- Imágenes etiquetadas como felicidad

Figura 5.9.- Imágenes etiquetadas como neutral

Figura 5.10.- Imágenes etiquetadas como tristeza

Figura 5.11.- Imágenes etiquetadas como sorpresa

En términos generales, las agrupaciones muestran información muy característica que pueden nutrir un entrenamiento fructífero. No obstante, dentro de las categorías se han encontrado imágenes poco representativas, llegando a resultar ambiguas. Se han colocado en primera posición de cada figura, un ejemplo de este tipo de imágenes dudosas. Estas imágenes ambiguas o mal etiquetadas no se presentan como un problema a priori, de todos modos, ha de tenerse en cuenta en etapas de análisis y mejora.

5.2.- Del dataset al Machine Learning

5.2.1.- NumPy y Pandas

Las librerías enfocadas al Machine Learning se desarrollarán en la sección 5.3.1. Estas cubren el funcionamiento estructural de las redes neuronales así como su entrenamiento. La conexión del dataset con esos algoritmos de aprendizaje es la capa donde actúa la librería NumPy y será descrita en este capítulo. Se recortarán fragmentos de código que aporten información y cambios del formato de datos. El resto de funcionalidades se obviarán ya que son irrelevantes para esta sección.

NumPy permite operar en Python con vectores y matrices de gran extensión. En el proyecto, permitirá trabajar con el dataset, almacenando su información en vectores y matrices multidimensionales [42].

Además aportará el formato de datos necesario para trabajar con la librería OpenCv, que se emplea en la puesta en práctica del modelo.

Como ya se ha observado en el análisis del dataset, una imagen es una conjunto de números comprendidos entre 0 y 255, donde cada uno aporta la información de un píxel. Dada la naturaleza de una imagen, el conjunto de números se organiza en realidad como

una matriz. De esta forma, la resolución utilizada (48x48) representa una matriz cuadrada de 48 filas y 48 columnas con 2304 elementos (píxeles).

Una fotografía con formato RGB posee tres canales de colores. La imagen final es la unión de la tres imágenes en rojo, azul y verde. La base de datos, como ya fue mencionado, posee las imágenes en blanco y negro, por lo tanto solo se utiliza un canal de color. Aún así, se describe la estructura vectorial de imágenes de tres canales ya que para aplicar algunas versiones de redes neuronales, estos serán necesarios.

Como el conflicto de compatibilidad solo es cuestión del formato, bastará con triplicar la información de la primera capa a las otras 2. De esta forma, los tres canales de color contendrán las misma información.

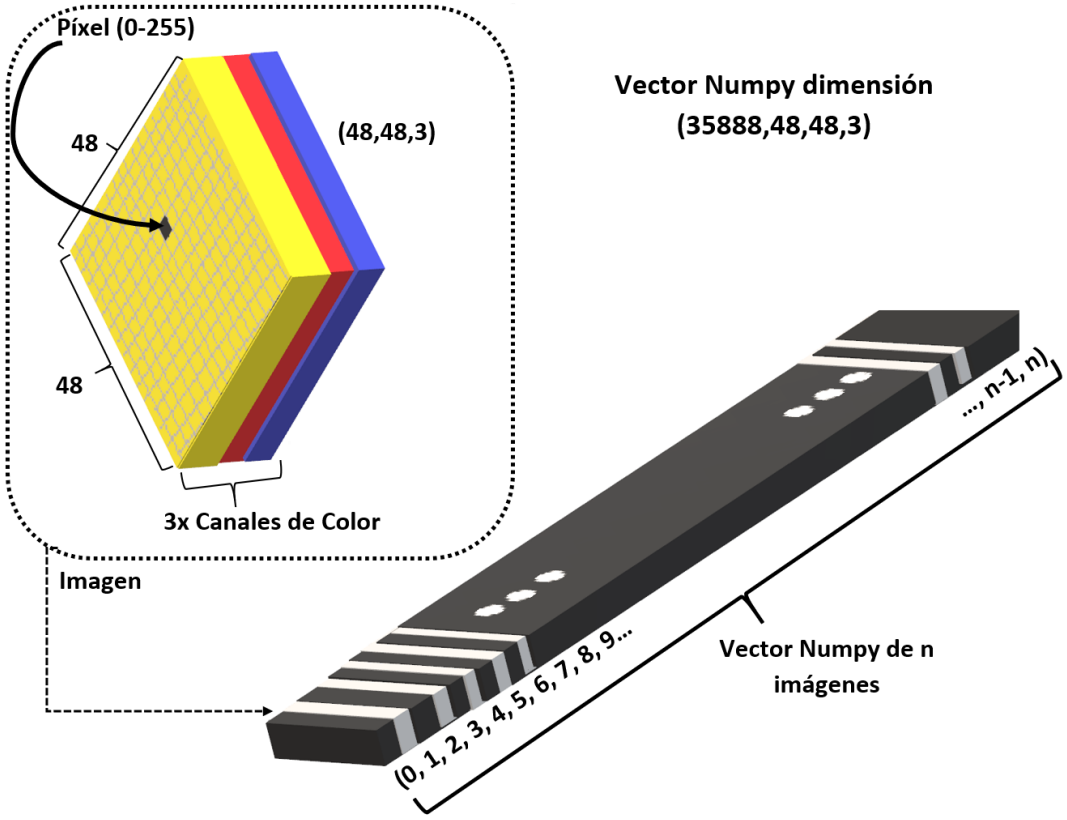


Figura 5.12.- Representación multidimensional del dataset

La figura 5.12, muestra una construcción de cómo podría visualizarse el vector NumPy con el que trabaja la red neuronal. Matemáticamente, el vector posee 4 dimensiones, (a,b,c,d).

El índice “a” representa el número de la imagen y, los índices “b” y “c” fijan el píxel de la capa que indica “d”. Una forma más sencilla de verlo es plantear por ejemplo, que una imagen es la unión de 3 matrices (48x48) y se colocará en la posición 1 del vector. Acorde a esta estructura, para acceder a la imagen completa, se deberá recorrer el vector desde la posición (1,0,0,0) a la (1,48,48,3).

Por otra parte, la librería Pandas es una extensión de NumPy que permite la lectura y escritura de datos así como su segmentación y reestructuración. Para este proyecto, se utiliza para leer el archivo en formato csv y proyectar el contenido a variables NumPy.

Dada la dimensión del dataset, es viable incorporarlo al completo en memoria para ejecutar el entrenamiento. Para un dataset mayor será necesario utilizar lotes de datos con un tamaño más reducido, de esta forma, se evita una sobrecarga de la memoria.

5.2.2.- Incorporación de los datos

La incorporación del dataset al programa es la primera tarea que se ejecuta en el main(). Como se aprecia en la figura 5.15, la función tiene un doble retorno, es decir, devuelve dos variables.

```
def main():
    #Obtenemos dos numpy array con las emociones y las imagenes en orden
    emotion_array_np, images_array_np  = load_data(CSV_FILE)
```

Figura 5.13.- Fragmento de código: llamada a la función load$_data$

El proceso de transformación de los datos del csv a los arrays es algo más compleja de lo que a simple vista parece. La función load_data, abstrae los procesos que se visualizan en la figura 5.15.

```python
# FUNCIONES
def load_data(csv_dir):
    # CARGAR DATOS DEL CSV
    # Todos los datos
    print('Comienza la importación de los datos')
    data_csv = pd.read_csv(csv_dir, header=0)

    # Se describen datos de interés
    print('La base de datos contiene',format(data_csv.shape[0]),'caras')
    print('La distribución de la base de datos es:\n', data_csv.emotion.value_counts())
    emotion_list = list(data_csv.emotion)
    pixels_list = list(data_csv.pixels)

    # Se transforman las emociones a un array categorical
    emotion_array_np = np_utils.to_categorical(np.asarray(emotion_list), num_classes=NUM_CLASSES, dtype='float32')
    # Se vuelca el contenido de los píxeles a un numpy array
    images_array_np = create_images_array(pixels_list)
    return emotion_array_np, images_array_np
```

```
Comienza la importación de los datos
La base de datos contiene 35888 caras
La distribución de la base de datos es:
 3    8989
6    6198
4    6077
2    5121
0    4953
5    4002
1     547
```

Figura 5.14.- Fragmento de código: función load_data y salida

A partir de una función de la librería Pandas, se lee el contenido del csv. Este formato es semejante a una tabla y permite extraer información. Por ejemplo, el número de entradas que presenta la tabla o ver la distribución de los datos según la columna etiquetada como "emotion". Esta información se observa en el cuadro superior derecho de la figura. En el área enmarcada de color blanco se crean dos listas; una contendrá la columna de las emociones (manteniendo el orden original), y la otra representará todos los píxeles de cada imagen en formato string. La primera lista es de enteros, mientras que la segunda, es de strings.

Para utilizar los datos, se necesita transformarlos a vectores Numpy. Para el caso de las emociones, es sencillo, con la línea marcada con un tick verde, se logra la transformación. Se emplean las funciones internas de la librería para crear un array de tipo categorical, obteniendo "emotion_array_np" (np denota formato NumPy).

Para el caso de las fotografías se requiere una transformación más compleja. Con el símbolo amarillo, se señala la llamada a una función que creará las imágenes a partir de los píxeles.

```python
def create_images_array(pixels_list):
    #Construimos un vector de matrices, donde cada matriz será una foto
    images_array = []
    for image in pixels_list:

        image_matrix = np.zeros((IMG_SIZE, IMG_SIZE), dtype=np.uint8)
        pixels_array = image.split()

        for row_num in range(IMG_SIZE):
            start_row = row_num * IMG_SIZE
            finish_row = start_row + IMG_SIZE
            image_matrix[row_num] = pixels_array[ start_row : finish_row]

        images_array.append(np.array(image_matrix))

    images_array_np = (np.array(images_array)).astype('float32') / 255.0
    return images_array_np
```

Figura 5.15.- Fragmento de código: función para transformar las imágenes

Las imágenes entran a esta función como strings, donde los valores de los píxeles están separados por espacios. Se iterará sobre la lista de strings para transformar cada imagen. En cada iteración, se declara una matriz vacía de dimensión 48x48 para almacenar la imagen. Se fragmenta el string “image” logrando un array de enteros, es decir, toda la imagen representada en un vector operable.

En el bucle for de la línea 126, se comienza a rellenar la matriz vacía que ya tenía la dimensión preestablecida. El método consiste en rellenar cada fila de la matriz con lotes de 48 enteros extraídos del vector pixels. Los índices start_row y finish_row, marcarán el inicio y fin de cada lote, siendo inicialmente 0:48, después 48:96 y así sucesivamente. Cuando el bucle haya terminado, la matriz contendrá ya una imagen completa y se agregará al vector NumPy. Con estos pasos se va obteniendo una estructura más cercana a la esquematizada en la figura 5.12

Cuando el vector NumPy está completo, se reescalan los valores de los píxeles para que estén entre 0 y 1 en lugar de 0 y 255. Esta transformación hace más eficiente el entrenamiento.

Terminadas las funciones, el programa obtiene los dos NumPy arrays, representando todas las imágenes en uno y todas las emociones en otro. La posición 0 del vector de

las emociones es la etiqueta solución de la imagen ubicada en la misma posición del vector imágenes. Pese a encontrarse en vectores separados, el orden no ha cambiado.

5.2.3.- Transformación de los datos

El formato del vector que lleva las etiquetas de las emociones, no requiere de ninguna modificación dado que independientemente del tipo de entrenamiento, la solución siempre mantiene el mismo formato. Sin embargo, el vector de las imágenes, sí presentará alteraciones según la estructura del modelo que se pretenda crear.

Con el propósito de mantener un formato más estándar de los datos, el vector NumPy de imágenes ha de ser modificado. Según la documentación oficial de Keras [43], es recomendable que el vector NumPy de imágenes sea de 4 dimensiones o una tupla. La lógica de esto recae en que los diferentes formatos de una imagen siempre presentan varios canales de color. De esta forma, trabajar con el vector de imágenes en 3 dimensiones es poco coherente. Esa estructura ilógica dificultaría la escalabilidad del proyecto y el uso de distintas técnicas.

Pueden producirse dos transformaciones diferentes según las técnicas que se deseen utilizar:

- **Aplicar modelos pre entrenados**: es el caso de los modelos VGG16 o VGG19, su estructura impone el formato de tres canales de color. Esto implica una cuarta dimensión del vector, dando lugar a que las fotografías tengan el formato (n,48,48,3). Se transforma un vector de rango 3 a rango 4, añadiendo esos 3 canales de color que ya fueron explicados en la sección 5.2.1. El esquema de la figura 5.12, representa a la perfección un vector de estas características.
- **Aplicar preprocesado**: Keras ofrece herramientas para aplicar preprocesamiento a las imágenes del conjunto de datos. Las clases denominadas “ImageDataGenerator”, permiten aplicar cambios a las imágenes de entrenamiento para aportar más conocimiento. Para este tipo de clase, el input puede ser el directorio con las imágenes o un vector NumPy como es el caso. El vector ha de ser también de rango 4. Al tratarse de imágenes en blanco y negro, la documentación de Keras

describe que simplemente se añada el valor 1. Con esto, se especifica la presencia de un único canal de color y al mismo tiempo, se mantiene el formato de vectores de rango 4. Por tanto la estructura final se corresponde a (n,48,48,1).

5.2.3.1.- Formato de datos: modelos pre entrenados

Desde el punto de vista del formato de datos, aplicar un modelo pre entrenado produce un ciclo de transformación del vector imágenes. Como se aprecia en la figura 5.16, la función "apply_PreTrained_model" aplica el modelo pre entrenado a todas las imágenes almacenadas en el vector NumPy. El tipo de modelo se selecciona mediante la variable global "PRE_TRAINED_MODEL".

```
161     # Las capas VGG formarán parte de nuestro modelo, pero es un modelo preentrenado por lo que no deben entrenarse.
162     # Para reducir carga de procesado, pasamos las imagenes por el modelo VGG primero y evitamos incluir el VGG en el fit
163     images_array_np, PreTrained_Model = apply_PreTrained_model(PRE_TRAINED_MODEL,images_array_np)
164
```

Figura 5.16.- Fragmento de código: llamada a la función modelo VGG

Aplicar un modelo pre entrenado transforma las imágenes de entrada a formato de información más reducido y eficiente, este proceso se detallará en la sección 5.4.0.3. Desde el punto de vista de los datos, este proceso produce una transformación en el vector NumPy.

```
136 def apply_PreTrained_model(model,data):
137     # Se crea el modelo pre entrenado selecionado
138     # include_top=False no incluye las ultimas capas softmax
139     if model== "VGG16":
140         CNN_model = VGG16(include_top=False, input_shape=(48, 48, 3), pooling='avg', weights='imagenet')
141
142     if model== "VGG19":
143         CNN_model = VGG19(include_top=False, input_shape=(48, 48, 3), pooling='avg', weights='imagenet')
144
145     result = CNN_model.predict(data)
146
147     # Se aplica a los datos y se retorna el modelo para contruir el modelo final completo
148     return result, CNN_model
```

Figura 5.17.- Fragmento de código: función que aplica un modelo pre entrenado

Tras seleccionar el modelo pre entrenado que se desea utilizar, la función retornará dos variables. Por un lado, devuelve la estructura del modelo ya que será necesaria para la compilación final. Por otro lado, devuelve el nuevo vector NumPy. Este vector se genera

aplicando la inferencia del modelo VGG sobre el conjunto de datos. Transforma el vector del formato (35888,48,48,3) a (35888,512). La red neuronal diseñada, se entrenará para interpretar imágenes representadas en (512) puntos en lugar de matrices (48,48,3). Se tratará de encontrar relaciones entre esos 512 puntos y la etiqueta solución correspondiente a esa imagen.

La construcción final del modelo será la unión del modelo convolucional pre entrenado y la red neuronal particular. Esto implica que, en el momento de pruebas, las imágenes que tome la cámara, pasarán primero por el modelo VGG y después por la red neuronal. Como el modelo final ya representa la unión de los dos, no se supone un inconveniente.

Se estudiará en los próximos capítulos si estos nuevos puntos son o no una buena representación de las imágenes, ya que podrían fijar características irrelevantes, lo que daría a la red neuronal más dificultades para lograr un entrenamiento eficaz.

5.2.3.2.- Formato de datos: preprocesado de imágenes

Para aplicar preprocesado de imágenes, se utilizará la clase “ImageDataGenerator” de Keras. Esta herramienta permite crear nuevas imágenes durante el entrenamiento a partir de modificaciones a las ya existentes. El flujo de datos se realiza desde el NumPy array que ya presenta las dimensiones compatibles.

```
182
183     # Preprocesado de imágenes, para crear nuevas imágenes
184     train_datagen = ImageDataGenerator(
185         shear_range = 0.2,
186         rotation_range=10,
187         zoom_range=0.2,
188         horizontal_flip=True,
189         width_shift_range=0.1,
190         height_shift_range=0.1,
191     )
192
193
```

Figura 5.18.- Fragmento de código: características del preprocesado

En la figura 5.18, se detallan las características que se aplicarán a los datos. La variable "shear_range", estirará la imagen tras aplicarle la rotación especificada. Esto se asemeja a cambiar la perspectiva de las fotografías, creando nuevos ángulos de visión artificiales. La variable "rotation_range" aplica rotaciones de 10 grados en el plano 2D. De forma similar actúa la característica del zoom. "Horizontal_flip" gira la imagen por el eje horizontal, es decir, la información del lado derecho se observará ahora en el lado izquierdo y viceversa. Una imagen simétrica no se verá alterada a este efecto.

Las características "width_shift_range" y "height_shift_range", crean nuevas imágenes aplicando desplazamientos de un 10 % a la fotografía inicial. Todas estas modificaciones actuarán siempre de forma aleatoria dando lugar a nuevas imágenes.

```
207    data_train = train_datagen.flow(images_train, emotion_train, BATCH)
```

Figura 5.19.- Fragmento de código: aplicar preprocesado

En la figura 5.19, se aplica el método "flow" de la clase "ImageDateGenerator" creada. Este método, toma como argumentos el tamaño deseado de los lotes de datos y los vectores NumPy de entrenamiento. El vector de emociones no sufrirá ninguna modificación, pero es necesario incluirlo ya que ha de conocerse la solución de cada imagen porque será también la solución de la nueva imagen artificial.

Es de interés mencionar que no se aplican modificaciones a las imágenes de pruebas. Estas no aportarán conocimiento al modelo por lo que no es necesario incrementar su número o transformarlas, representan un reto de predicción por sí mismas.

5.2.3.3.- Formato del modelo: entrada y salida

Abstrayendo el funcionamiento del modelo así como su estructura interna, se puede explicar el concepto de entrada y salida. En lo referente a esta sección que abarca el formato de los datos, es de interés primordial comentar las estrictas compatibilidades que han de mantenerse. En secciones anteriores, no se ha incidido en el formato de salida del modelo dada su sencillez. Por su condición de resultado, se espera siempre el formato

correspondiente al número que representa cada emoción. Esta estructura es invariante pese a la diversidad de estructuras con las que se trabajará.

En la construcción del modelo, se plantea una estructura interna que afecta al rendimiento y a la eficiencia, pero no supone un trabajo de adaptación si se utiliza adecuadamente el formato de entrada y salida. Como se detalló en este capítulo de transformación de datos, el input del modelo es un vector NumPy. Se distinguían dos tipos de inputs:

- **Vectores con 3 canales de color**: esta estructura se empleaba para el uso de un modelo pre entrenado. Para ello, se añadía un rango al vector dando lugar a la estructura (n,48,48,3), donde n representa el número de imágenes. El proceso de compilación del modelo es la concatenación de los dos modelos utilizados. Es decir, el modelo final estará formado por el pre entrenado seguido de la red neuronal. Esto implica que el output del modelo pre entrenado es el input de la red neuronal que se pretende entrenar.

```
231     # CONSTRUCCION DEL MODELO COMPLETO
232     neuronal_input = PreTrained_Model(initial_inputs)
233     model_output = neuronal_network(neuronal_input)
234
235     # Pre trained model + neuronal_network
236     convolutional_model = Model( initial_inputs, model_output)
237     convolutional_model.compile(loss='categorical_crossentropy', optimizer=Adamax(), metrics=['accuracy'])
238
272     #GUARDADO DEL MODELO Y FIGURA
273     epochs = str(len(accuracy))
274     a = round(results[1], 3)
275     precision = str(a)
276
277     name = "imagen_e_" + epochs + "-acc_" + precision +".png"
278     model_name = "model_e_"+ epochs + "-acc_" + precision +".hdf5"
279     try:
280         plt.savefig(SESSION_PATH + name )
281         convolutional_model.save(SESSION_PATH + model_name )
282     except Exception:
283         print("Error al guardar el modelo y el gráfico \n", traceback.print_exc())
284
```

Figura 5.20.- Fragmento de código: compilado y guardado del modelo final

En el primer recuadro blanco de la figura 5.20, se observa la unión de las dos estructuras. Tras haber sido compilado, se guardará en un archivo. En el segundo recuadro, se observa la sentencia de guardado, donde se especifica el nombre del archivo incluyéndolo en la ruta donde se desea guardar.

- **Vectores con 1 canal de color**: se añadía un rango al vector para obtener la estructura (n,48,48,1) donde n representa el número de imágenes. Para este caso, como no se aplicaban modelos pre entrenados, la construcción final y guardado del

modelo es sencilla. Solo ha de ser compilado y guardado como se muestra en el segundo recuadro blanco de la figura 5.20.

El formato de entrada y salida que presente el modelo para la etapa de entrenamiento, ha de ser el mismo que el empleado para los procesos de inferencia. En otras palabras, si el modelo se entrena a partir de imágenes de 48x48 píxeles, es lógico que sus procesos de predicción han de efectuarse sobre imágenes de las mismas dimensiones.

En el programa de predicción, la cámara tomará fotografías de las cuales, gracias a OpenCv, se extraerá un rostro. Las dimensiones de esa cara deben cumplir el formato del vector NumPy. Una imagen en formato matricial (48,48) produciría una incompatibilidad, ya que no corresponde con el input que espera el modelo.

```
47          #Adaptación de la dimensión
48          image = cv2.resize(roi_gray, (48, 48))
49          np_image_array = image.astype('float32') / 255.0
50          for index, item in enumerate(vg_input):
51              item[:, :, 0] = np_image_array
52              item[:, :, 1] = np_image_array
53              item[:, :, 2] = np_image_array
54
55          prediction = CNN_model.predict(vg_input)
56
```

Figura 5.21.- Fragmento de código: adaptación de imagen para realizar la inferencia

En la figura 5.21, que corresponde al programa de predicción, se muestra la transformación que se realiza a la imagen. El objetivo es adaptarla a los formatos de input que ya fueron ampliamente detallados. Particularmente, este ejemplo corresponde a un modelo VGG, por lo que el input seguirá la estructura 5.4.0.1.

La figura muestra el fragmento de código de interés abstrayendo el funcionamiento del programa de predicción ya que no corresponde a este capítulo.

5.3.- Tensorflow introducción

TesorFlow es la API principal bajo la que se desarrolla la IA del proyecto. Como la definen sus creadores de Google, se trata de una plataforma de código abierto de

extremo a extremo para diseñar y entrenar modelos de aprendizaje automático. Posee un ecosistema completo de herramientas, bibliotecas o recursos que permitirán la innovación e implementación de aplicaciones ML de forma sencilla [44].

5.3.1.- Keras y Tensorflow

Para desarrollar la API de Tensorflow es de gran interés describir Keras. "Keras es una API diseñada para ser utilizada por humanos y no para máquinas" [45]. Así se describe en su documentación oficial, pues se trata de una API de alto nivel. En los inicios del Deep Learning, el desarrollo de programas era semejante al lenguaje ensamblador. Crear redes neuronales era un proceso tedioso y largo, donde resultaba difícil identificar los errores. Para dar solución a este conflicto de trabajo, François Chollet, un ingeniero de Google, diseñó Keras en 2015. Desde sus inicios, esta API estaba destinada a actuar como interfaz de alto nivel capaz de abstraer los procesos computacionales que ocurrían detrás, pudiendo utilizarse diferentes backends.

Por otra parte, en los inicios de Tensorflow libre en 2015, se ofrecía la capacidad de diseñar redes convolucionales en su pura esencia. Su sintaxis inicial destacaba por ser poco intuitiva y con una depuración imposible. Las variables y tensores se creaban manualmente y se debía emplear grafos de computación bajo sesiones [46]. El potencial de esta API era inmenso, comenzando a desarrollarse miles de proyectos dando vida a la inteligencia artificial.

El futuro y la conexión de Keras con Tensorflow era evidente. Se comenzaron a emplear de forma conjunta, aunque con importaciones separadas hasta su unión oficial. En la versión de TF 2.0, Keras se convertiría en el backend oficial de Tensorflow unificando sus caminos de desarrollo y cerrando la posibilidad de utilizar Keras con otros backends de la competencia.

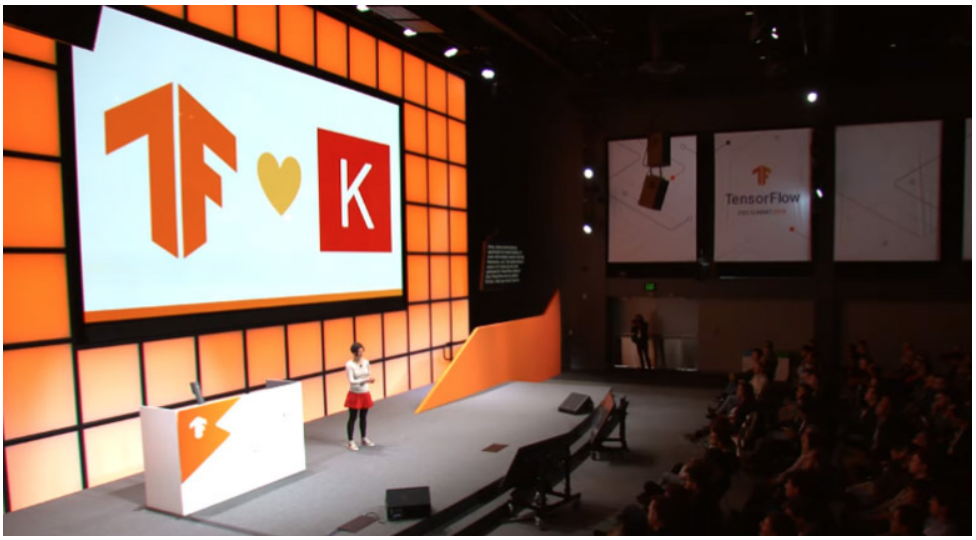


Figura 5.22.- Conferencia de Google. Fuente: Web Keras [45]

Aunque en este proyecto se describan las herramientas de Google, en esta descripción del marco del Machine Learning, ha de mencionarse la existencia de competencia. PyTorch es otra herramienta muy potente diseñada por Facebook con propósitos semejantes, que compite por el liderazgo del sector.

5.4.- Tensorflow: arquitectura

En este proyecto se trabaja con diferentes versiones de Tensorflow por el uso de distintos equipos. Dado que todas se encuentran por encima de la 2.0, no se producen incompatibilidades. Las versiones de Python utilizadas son la 3.7 y superiores.

La red neuronal se nutre de un conjunto de datos. Para todas las versiones creadas, siempre se emplea el mismo dataset, aquel que ya fue descrito y analizado en el capítulo 5.1. Sin embargo, la estructura de las redes neuronales, así como su tipo, serán diferentes. El propósito de esto será hacer una comparativa en busca de mayor eficiencia en la interpretación de los rostros. En el capítulo 6, se encontrarán estos estudios de eficacia.

Hasta ahora se ha hecho una descripción esquemática de las redes neuronales. Dicho enfoque esquemático es muy útil para tener una visión estructural. Se continuará empleando en la comparativa de las diferentes redes que se han diseñado en el proyecto.

Sin embargo, ha de ser detallada la transcripción de esa visión estructural al código programado en Python. Este es el propósito de la sección, facilitar al lector la comprensión del código especificando puntos de interés y dejando al margen detalles que pertenecen a otras funcionalidades.

5.4.0.1.- Tensorflow primeras capas: modelo VGG

Inicialmente, se detallará la sintaxis de interés sobre la primera versión del modelo construida. Como ya se viene describiendo en capítulos anteriores, el uso de una red neuronal convolucional pre entrenada puede ser combinado con una red neuronal simple. Esta arquitectura será descrita desde un enfoque de programación para detallar características que no figuran en el marco teórico.

Dentro de la librería de Tensorflow, bajo su importación, pueden utilizarse los modelos convolucionales pre entrenados. El concepto teórico de la red convolucional ya fue descrito en el capítulo 4.3.2. Estas redes se emplean de forma frecuente en algoritmos que trabajan con imágenes ya que es donde muestran gran eficiencia.

Como a lo largo del proyecto se pretenden probar diferentes arquitecturas, los diseños están controlados en mayor medida por funciones y variables globales. De está forma se evitan números mágicos o especificaciones manuales de dimensiones.

En este caso, la construcción comienza con la función “apply_PreTrained_mode”, donde según el interés, se incorpora el modelo VGG16 o 19.

```
136  def apply_PreTrained_model(model,data):
137      # Se crea el modelo pre entrenado selecionado
138      # include_top=False no incluye las ultimas capas softmax
139      if model== "VGG16":
140          CNN_model = VGG16(include_top=False, input_shape=(48, 48, 3), pooling='avg', weights='imagenet')
141
142      if model== "VGG19":
143          CNN_model = VGG19(include_top=False, input_shape=(48, 48, 3), pooling='avg', weights='imagenet')
144
145      result = CNN_model.predict(data)
146
147      # Se aplica a los datos y se retorna el modelo para construir el modelo final completo
148      return result, CNN_model
149
```

Figura 5.23.- Fragmento de código: creación red convolucional VGG

En la figura 5.23, se pretende incorporar el modelo VGG16. Tras su primera ejecución, el modelo se descargará junto a los pesos asociados a las neuronas. Esto significa que la red ya está entrenada, de ahí su denominación. Se trata de una red convolucional muy profunda con 16 capas, 13 convolucionales y 3 densas. Su entrenamiento sería muy largo y con requisitos computacionales elevados. Es por eso que se emplea habitualmente con los pesos ya cargados mediante el parámetro "weights='imagenet".

La red fue entrenada con bases de datos muy grandes en ImageNet. El propósito de los creadores del modelo era lograr que las 13 capas convolucionales realizaran una extracción de características significativas a las imágenes. Tras recorrer esas 13 capas, las imágenes cambian a un formato nuevo que proporciona un input para la red neuronal de 3 capas. Estas tres últimas capas no se utilizarán ya que se desea particularizar las predicciones a las categorías particulares, las emociones. La exclusión de estas últimas capas se indica mediante el parámetro "include_top = False". Además en el parámetro "input_shape", se especifica la dimensión de cada imagen de forma individual (48,48,3), es decir, el conjunto se presentará como (n,48,48,3).

Interpretar imágenes presenta una gran dificultad para un computador con métodos clásicos. Es aquí donde la extracción de características adquiere vital importancia, transformando un handicap de la computación, en una tarea simple. Para visualizar el concepto puede plantearse el ejemplo de un reto. Un humano ha de elaborar una lista de datos con respuesta binaria, es decir sí o no. Otra persona, tras leer e interpretar la lista, ha de adivinar qué se está describiendo en ella.

Su objetivo será categorizar la información para un producto del supermercado. Para ello, tras observar muchas imágenes de las distintas zonas del supermercado, comienza a elaborar su lista. Producto cuadrado, producto rectangular, producto de vidrio, presencia de color...

Tras tener la lista completa, se procede a transcribir la primera imagen, que en este caso contiene una naranja. El primer humano observa la imagen de la naranja y rellena su exhaustiva lista donde figurará información como: bordes redondeados sí, color naranja sí, rectangular no, grande no, simétrico sí...

La segunda persona, cansada tras leer la interminable lista, ofrece el resultado que ya sospechaba apenas comenzó a leer: ¡es una naranja!. Toda la información que aportaban las imágenes en 4k que observaba el primer humano, se podían resumir perfectamente en la tabla binaria.

Este es un símil del funcionamiento del modelo VGG16, sus 13 capas convolucionales leen los píxeles de entrada para ofrecer la información relevante en un formato más sencillo de utilizar. Con las imágenes en ese nuevo formato, es mucho más asequible el entrenamiento de una red neuronal simple, ya que el problema ahora posee una naturaleza más cercana a la computacional.

Volviendo a la estructura del código, una sintaxis posible sería unir previamente el modelo VGG16 (sin las 3 últimas capas) a la red neuronal particular. Tra unificarlos en un modelo, se podría ejecutar el entrenamiento de este diseño. Durante el proceso de aprendizaje, se pretendería modificar los pesos de cada neurona de cada capa lo que sería un proceso eterno para una red tan profunda. También podrían establecerse las 13 primeras capas como no entrenables para fijar su peso. Aún así, en cada iteración del entrenamiento, se debería recorrer todas las capas lo que ralentizaría mucho el proceso y cargaría a la memoria.

Por ello, la mejor opción es la empleada en el recuadro blanco de la figura 5.24. Mediante "result=CNN_model.predict(data)", toda la base de datos se transforma al formato de salida VGG16 (sin las tres últimas capas). Este formato de salida corresponde a un vector NumPy (35888,512) que ya fue mencionado en las sección .

```
136  def apply_PreTrained_model(model,data):
137      # Se crea el modelo pre entrenado selecionado
138      # include_top=False no incluye las ultimas capas softmax
139      if model== "VGG16":
140          CNN_model = VGG16(include_top=False, input_shape=(48, 48, 3), pooling='avg', weights='imagenet')
141
142      if model== "VGG19":
143          CNN_model = VGG19(include_top=False, input_shape=(48, 48, 3), pooling='avg', weights='imagenet')
144
145      result = CNN_model.predict(data)
146
147      # Se aplica a los datos y se retorna el modelo para contruir el modelo final completo
148      return result, CNN_model
```

Figura 5.24.- Fragmento de código: creación red convolucional VGG

5.4.0.2.- Tensorflow últimas capas: red neuronal

Con la base de datos transformada al nuevo formato, puede introducirse en le input de la red neuronal. La sintaxis para crear las capas de la red se observa en la figura 5.25. Inicialmente se establece que la red sera de tipo secuencial. Existen dos tipos de modelos; funcionales y secuenciales. La estructura secuencial establece que las capas serán concatenadas mientras que el modelo funcional, permite conectar las capas cualesquiera entre sí.

Dada su estructura secuencial, la red neuronal presentará las capas en el orden en el que se agreguen. De esta forma, cada capa solo podrá estar conectada con la capa anterior y posterior.

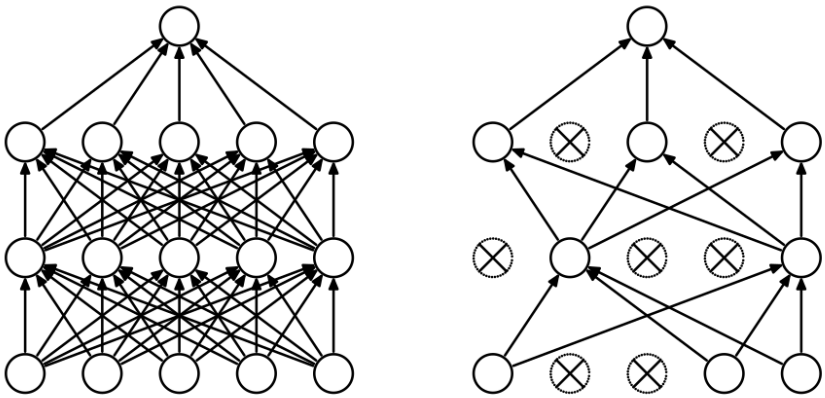
Las tres capas creadas en la figura 5.23, poseen neuronas de tipo dense [47]. La primera capa contiene 256 neuronas y ha de indicarse el formato que tendrán los datos al entrar a la red neuronal. El valor se especifica en el parámetro "input_shape", mediante np.shape(images_train[0]), que devuelve la dimensión del vector NumPy. En este caso, el formato de cada entrada corresponde a (512) valores. Al obtenerse de forma dinámica, algún cambio del formato, cambiaría también el "input_shape".

```
180    #Construimos la red neuronal
181    neuronal_network = Sequential()
182
183    #Completamos el modelo con nuestras capas
184    neuronal_network.add(Dense(256, input_shape= np.shape(images_train[0]) , activation='relu'))
185    neuronal_network.add(Dropout(0.2))
186
187    neuronal_network.add(Dense(128, input_shape= (256,) , activation='relu'))
188    neuronal_network.add(Dropout(0.2))
189
190    neuronal_network.add(Dense(64, input_shape=(128,), activation='relu'))
191    neuronal_network.add(Dropout(0.2))
192
```

Figura 5.25.- Fragmento de código: creación de la red neuronal

Las funciones Dropout que se añaden a cada capa son una técnica de regulación que producen la desactivación de neuronas durante el entrenamiento. En cada entrada al entrenamiento, aleatoriamente se apagaran el porcentaje de neuronas especificado. De esta forma, se evita que las neuronas encuentren caminos predeterminados hacia los resultados. Su funcionamiento se esquematiza en la figura 5.23.

La segunda capa tiene 128 neuronas y la tercera 64. Los formatos de entrada de la capa 2 y 3 se han especificado para facilitar al lector su comprensión. Cada neurona está conectada a todas las neuronas de la capa superior e inferior. Al tratarse de una estructura secuencial, el número de neuronas de la primera capa es de forma impuesta el "input_shape" de las neuronas de la segunda capa y lo mismo sucede con la capa 3. La función de activación utilizada es "relu".



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

Figura 5.26.- Ejemplo del efecto Dropout. Fuente: publicación científica [48]

La última capa de neuronas se observa en la figura 5.26. El número de neuronas se corresponde con el número de clases por eso se especifica mediante la variable global NUM_CLASSES. La función de activación debe ser Softmax, ya que es la que ofrece la distribución de probabilidad de cada clase. En otras palabras, mide que probabilidad hay asociada a cada solución y selecciona como respuesta la más alta.

```
192
193     neuronal_network.add(Dense(NUM_CLASSES, activation='softmax'))
194
```

Figura 5.27.- Fragmento de código: creación de la red neuronal

5.4.0.3.- Tensorflow: compilación del modelo

Tras la construcción del modelo, debe realizarse la compilación. Este proceso es previo al entrenamiento de la red neuronal y se representa en la figura 5.28.

```
195     neuronal_network.compile(loss='categorical_crossentropy',
196                              optimizer=Adamax(lr= LEARNING_RATE, decay= DECAY),metrics=['accuracy'])
197
```

Figura 5.28.- Fragmento de código: compilación

Los parámetros especificados son:

- **"Metrics"**: representa la métrica empleada en el modelo para analizar su mejora. La clase accuracy calcula la frecuencia con la que las predicciones coinciden con las etiquetas solución [49]. Esta clase creará dos variables locales una para llevar el conteo total de predicciones y otra para cuantificar aquellas que fueron exitosas. Particularizando al programa, tras leer una imagen del vector NumPy y procesarla por toda la red neuronal, la función Softmax ofrecerá la probabilidad asociada a cada emoción. La probabilidad más alta será seleccionada como resultado. Tras ofrecer la respuesta, se comprueba mediante la etiqueta solución. Accuracy, incrementará su variable "total" y de ser exitosa, incrementara también "count". Esta métrica es lo que se conoce como media aritmética.
- **"Loss"**: el parámetro de pérdida es de gran importancia para el modelo ya que es su método principal de mejora. En el proceso de compilación ha de especificarse el tipo de pérdida con el que se trabajará en el entrenamiento. El problema descrito en este proyecto es conocido como "Muliti-class classification", que implica la existencia de múltiples clases. El dataset describe 7 emociones diferentes, presentándose solo una en cada imagen. Este tipo de resultado se denomina "One-hot vector" [50]. Una imagen neutral presenta su solución como un vector (0, 0, 0, 0, 0, 0, 1), todos los elementos valdrán 0 excepto uno, que será representado por el valor 1.

  Para mejorar, una red neuronal ajusta sus pesos al terminar cada época, del entrenamiento. Para conocer la magnitud del ajuste que ha de realizarse a cada segmento de la red neuronal, se utiliza la función de pérdida. Esta función busca cuantificar el error asociado a cada configuración de la red neuronal. Sin entrenar en cálculos complejos, minimizar está función de pérdida mediante variaciones de los pesos, implica mejorar la red.

"Categorical_crossentropy" es la función que se utiliza frecuentemente en modelos de "Muliti-class classification" [51]. Podría traducirse como función de pérdidas de entropía cruzada. Como se explicaba en el punto anterior, el modelo aporta su resultado mediante las 7 neuronas finales con función Softmax.

Estas neuronas pueden verse como portavoces de cada categoría. Su tarea es ofrecer que probabilidad creen que hay de ser su categoría la solución asociada a la imagen. El caso idílico es aquel donde una etiqueta muestra el 100 %, y el resto un simple 0. Aplicar la función de error cruzada implica que todas las categorías influyen entre sí. Es decir, si el resultado esperado de una categoría era 1, estrictamente las demás deberían haber mostrado 0.

Esta función de error aplica una "penalización" a cada rama de la red neuronal partiendo de las 7 neuras "portavoces". Este proceso se denomina: back propagation, del final hacia el principio. La penalización se ajusta mediante escala logarítmica por lo que es mayor si el porcentaje aportado dista mucho del esperado. De igual forma, la penalización es baja si la probabilidad de esa categoría se encontraba próxima a lo esperado.

- **"Optimizer"**: el optimizador es la clase que desempeña los cambios en los pesos de las neuronas con el fin de mejorar el modelo. El optimizador más habitual y con el que se trabaja en este proyecto es Adamax, una extensión del algoritmo Adam. La optimización del modelo se consigue mediante el método descenso de gradiente [52]. Este es el proceso más complejo en cuanto a términos matemáticos se refiere. El optimizador alterará los valores de las variables (pesos) y analizará sus efectos en la función de pérdidas. En dos dimensiones, la función de pérdidas podría ser por ejemplo, una función convexa. Si se evalúa la pendiente a lo largo de la función, es conocido que valdrá cero cuando se alcance un punto mínimo. Además, dado el ejemplo de función convexa, por su naturaleza, ese mínimo será global. En otras palabras, ese punto representa el valor más bajo de la función.

  Teniendo en cuenta que lo representado es el error, es decir, algo indeseado, hallar su punto mínimo es la mejor posición posible. Los pesos de las variables asociados a ese punto, serán los mejores, ya que producen un error mínimo. Al aumentar las dimensiones, la pendiente se transforma en el vector gradiente.

Pude visualizarse el mismo efecto en 3 dimensiones imaginando un relieve montañoso. El objetivo, será descender por las montañas para encontrar el punto más bajo. En este caso, el vector gradiente nos indica la dirección en la que la pendiente asciende. Como el propósito es hallar el mínimo global, se toma el sentido opuesto y así se logra descender por la superficie.

Llegado a este punto, el lector puede plantearse lo sencillo que es simplemente colocarse en el punto mínimo y el modelo sería perfecto. El problema es que las dimensiones con las que el optimizador debe lidiar son demasiadas como para calcular todos los puntos. En otras palabras, el optimizador se mueve a ciegas por la superficie multidimensional, resultando un terreno inexplorado.

El optimizador avanza en el sentido contrario al gradiente mediante la modificación de las variables, de esta forma, consigue reducir progresivamente el error. La cantidad de cambios que efectúa sobre las variables se especifica por el parámetro "learning_rate".

Esta tasa de aprendizaje ocupa gran importancia para el modelo. Como se ha dicho, el optimizador conoce la dirección hacia donde debe moverse. Si la tasa de aprendizaje es pequeña, las nuevas variables seguro supondrán una mejora del modelo, pero su entrenamiento sería eterno. El modelo debería repetir una y otra vez esos diminutos cambios.

Por otra parte, una tasa de aprendizaje alta, supone una gran mejora si la dirección de descenso se mantiene tras los cambios. En otras palabras, si el salto es demasiado grande, podría pasarse del punto mínimo y encontrarse perdido en otra ubicación incluso peor que la anterior.

La solución más lógica es emplear el parametro "decay" que actúa reduciendo la tasa de aprendizaje con el paso de las épocas. Es de gran utilidad ya que el modelo comienza con variables aleatorias, es decir, con un error enorme. Esto implica que al principio, el modelo resultará muy fácil de mejorar. Los primeros saltos de las variables, trasladarán al modelo a puntos mínimos por lo que interesa que sean grandes.

Tras numerosas épocas, el problema de minimizar el error comenzará a ser muy complejo. Los puntos mínimos serán numerosos y próximos por lo que interesan las transformaciones pequeñas.

Esta reducción del "learning rate" facilita ponerle fin a un entrenamiento y conseguir resultados concluyentes.

5.4.0.4.- Tensorflow: entrenamiento del modelo

Para el proceso de entrenamiento del modelo, previamente debe fragmentarse la base de datos. Es un proceso muy sencillo ya que los datos se encuentran en el formato correcto solo que se pretenden emplear para tareas diferentes. Como ya fue mencionado, un 80 % de datos deben ser destinados al entrenamiento. De estos datos el modelo se nutrirá y conseguirá mejorar progresivamente. El 20 % restante ha de destinarse a pruebas, evitando que la red neuronal los utilice para mejorar. El propósito de esto es lograr una validación más realista donde las imágenes a clasificar siempre resulten novedosas para el modelo. Esta división se visualiza en la figura 5.29, donde "NUM_TRAIN" representa el índice donde terminan las imágenes catalogadas para entrenamiento. Por consiguiente, las imágenes de pruebas comienzan en el índice "NUM_TRAIN +1". Con la misma lógica, se fragmenta el vector con las etiquetas solución.

```
171    #SE DIVIDE EN:
172    #TRAIN
173    emotion_train = emotion_array_np[0:NUM_TRAIN]
174    images_train = images_array_np[0:NUM_TRAIN]
175
176    #TEST
177    emotion_test = emotion_array_np[NUM_TRAIN+1:]
178    images_test = images_array_np[NUM_TRAIN+1:]
179
```

Figura 5.29.- Fragmento de código: fragmentación de los datos

Cuando el conjunto de datos ya está separado en entrenamiento y pruebas, puede construirse y ejecutarse el entrenamiento. En la documentación oficial y en otros proyectos de Tensorflow se habitúa a llamar a las variables de entrenamiento x e y. En este caso, se

considera que será más sencillo de comprender para el lector la terminología “image_train” y “emotion_train”. Funcionalmente, la relación entre la imagen y su emoción puede verse como una función, de ahí esa terminología [53].

Si se emplea un preprocesado de las imágenes mediante las clases ya descritas, únicamente se pasa al entrenamiento el objeto “data_train”. Esta estructura se detalló en la figura 5.19.

```
221     #ENTRENAMIENTO
222     model_info = neuronal_network.fit(
223             images_train, emotion_train,
224             batch_size=BATCH,
225             epochs=EPOCH,
226             validation_data=(images_test, emotion_test),
227             verbose=1,
228             callbacks= callbacks
229             )
230
```

Figura 5.30.- Fragmento de código: entrenamiento

En la figura 5.30, se observa el método “fit” que recoge algunos parámetros de interés. El parámetro “epochs” representa el número de épocas que ejecutará el entrenamiento. Una época corresponde a una iteración por todo el conjunto “images_train” y “emotion_train” proporcionado. “Batch_size” representa cada cuántas imágenes de entrada se ajusta el vector gradiente.

“Validation_data” corresponde a las imágenes de validación. Al terminar cada época el modelo calculará la función de pérdidas y otros parámetros con estos datos. Como ya se mencionó, no aportan aprendizaje, solo métricas. “Verbose” se emplea para definir cuánta información se desea proyectar en la terminal durante el entrenamiento. Sus valores pueden ser 0, 1 o 2.

“Callbacks” representan las llamas que se ejecutarán durante el entrenamiento. Esto permite añadir operaciones o procesos intermedios mientras la red neuronal está entrenando. Esta herramienta abre la posibilidad de agregar numerosas funcionalidades que se comentarán en la siguiente sección.

5.4.1.- Tensorflow: funcionalidades del programa

El proceso de entrenamiento, como ya se ha detallado, se construye mediante muchos hiperparámetros y estructuras diferentes. Además, según el número de épocas, puede ser de larga duración, siendo crítico a fallos o resultados inesperados.

Para efectuar entrenamientos más robustos, se dota al script de Python de muchas funcionalidades que aportan seguridad e información. En secciones anteriores ya se han descrito funciones empleadas para tareas como aplicar modelos pre entrenados o realizar la lectura de los datos. Además, la mayor parte de los parámetros como dimensiones o valores, se obtienen de forma dinámica o con variables globales.

A simple vista, estas buenas prácticas pueden verse como algo irrelevante. Tras efectuar muchas pruebas y cambios, este dinamismo permite adaptar el modelo a muchos formatos sin apenas realizar modificaciones extras, marcando una gran diferencia.

5.4.1.1.- Callbacks

```
#CALLBACKS
checkpoint_name = 'val_acc_{val_categorical_accuracy:.4f}-{epoch:02d}.hdf5'
checkpoint_filepath = CKPT_DIR + checkpoint_name

csv_logger = tf.keras.callbacks.CSVLogger(LOG_CSV, append=False)
early_stop = tf.keras.callbacks.EarlyStopping('val_loss', patience=PATIENCE)
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_filepath, monitor='val_categorical_accuracy', verbose=1, mode='max', save_best_only=True)

callbacks = [myCallback(), tensorboard, model_checkpoint, csv_logger]
```

Figura 5.31.- Fragmento de código: Callbacks

En la figura 5.31, se crean tres objetos que representan funcionalidades callbacks. El primero “CVSLogger”, almacena las métricas del entrenamiento en un archivo. De esta forma, pueden crearse representaciones acerca de los cambios que experimentaron, por ejemplo, las pérdidas por época [54].

El objeto “early_stop”, permite detener el entrenamiento cuando los resultados obtenidos no estén cumpliendo con unas características deseadas. Especificando la métrica que se desee controlar, el objeto saltará a detener el aprendizaje cuando el modelo no esté mejorando. Es habitual que, por ejemplo, la métrica de pérdidas no mejore en todas las épocas, sobre todo al pasar las primeras iteraciones. Para evitar paradas inoportunas,

se especifica en el parámetro “Patience” el número de épocas consecutivas que deben trascurrir sin ninguna mejora para que se interrumpa el aprendizaje.

El objeto “model_checkpoint” es uno de los callbacks más importantes. Este objeto efectúa copias del modelo durante el entrenamiento. Gracias a esto, los pesos calculados no se perderán cuando el programa rompa por algún error o se detenga por una causa desconocida. En el objeto se ha de indicar el nombre del archivo en formato de ruta, que es donde se almacenará. Además, se selecciona que solo se guarde un checkpoint cuando el modelo haya mejorado en base a la métrica especificada en “monitor”.

Para otorgar mayor información y organización de los checkpoints, se nombran con el número de época en el que fueron creados y su precisión. De esta forma, cada copia de seguridad del modelo tendrá un nombre diferente y mediante este, se podrá conocer su eficiencia.

Adicionalmente, se desarrolla una clase particular, “myCallback()”. Está permitirá crear funciones adicionales a las que facilita la API de forma predeterminada. Su funcionamiento se observa en la figura 5.32.

```
71  #CLASE
72  class myCallback(tf.keras.callbacks.Callback):
73      def on_epoch_end(self, epoch, logs={}):
74          if(epoch % 5 == 0) :
75              files_to_delete = os.listdir(SESSION_PATH + "ckpt\\")
76              #Ordenamos la lista de ficheros alfabeticamente
77              files_to_delete.sort(reverse=True)
78              #Solo puede haber dos ficheros, csv.log y el mejor checkpoint
79              while(len(files_to_delete)>1):
80                  #Se borra el archivo
81                  try:
82                      remove(SESSION_PATH + "ckpt\\" + files_to_delete[1])
83                      del(files_to_delete[1])
84                  except Exception:
85                      print("Un archivo aún no ha podido ser eliminado, se eliminará más tarde")
86
87          #sys.stdin.flush()
88          if keyboard.is_pressed('ç'):
89              print('Se aborto manualmente el entrenamiento')
90              self.model.stop_training = True
91
```

Figura 5.32.- Fragmento de código: Callback personalizado

Tras ejecutar los primeros entrenamientos, con un número de épocas elevado como 10.000, surgió la necesidad de los checkpoints. La duración de este entrenamiento podía ser de un día completo, aplicando toda la potencia computacional que se explicará más adelante.

Por alguna circunstancia extraña o inesperada, ya sea una caída de la luz o un error de la memoria RAM, el programa se detenía y se perdía todo el conocimiento. Los checkpoints solventaron perfectamente este inconveniente, pero dieron lugar a otro. Las excesivas copias de seguridad que se almacenaban. Como cada incremento de la métrica "accuracy" provocaba una copia de seguridad, el directorio se llenaba de archivos. Con el propósito de mejorar este sistema, se define el objeto "myCallback()" de forma particular. Mediante su método incorporado, "on_epoch_end", puede ejecutarse cualquier instrucción al terminar cada época.

Así, cada n número de épocas deseado, se procederá a ejecutar una limpieza de las copias de seguridad obsoletas. Carece de sentido almacenar una copia de seguridad de la primera época cuando ya existe una mejor. Se obtendrá una lista de los archivos existentes en el directorio ordenados de forma inversa para mantener el mejor. En las figurarse 5.33 y 5.34, se puede visualizar el directorio antes y después, gracias a utilizar breakpoints.

```
71  #CLASE
72  class myCallback(tf.keras.callbacks.Callback):
73      def on_epoch_end(self, epoch, logs={}):
74          if(epoch % 10 == 0) :
75              files_to_delete = os.listdir(SESSION_PATH + "ckpt\\")
76              #Ordenamos la lista de ficheros alfabeticamente
77              files_to_delete.sort(reverse=True)
78              #Solo puede haber dos ficheros, csv.log y el mejor checkpoint
79              while(len(files_to_delete)>1):
80                  #Se borra el archivo
81                  try:
82                      remove(SESSION_PATH + "ckpt\\" + files_to_delete[1])
83                      del(files_to_delete[1])
84                  except Exception:
85                      print("Un archivo aún no ha podido ser eliminado, se eliminará más t
```

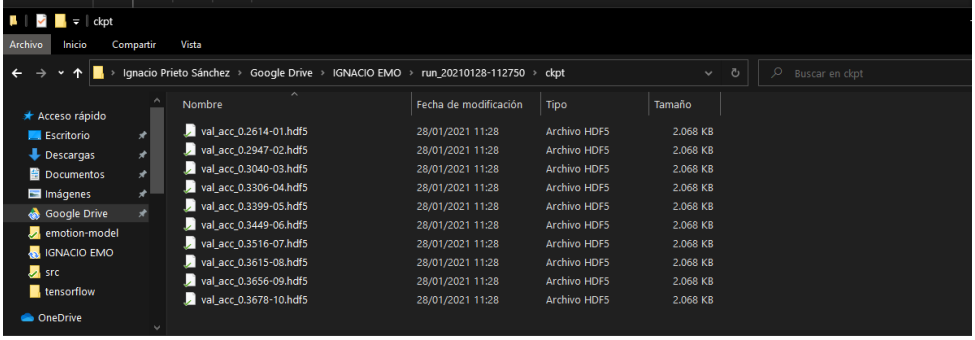


Figura 5.33.- Fragmento de código: directorio antes de la limpieza

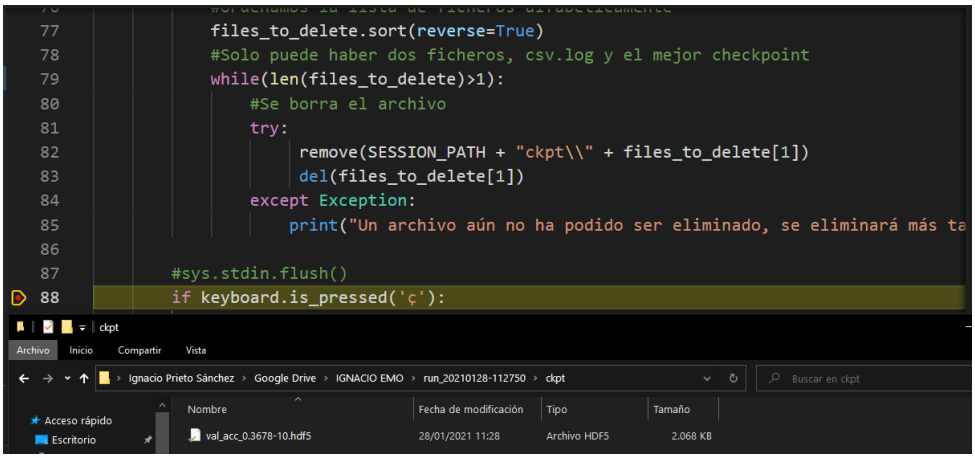


Figura 5.34.- Fragmento de código: directorio tras la limpieza

Como se observa en la imagen 5.34, se han eliminado todos los modelos manteniendo el que mostraba mayor "accuracy". Finalmente, también se incluyó la posibilidad de finalizar el entrenamiento de forma manual sin romper el programa. Tras cada época, se comprobará que la letra "ç" no esté presionada. En caso afirmativo, el entrenamiento se detiene de forma manual como si hubiera alcanzado la última época. Si se pretende efectuar un estudio controlado del modelo, esta funcionalidad es mucho más recomendada que un "early_stop", donde la interrupción es automática en lugar de manual.

Existe un último elemento con características de callback que aún no ha sido descrito, Tensorboard. Se trata de un generador de gráficas, que dada su potencia y funcionalidad, se describirá en una sección aparte.

5.4.1.2.- Organización de archivos

Dado que se construyen diferentes versiones de entrenamiento mediante copias del archivo "train.py", su organización es fundamental. Ya han sido detallados los números hiperparámetros que pueden modificarse, además de estructuras y tipos de neuronas. Si se pretende realizar una evaluación final y comparativa de resultados, el orden es imprescindible.

La organización empleada consiste en colocar todas las versiones de entrenamiento en el mismo directorio así como una única copia de la base de datos. Todos los scripts podrán acceder a dicha base de datos y plasmar su información en el mismo directorio.

```
40  # DIRECTORIOS
41  CSV_FILE = 'dataset.csv'
42  TIME = datetime.datetime.now().strftime("%Y_%m_%d-%H_%M_%S")
43  SESSION_PATH= "run_" + TIME +"\\"
44  LOG_CSV = SESSION_PATH + "log_csv" + ".log"
45  LOG_DIR = "logs\\fit\\" + TIME
46  CKPT_DIR = SESSION_PATH+ "ckpt\\"
```

Figura 5.35.- Fragmento de código: creación de directorios

En cada entrenamiento, deberán especificarse todos los directorios que se van a emplear durante el programa. Algunos serán para lectura, como ocurre con la base de datos, y otros serán de escritura como los registros de datos. Todas las acciones de escritura que efectuará el programa serán iguales independientemente del tipo de entrenamiento, sin embargo, cada ejecución deberá representar una instancia de datos independiente. Esto quiere decir que un mismo código debe generar instancias de datos diferentes y nunca superponer a otras ya almacenadas.

Para facilitar esta tarea se optó por utilizar una variable “TIME” que representa la fecha exacta en la que se ejecuto el archivo. En la figura 5.35 , se observa la declaración de esta variable, que tomará un valor único en cada ejecución. A partir de esa variable única, se crea un directorio “run+TIME”, dando lugar a una carpeta única para introducir toda la información de ese entrenamiento. Se comprobará la existencia de los directorios para evitar rupturas del programa, creándolos cuando no existan.

En esta carpeta se guardará el modelo final, la recolección de datos en archivos “.logs” y los checkpoints, de la forma ya detallada anteriormente. Además se construirá un documento de texto, de la forma indicada en la figura 5.36.

```
245     #GUARDAMOS CONFIGURACIÓN
246     file = open(SESSION_PATH + "info_model.txt", "w")
247     neuronal_network.summary(print_fn=lambda x: file.write(x + '\n'))
248     file.write("\n\n" + "Learning rate: " + str(LEARNING_RATE) + "\nDecay: " + str(DECAY))
249     file.close()
250
```

Figura 5.36.- Fragmento de código: recopilación información de interés

Gracias a este documento, podrá consultarse qué hiperparámetros y estructuras fueron empleados para ese modelo que, a simple vista, solo aportaba su precisión y número de épocas. También se incluye una figura en forma de gráfica que aporta una vista global de todo entrenamiento. La gráfica permite una vista rápida durante la navegación entre los archivos. No tiene mayor uso dado que los análisis importantes se efectúan con Tensorboard, que es una herramienta completa que supera a cualquier imagen individual que pueda construirse.

Una vista final de cada directorio de entrenamiento podría ser la observada en la figura 5.37.

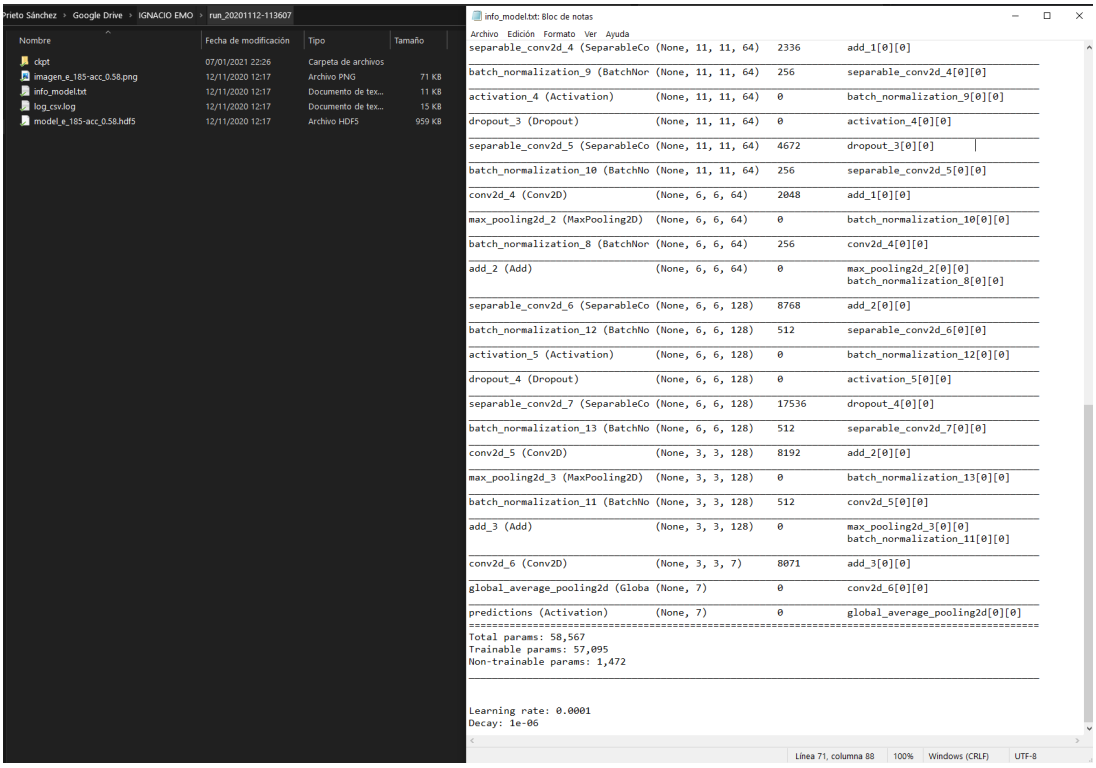


Figura 5.37.- Organización de directorios y vista de la ficha resumen estructural

Cabe explicar que los símbolos verdes en cada archivo corresponden a la copia de seguridad en la nube. Como se efectúan entrenamientos desde diferentes equipos pero utilizando el mismo código, es útil que la carpeta sea compartida. De esta forma, cuando uno de los dos equipos esté en ejecución de un entrenamiento, los datos aportados así como los checkpoints comienzan a sincronizarse y ya pueden visualizarse desde el otro equipo. La creación y uso de modelos mediante un directorio común, es muy eficiente cuando se trabaja con múltiples equipos.

5.5.- Tensorboard

Tensorboard es un conjunto de herramientas de visualización y monitorización de Tensorflow. Como ya se ha adelantado anteriormente, posee un gran potencial para controlar en tiempo real la evolución que experimentan los entrenamientos en ejecución. Pueden seleccionarse las métricas deseadas, ver histogramas y otras características que se modifican a lo largo del tiempo [55].

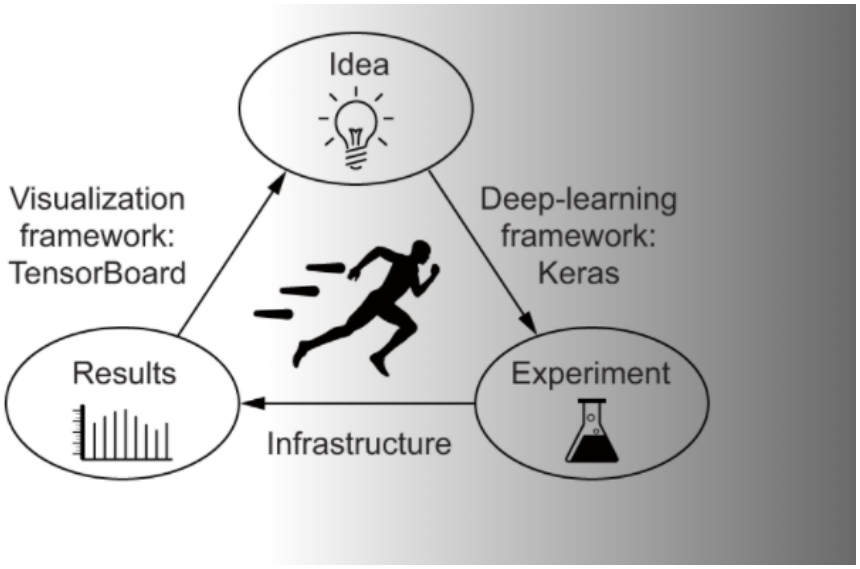


Figura 5.38.- Diagrama de trabajo. Fuente: Web Keras [45]

Como muestra el diagrama de ciclo de trabajo 5.38, la idea representa el peso principal que debe plasmarse al Deep Learning. Tras tener construida la infraestructura, se efectúan los entrenamientos que aportarán la evolución de los modelos y generarán resultados. Estos

se visualizan mediante Tensorboard y nutrirán a las ideas para aumentar la eficiencia. La conclusión de todo esto es la importancia del análisis de resultados ya que alimenta el motor de la mejora.

Primeramente, para poder utilizarlo, ha de declararse su uso en el código. El único parámetro necesario es un directorio donde se guardaran los ".log" propios de esta herramienta. Es de interés comentar que este directorio destinado a Tensorboard se encuentra fuera de cada carpeta individual de entrenamiento. El motivo de esto es unificar todos los datos en el mismo directorio, facilitando las comparativas entre diferentes redes neuronales, independientemente de su fecha de creación o estructura.

Cuando se inicie el entrenamiento, ya se encontrarán disponibles datos para comenzar a estudiar. En el código funciona como un objeto de tipo Callback, es decir, será llamado y actualizado a lo largo del entrenamiento. En su documentación oficial se especifica que para su ejecución debe iniciarse mediante terminal. La información se lanzará en el puerto local 6006 por lo que se accede mediante un navegador. Este proceso se visualiza en la siguiente figura 5.39.

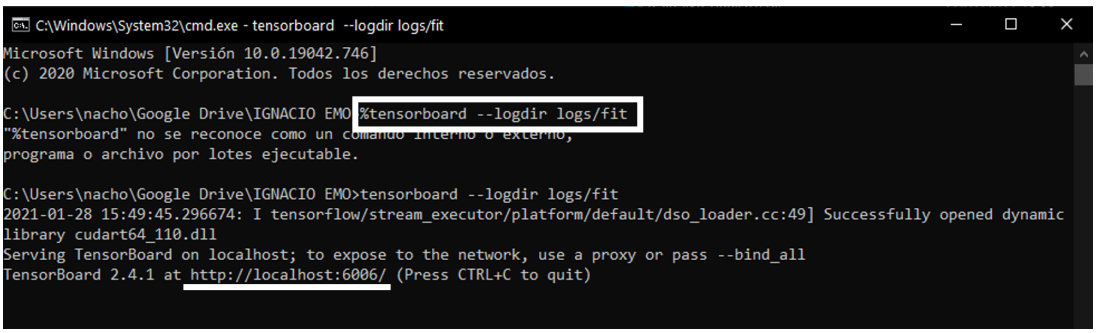


Figura 5.39.- Ejecución Tensorboard en terminal

Para efectuar su lanzamiento de una forma mucho más práctica, mediante el programa, se agrega el código mostrado en la siguiente figura 5.40.

```
198    # Gráficos a timepo real
199    try:
200        # Se define tensorboard para seguir el entrenamiento a tiempo real
201        tensorboard = tf.keras.callbacks.TensorBoard(log_dir=LOG_DIR)
202        # Lanzamos los tensorboard
203        process = subprocess.Popen('tensorboard --logdir .\logs', stdout=subprocess.PIPE, stderr=subprocess.PIPE)
204        # Abrimos el navegador
205        os.system('start chrome http://localhost:6006/')
206    except Exception:
207        print ("No pudo lanzarse tensorboard. Excepción: \n", traceback.print_exc())
208
```

Figura 5.40.- Fragmento de código: ejecución y lanzamiento de Tensorboard

Su inicialización siempre ha de efectuarse con el directorio de los archivos ".log" como se aprecia en la línea 201. Sin embargo, mediante la librería "subprocess" puede lanzarse por código como un subproceso, evitando así tener que ejecutarlo manualmente en una terminal paralela.

Además, también se ejecuta un navegador con la dirección al puerto local 6006, que es donde se visualiza el contenido. Todo este conjunto de instrucciones permiten que, al ejecutar un entrenamiento, de forma automática se inicie una pestaña del navegador y se puedan visualizar ya gráficas deseadas.

Como no todos los dispositivos poseen el mismo navegador o quizás no tengan la herramienta Tensorboard instalada, las instrucciones están ejecutadas sobre un "try" que, en caso de error, permita continuar con la ejecución del programa.

Un ejemplo de visualización al ejecutar el entrenamiento es la figura 5.41, donde se comienzan a observar los datos del modelo para las primeras épocas.

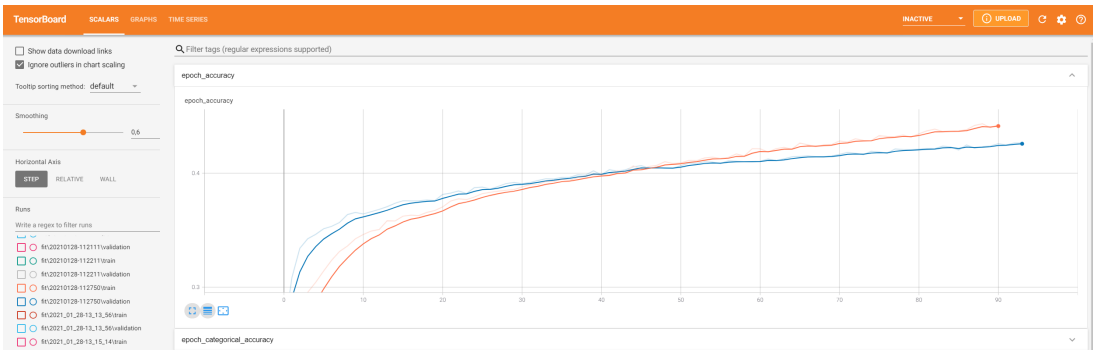


Figura 5.41.- Visualización de Tensorboard en el navegador

Además, en la sección de la izquierda pueden señalarse todos los datos que se deseen representar. De esta forma, se puede comparar, por ejemplo, cómo evoluciona a tiempo real el modelo en entrenamiento respecto a otro que ya se haya completado anteriormente. La figura 5.42, muestra todos los "logs" que se han guardado a lo largo del desarrollo del trabajo para efectuar comparativas. De un vistazo puede observarse una gran cantidad de información. Todas las funcionalidades que se pueden explotar, en vez de explicarlas, serán empleadas en la etapa de análisis de resultados.

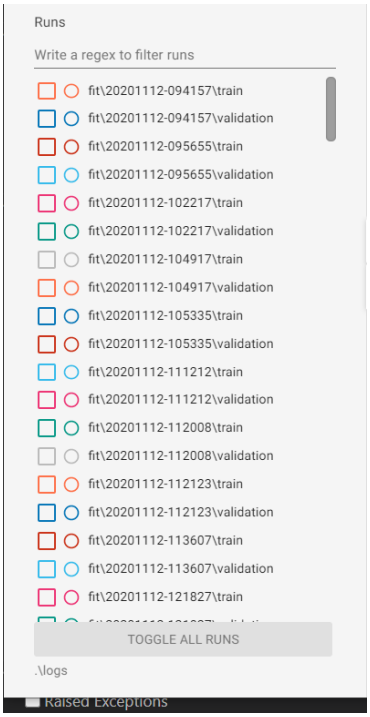


Figura 5.42.- Registro de logs generados por todos los entrenamientos

5.6.- Aceleradores de IA: GPU

Cuando se trata de ejecutar entrenamientos de redes neuronales, el hardware utilizado para ello tiene una gran importancia. Modificar características de la red neuronal, así como añadir capas, incrementa notoriamente la carga computacional.

Durante el entrenamiento de una red neuronal se producen dos flujos de datos. El primero va desde el principio de la red hasta el final para obtener el resultado. El segundo, ya

mencionado anteriormente como “backpropagation”, va desde el final de la red hasta el principio, actualizando los pesos en función del error.

Los computadores son las grandes herramientas de nuestra sociedad, les confiamos todas las tareas y conocemos su gran potencia. Sin embargo, desde el nacimiento del Deep Learning, los procesadores comenzaron a verse débiles ante estos cálculos. Se concluía que estos entrenamientos eran muy largos, pesados y “profundos”. Ahora se plantea la gran cuestión: ¿Dónde está el problema?.

Para conocer qué le supone al ordenador efectuar un entrenamiento, basta con volver atrás y pensar cuál era el formato de los datos. Un filtro convolucional puede visualizarse como una matriz (3x3) que irá recorriendo toda una imagen pasando por encima de cada píxel. Multiplicará la matriz filtro por la pequeña matriz que formen 9 píxeles de la imagen.

Cuando un filtro recorre una imagen completa, genera una nueva imagen. A esta nueva imagen se le volverá a pasar un filtro y así sucesivamente. ¿Cuál es la conclusión de todo esto?. Muchas operaciones con matrices de grandes dimensiones, el peor enemigo de una CPU.

Las CPU’s son muy eficaces en operaciones de escalares y ofrecen procesamiento de propósito general. Sin entrar demasiado en la temática hardware, la naturaleza de la CPU es realizar operaciones muy rápido, pero de forma secuencial. Con los avances en este campo, las CPU ya incluyen varios núcleos (en torno a 10) [56], pero su metodología es la misma.

Las GPU’s están creadas con propósitos gráficos, por lo que presentan una naturaleza diferente. Su objetivo es realizar el mayor número de operaciones simultáneamente, por eso el número de núcleos que integran, es mucho mayor.

Los cálculos matriciales en GPU se operan dividiendo las operaciones y calculándolas de forma independiente. Esto supone que el cálculo matricial de muchas dimensiones sea sencillo para la tarjeta gráfica. En la CPU, se realizan de forma secuencial, incluso aunque utilice sus 10 núcleos para dividir la tarea, para grandes magnitudes sigue siendo un proceso secuencial. La siguiente figura 5.43, ilustra el número de núcleos de cada uno.

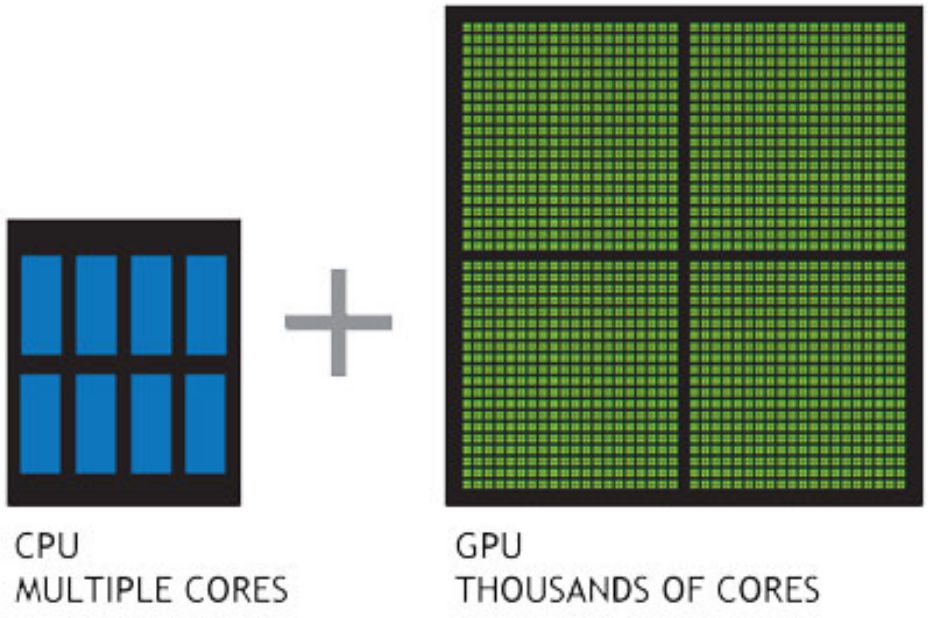


Figura 5.43.- Comparativa de núcleos entre CPU y GPU. Fuente: NVIDIA [57]

La conclusión es que la CPU comienza a ejecutar las operaciones de forma secuencial lo que resulta ridículamente lento en comparación a ejecutarlas en paralelo como hace la GPU. Cuantas más capas y profundidad tenga la red neuronal, más operaciones deberán ser ejecutadas y la lista de espera de la CPU será interminable.

Emplear esos núcleos procedentes de una tarjeta gráfica para otras tareas es una tecnología donde NVIDIA lleva trabajando muchos años. Su herramienta denominada CUDA, ha cambiado el mundo del Deep Learning y se abre camino en el denominado GPGPU (computación de propósito general en unidades de procesamiento gráfico). Es decir, aprovechar el número de núcleos de una GPU no solo para generar imágenes sino también para renderizar vídeos, modelar elementos tridimensionales y mucho más.

5.6.1.- CUDA

CUDA es la herramienta que permite emplear los núcleos de la tarjeta gráfica para efectuar operaciones en paralelo. Es compatible con Python y Tensorflow permitiendo acelerar drásticamente el aprendizaje. Durante el entrenamiento, gracias a su gran ancho de banda

de memoria y el paralelismo, se reducirá el tiempo que tarda el programa en completar una época [58].

Para poder funcionar, CUDA necesita que la biblioteca cuDNN esté instalada en el equipo. Python, Tensorflow, CUDA y cuDNN confieren un ecosistema estricto en cuanto a versiones se refiere. Las incompatibilidades pueden ser algo problemático ya que la actualización de una de las herramientas puede imposibilitar el uso de otra. Para el caso de la tarjeta gráfica NVIDIA 3070, las versiones se modificaron por completo respecto a la otra empleada, la 2070.

La generación 20, trabajaba con Python 3.7, Tensorflow 2.1, CUDA 10.0 y cuDDN 7.0. La generación 30 presenta una arquitectura Ampere lo que obliga estrictamente el uso de CUDA 11.1 y cuDNN 8.0. Esto influye a la versión de Python que estrictamente ha de ser la 3.8 y, por tanto Tensorflow, 2.5.

En el manual de usuario se facilitará ayuda para instalar estos complementos si desea utilizarlos. En cuanto a eficiencia se refiere, en vez de emplear algunas gráficas de otros experimentos, se han efectuado pruebas para este caso específico. Las siguientes imágenes muestran marcado en color verde, el tiempo que tarda en completarse una época en cada caso. Para cada ordenador, se han efectuado dos pruebas, en un caso el entrenamiento se efectúa a partir de la CPU y en el otro, mediante la GPU (empleando las herramientas descritas). Para obtener información de forma visual, se han adjuntado a cada imagen las características de los componentes así como el rendimiento que experimenta cada entrenamiento.

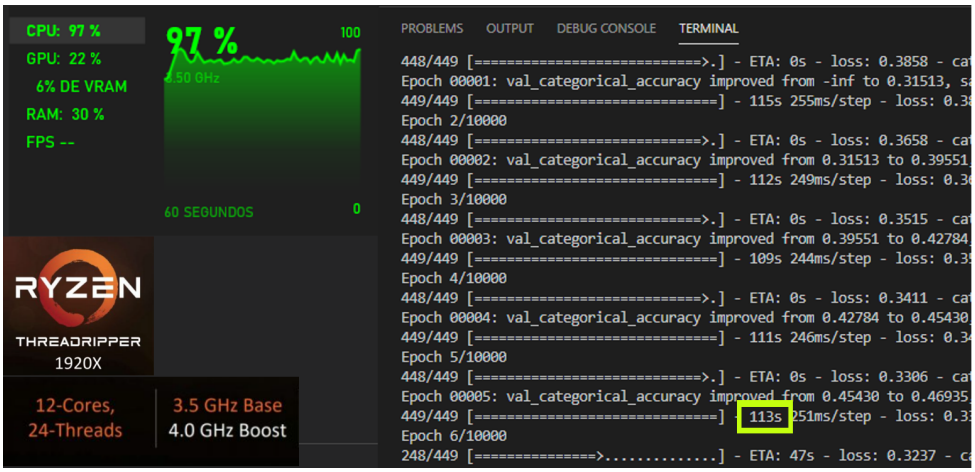


Figura 5.44.- Entrenamiento mediante CPU: Ryzen

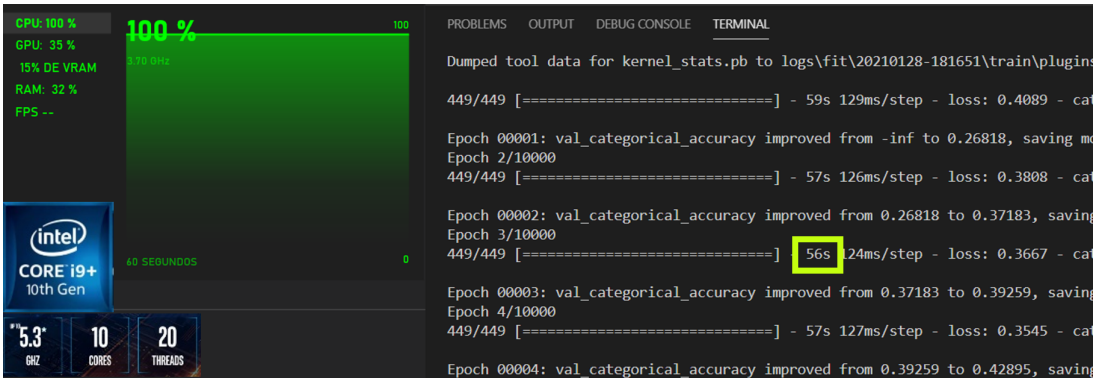


Figura 5.45.- Entrenamiento mediante CPU: Intel i9

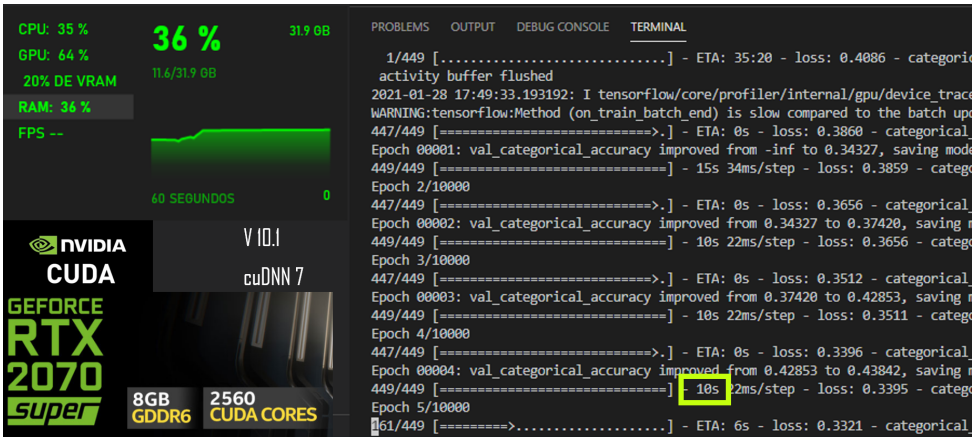


Figura 5.46.- Entrenamiento mediante GPU: NVIDIA 2070 super

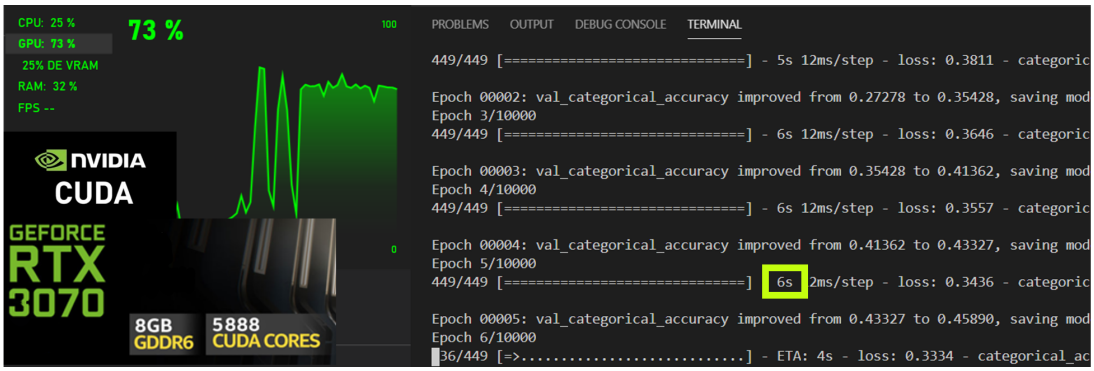


Figura 5.47.- Entrenamiento mediante GPU: NVIDIA 3070 RTX

Además, en las imágenes 5.46 y 5.47, se muestran los núcleos de procesamiento CUDA que incluye cada tarjeta. Teniendo en cuenta que, para su estudio, muchos de los modelos fueron entrenados 10.000 épocas, el tiempo de entrenamiento en CPU hubiera supuesto algo más de 13 días, mientras que en GPU son en torno a 16 horas y media. Los resultantes son arrolladores, las GPU's son imprescindibles para trabajar con redes neuronales convolucionales y evolucionar en el Deep Learning.

Entre la generación 20 y la 30, se observa una reducción aproximada de 4s en cada época. Aparentemente parece poco, pero en el balance total supone unas 11 horas y 7 minutos.

5.7.- Fenómenos del entrenamiento

De los datos que genera la red neuronal durante el entrenamiento, pueden identificarse diferentes fenómenos que nos anticipan las características que tendrá el modelo. Interpretar las métricas del modelo es fundamental para su mejora, por ello, se hizo tanto énfasis en la importancia de la herramienta Tensorboard.

A pesar de tener clara la estructura que se pretende utilizar para la red neuronal, la cantidad de modificaciones posibles es inmensa. Ya fueron detallados gran cantidad de hiperparámetros y factores que pueden causar gran impacto en el modelo. El trabajo del ingeniero de datos, recae en saber interpretar lo que sucede en cada modelo y encontrar las modificaciones pertinentes para mejorarlo. Cada caso es muy característico y debe ser estudiado con detenimiento, ya que puede no ser tan simple como las categorías

describen. Estos análisis de efectuarán en el próximo capítulo donde se evaluarán los resultados obtenidos. De antemano, la ciencia de datos que trabaja los aprendizajes de redes neuronales, advierte de las tres situaciones posibles. Cabe mencionar que todos los modelos pasan por las tres etapas a lo largo de su proceso de aprendizaje, al menos por los dos primeros casos.

5.7.1.- Underfitting

El primer caso se denomina bajo entrenamiento, o lo que es lo mismo, falta de ajuste. Esta situación describe un modelo incapaz de adquirir conocimiento e información del conjunto de datos. Este problema no es habitual ya que describe un modelo completamente ineficiente frente a los datos de entrenamiento y los datos de pruebas. En la mayor parte de estas circunstancias, la estructura , el tipo de red neuronal o el conjunto de datos han de tener graves incompatibilidades. Es decir, no se están utilizando los métodos adecuados para tratar el problema o los datos son poco coherentes [59].

Una base de datos muy dispersa y con poca lógica podría generar underfitting. Pese a ser inteligente, la aleatoriedad de los datos impide al modelo encontrar o entender algo. La siguiente figura 5.48, representa las métricas que mostraría un modelo de inteligencia artificial que sufra underfitting.

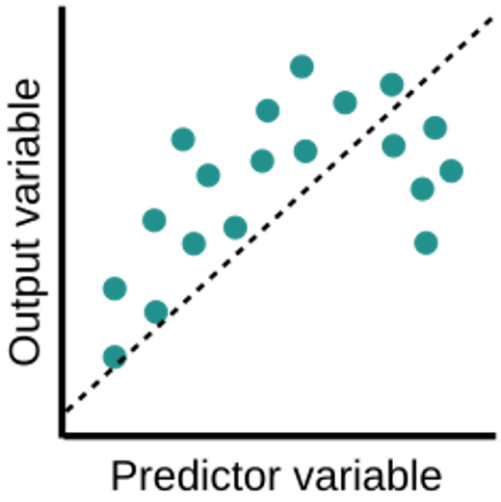


Figura 5.48.- Problema del modelo, underfitting. Fuente: [60]

Es complicado representar un modelo que de forma íntegra sufra underfitting. Su representación se efectúa como una serie de puntos que describen una función curva y se pretenden modelar como una recta. Habitualmente durante un proceso de entrenamiento normal, se señala un fragmento de este como underfitting.

Durante los entrenamientos para catalogar estos problemas se monitoriza la función de pérdidas. Al iniciar un entrenamiento, esta función de pérdidas o error comienza a reducirse. Este decrecimiento simboliza aprendizaje hasta el punto donde se minimiza y el modelo alcanza su potencial. Es por ello que puede verse a menudo señalado como underfitting este fragmento previo a lograr la minimización. En realidad, ese fragmento de underfitting es natural y no simboliza un problema. Se dice que un modelo sufre de underfitting cuando este efecto no es un estado transitorio, sino constante, no logrando minimizar el error pese al paso del tiempo.

5.7.2.- Overfitting

Este efecto es el opuesto a la falta de ajuste explicada anteriormente. Se produce cuando un modelo cae en la particularización y comienza a ajustarse demasiado al conjunto de datos. Este es un problema muy habitual cuando se entrenan redes neuronales ya que es una tendencia relativamente esperada. Existen muchas causas que desembocan en este problema, desde bases de datos desbalanceadas o poco generalizadas a malas estructuras para el modelo. Todo conjunto de datos posee un patrón común que permite extenderse a otros ejemplos, hallarlo es el propósito del entrenamiento. Del mismo modo, en todas las bases de datos existen patrones particulares que permiten categorizar cada entrada y serán pésimos para resolver otros problemas generales.

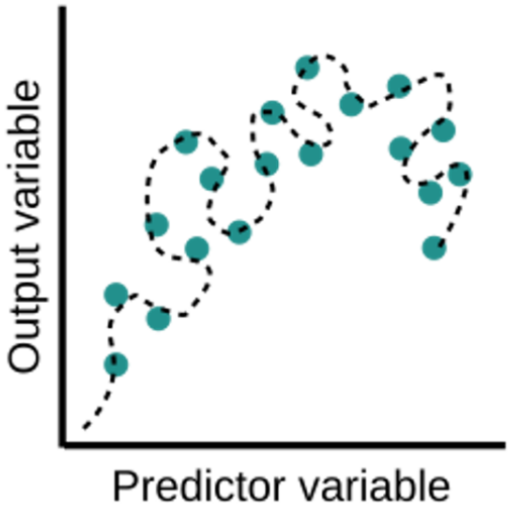


Figura 5.49.- Problema del modelo, overfitting. Fuente: [60]

En la figura 5.49, se observa como el modelo se ajusta perfectamente a todos los puntos. De esta forma, ofrece excelentes resultados al predecir un punto conocido, pero fracasará ante los puntos nuevos. Al contrario del efecto que ocurría al inicio de los entrenamientos, este efecto tiende a producirse tras varias épocas. Cuando el modelo está esforzándose demasiado en aprender, comienza a adoptar tendencias particulares para su conjunto de datos.

El sobre ajuste es un problema muy presente en la mayor parte de los modelos. Es el conflicto principal de los ingenieros de datos, cerciorarse de que su modelo sea una buena generalización para los problemas.

5.7.3.- Correct-fit

El correcto entrenamiento se ubica en un término medio entre la excesiva generalización y la exagerada particularización. Un modelo será bueno cuando ofrezca buenos resultados ante las imágenes de entrenamiento y las de pruebas. En la siguiente figura 5.50, se observa como modelar los ejemplos antes mostrados.

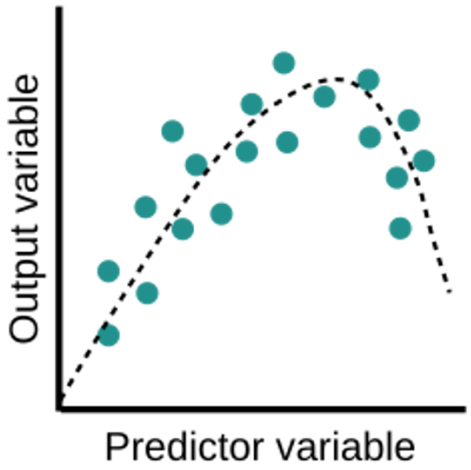


Figura 5.50.- Solución al modelo, entrenamiento correcto. Fuente: [60]

5.7.4.- Encontrar el punto perfecto

Tras haber sido mencionados los problemas que ocurren durante el entrenamiento y el punto que se desea alcanzar, más importancia recae en el análisis de los datos. Típicamente se controlan dos métricas para analizar un entrenamiento, "epoch_loss" y "accuracy". Como ya fue explicado anteriormente, la herramienta que emplea una red neuronal para mejorar es el error. Mediante los datos de entrenamiento evalúa una entrada de la base de datos y la compara con el resultado esperado. Tras terminar cada época, el optimizador aporta un valor de la función de pérdidas y se propone reducirlo para la siguiente época mediante la actualización de los parámetros internos. Si tras cada época se minimiza el error, se alcanzará el mínimo para la función de pérdidas y el modelo estará optimizado. Si tras encontrar dicho mínimo, el modelo sigue iterando sobre los datos, la teoría dice que producirá un sobre ajuste.

Durante el análisis de las gráficas se crearán dos diferentes. Una representará la precisión del modelo y la otra, las pérdidas que este presenta, ambas en función del número de épocas. Las gráficas se pueden visualizar a tiempo real con Tensorboard.

Además, ambas gráficas se generarán para los datos de entrenamiento y para los datos de validación. Es decir, el modelo calcula el error y la precisión que tiene ante el conjunto de datos de entrenamiento y también para el conjunto de pruebas.

Esto aporta una gran cantidad de información, pues pueden observarse los comportamientos que muestran a lo largo de las épocas. Podrá estudiarse si presentan underfitting, overfitting o, por el contrario, se ha logrado un punto de eficiencia y puede finalizar el entrenamiento.

Es importante distinguir qué significan exactamente las métricas empleadas. El caso particular de este proyecto, al tratarse de un problema multi clase, tiene que asumirse como categorical. Por tanto la función de pérdidas empleada es ”crossentropy_loss” y lo mismo ocurre con la precisión “accuracy”, que se ajusta automáticamente a la función de pérdida. En este caso será “cateogrical_accuracy”. Aparentemente parece que la precisión guarda una relación inversamente proporcional al error, pero no es tan simple como parece.

El hecho de que la función de pérdida sea cruzada y de tipo categorical implica efectos que en muchos trabajos y proyectos pasan desapercibidos. La precisión es bastante sencilla, solo debe dividir el número de aciertos entre el número de intentos. Sin embargo, la función de pérdida es algo más compleja. Al emplearse en la última capa la función Softmax, el resultado se proporcionará como una probabilidad. Cada una de las 7 neuronas finales, expresarán el resultado como la probabilidad asociada a su categoría para ser la correcta. Como se ha empleado la función de error de entropía cruzada, significa que el resultado ideal que se espera es que sin ninguna duda, el modelo declare una categoría como solución con una probabilidad del 100 %.

Esto implica que el error no solo depende de seleccionar la etiqueta correcta, sino también de lo seguro que se estaba para cada respuesta. Si una respuesta es exitosa con un 50 %, frente a un 40 % de la segunda categoría, se está obteniendo un error muy alto en esa segunda categoría. El error cometido es alto para la entropía cruzada pues se esperaba un 0 % para esa categoría, ya que no era la etiqueta correcta. El error es genérico para el modelo y no individual.

Es posible encontrarse con un modelo que aumente su probabilidad de acierto (accuracy) mientras que la función de error aumenta en lugar de disminuir, como se esperaba. Si la función de error aumenta, un análisis simple dirá que se trata de overfitting, sin embargo, se esconcen resultados más complejos tras este fenómeno. Este efecto se detallará más a fondo en la etapa de análisis de resultados.

5.8.- Predicción de emociones en la RaspBerry

El programa para clasificar las emociones se ejecutará en la Raspberry. Para facilitar el uso en el PC, también se ha adaptado una versión para ejecutarse en ordenador. El script diseñado para la Raspberry posee características particulares destinadas a la eficiencia en esta placa. Para que pueda ejecutarse, debe estar conectada a la placa una Webcam que proporcione las imágenes en tiempo real. Toda la información de ejecución será mostrada por un panel de LEDs que ha sido ensamblado en una estructura robusta. Gracias al guardado de los modelos como archivos exportables, pueden trasladarse a la placa cuando se finalicen los entrenamientos. La Raspberry, la Webcam y el panel conforman una estructura autosuficiente para ejecutar las predicciones con los potentes modelos entrenados por otras máquinas.

Junto con el programa de inferencia capaz de identificar las emociones, se ha diseñado una biblioteca de funciones denominada Pixel. El programa de inferencia actúa en tiempo real analizando las imágenes captadas y ofreciendo resultados. Estos resultados son traducidos y representados en el panel gracias a esta librería personalizada. El panel LED posee una dimensión de 8x32 píxeles, es flexible y tiene tres conexiones para su control. La figura 5.51 representa este componente antes de su reconstrucción.

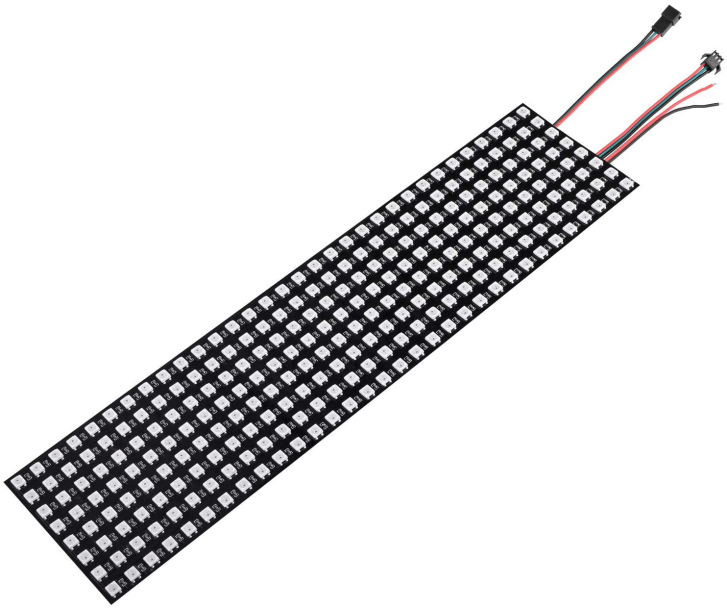


Figura 5.51.- Panel LED 8x32 píxeles

Para el control del panel LED se emplea la API neopixel. Esta librería permite incorporar el panel en el código como un objeto de la clase neopixel. Por parámetro, se indica en la instancia el número de LEDs y el pin de la placa que se desea utilizar para controlarlo. También debe indicarse con "auto_write" si se desean aplicar los cambios en los LED en el instante de su modificación, o por el contrario, mediante una función de actualización.

En la figura 5.52 se muestra el mapa de pines de la placa. Los señalados en el lado izquierdo se utilizarán para el ventilador que refrigera la placa. Corresponden a una alimentación de 3.3V y la toma tierra. Los que se indican en el lado derecho, se utilizarán para el panel. Los marcados en naranja y gris, corresponden a 5V de alimentación y a la toma tierra, respectivamente.

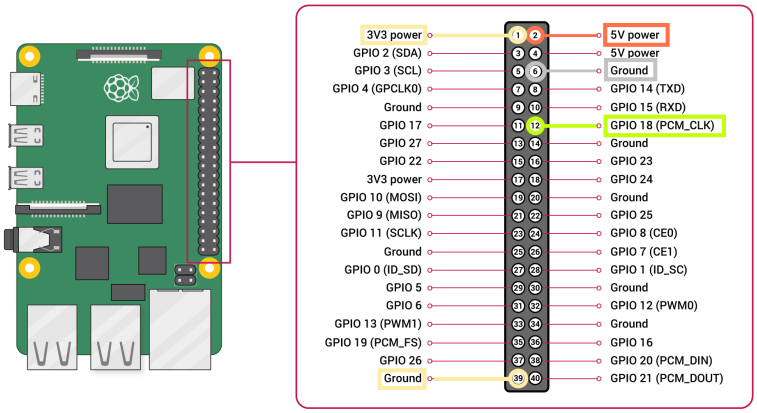


Figura 5.52.- Raspberry Pi mapa de pines. Fuente: documentación Raspberry [61]

El pin indicado con el color verde corresponderá al GPIO 18, que controlará la información que proyectan los LEDs. Este pin se indica en el parámetro de neopixel como ya se mencionó. La instancia del panel en el código se efectúa como se observa en la figura 5.53.

```
1  import pixel
2  import board
3  import neopixel
4  import keyboard
5
6  # Conexión panel leds
7  panel = neopixel.NeoPixel(board.D18, 256, auto_write=False)
```

Figura 5.53.- Creación del objeto neopixel para controlar el panel

Lo cierto es que la librería neopixel es bastante simple. Actúa como backend, proporcionando la posibilidad de indicar el color de cada píxel en un formato práctico. A partir de esta abstracción, todas las funciones se han creado en el archivo píxel, que actúa como una librería de métodos para utilizar en el programa de inferencia. El control de estas placas no está muy documentado, por lo que existen múltiples funciones elaboradas por los usuarios con el propósito de controlarlas. Para cada panel, las funciones son diferentes ya que la dirección de cada LED es diferente. Esto implica que una función para proyectar texto, pierda toda la lógica cuando se utiliza en un panel de dimensiones diferentes.

El nombre de “PIXEL” fue escogido para representar este conjunto de funciones. Los métodos de PIXEL, conforman un conjunto de herramientas particularizadas en su totalidad para este panel. Para su construcción, se utilizó como base un conjunto de funciones elaboradas por otros usuarios para otros paneles. Proporcionaban la posibilidad de escribir letras, construir texto estático y mostrarlo en desplazamiento. Todas estas funciones han tenido que ser adaptadas al panel particular. Al carecer de una librería oficial propia, los usuarios han de configurar de forma íntegra el control de su panel. Todas las letras se han adaptado a mano, como se visualiza en la figura 5.54, para poder ser representadas con una combinación de LEDs.

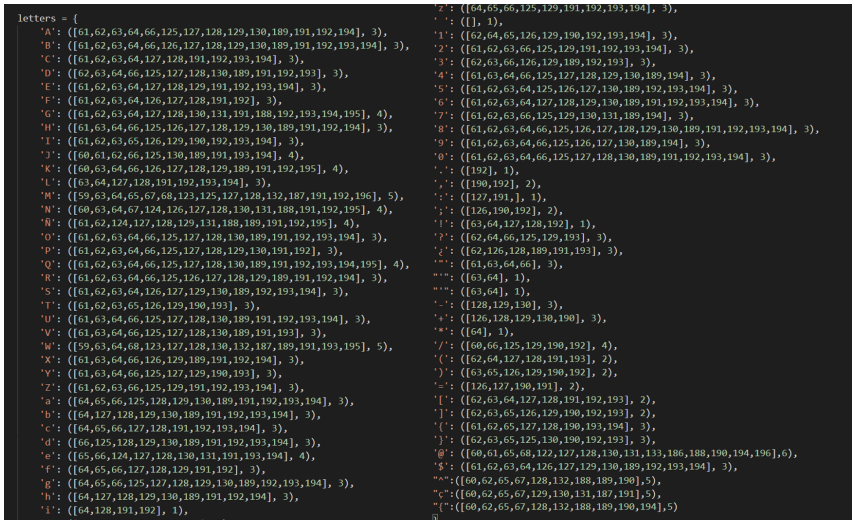


Figura 5.54.- Funciones PIXEL: representación de caracteres

Las funciones que forman el conjunto de PIXEL son:

- ***draw_letter(letter, pos, np, color):*** esta función toma como parámetros un carácter, una posición, el panel neopixel y el color. Con ella podrá escribirse el carácter en la posición deseada del panel. El color de los LEDs que se han de encender corresponderá al indicado por parámetro.
- ***clean(letter, pos, np):*** permite borrar un carácter de un posición concreta del panel. Es útil para la función clock, ya que permite modificar el valor de un único dígito sin necesidad de actualizar la hora completa.

- ***scroll_text(text, np, speed, color):*** reproduce la cadena de texto indicada por parámetro en el panel. El texto se desplazará por el panel a la velocidad indicada por el parámetro speed. El color también se selecciona por parámetro.
- ***accuracy_clean(np):*** permite limpiar la linea inferior del panel. Esta línea es la que se emplea para mostrar la probabilidad de éxito de la respuesta dada por el modelo. En otras palabras, la confianza de cada respuesta.
- ***accuracy_print(acc, np):*** permite escribir la confianza de cada inferencia en el formato de línea inferior. Escalará el % indicado a los 32 píxeles que posee la línea y los rellenará en función de la respuesta. Si la confianza es baja, la barra se llenará con pocos píxeles y con un color oscuro. Si la confianza de la respuesta es alta, la barra estará casi al completo y en color verde.
- ***write_text(text, np, color,border):*** permite proyectar en el panel un texto estático. Es útil para escribir las emociones con su color.
- ***digits(a):*** función que emplea el reloj para extraer dígitos de una cifra.
- ***zero(np,color):*** esta función permite establecer a 0 todos los dígitos del reloj.
- ***clock(np,color):*** con esta función puede ejecutarse un reloj en el panel en el color deseado. Es interesante para darle otras utilidades secundarias al panel.

En el programa de predicción de emociones se definen dos diccionarios. El primero asocia a cada identificador numérico la emoción que le corresponde. El segundo, asocia a cada una de las emociones representadas, un color diferente.

```
27  # Activamos un hilo para no detener la ejecución
28  _thread.start_new_thread(pixel.scroll_text, ("Bienvenido", panel, 10, color))
29
30  # Archivo de modelo a utilizar
31  model='./modelos/modelo_CNN.hdf5'
32  CNN_model = tf.keras.models.load_model(model)
33
34
35  # Diccionario con las emociones
36  emotion_dict = {0: "Enfadado", 1: "Asco", 2: "Asustado", 3: "Feliz", 4: "Triste", 5: "Sorpresa", 6: "Neutral"}
37  color_dict = {0: (255, 0, 0), 1: (129, 3, 203), 2: (255, 20, 20), 3: (87,255, 0), 4: (69, 59, 244), 5: (244, 59, 184), 6: (59, 244, 199)}
```

Figura 5.55.- Programa de predicción de emociones, creación de diccionarios

En la figura 5.55, se observa la incorporación del modelo al código y la creación de los dos diccionarios. También puede observarse que se proyectan mensajes en el panel con propósito informativo para el usuario. Se emplea un hilo de ejecución para la función

de texto “scroll” ya que así, puede cargarse el modelo mientras se está visualizando el mensaje.

```
41  cap = cv2.VideoCapture(0)
42  while(state):
43      ret, frame = cap.read()
44
45      # Se notifica el comienzo de la lectura hasta captar el primer rostro
46      if(emotion == ""):
47          pixel.write_text('leyendo', panel, color, 0)
48
49      # Si no hay imagen, se termina el bucle
50      if not ret:
51          pixel.write_text('Error', panel, color, 0)
52          time.sleep(2)
53          break
54      haarcascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
55      gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
56      faces = haarcascade.detectMultiScale(gray, 1.3, 5)
57
58      if keyboard.is_pressed('ç'):
59          state=False
60
```

Figura 5.56.- Programa de predicción de emociones, inicio bucle de OpenCV

En el código del programa de predicción, representado en la figura 5.56, se utiliza la API OpenCV. Gracias a ella, podrán detectarse las caras aplicando “Haar Cascades”. Existen diferentes archivos de estos clasificadores. Su funcionamiento se basa en filtros kernel para detectar objetos, en este caso, serán rostros. Mediante la WebCam, se capturarán imágenes donde se intentarán localizar caras.

```
61      for (x, y, w, h) in faces:
62          roi_gray = gray[y:y + h, x:x + w]
63          # Adaptacion dimensión
64          image = cv2.resize(roi_gray, (48, 48))
65          np_image_array = image.astype('float32') / 255.0
66          img_ready = np.expand_dims(np.expand_dims(np_image_array, -1), 0)
67
68          # Inferencia
69          result = CNN_model.predict(img_ready)
70          tiempo_final = time()
71
72          emotion_number = int(np.argmax(result))
73          if(np.max(result) > ACCURACY):
74              # Actualización panel led
75              if(emotion_dict[emotion_number] != emotion):
76                  pixel.write_text(emotion_dict[emotion_number], panel, color_dict[emotion_number], 0)
77                  emotion = emotion_dict[emotion_number]
78          else:
79              if(emotion_dict[emotion_number] != emotion):
80                  pixel.write_text("       ¿?", panel, (255,120,84), 0)
81                  emotion = "       ¿?"
82                  pixel.accuracy_clean(panel)
83          if(ACC_ACTIVATED):
84              pixel.accuracy_print(np.max(result), panel)
```

Figura 5.57.- Programa de predicción de emociones, inferencia en las imágenes

Como se observa en la figura 5.57, tras identificar una cara, se efectúa el proceso de inferencia y se actualiza el panel. El bucle de procesamiento de las imágenes solo finalizará si el usuario pulsa la tecla “ç”.

5.9.- Ensamblaje y resultados del panel

Para otorgarle mayor robustez al dispositivo, se construyó una estructura de madera incorporando el panel LED en el interior. Con una tela tensada, se cubrió la parte trasera dejando un pequeño hueco para los cables que deben conectarse a la placa. En la parte frontal se agregó una lámina plástica para proteger el panel. Además, se añadieron dos capas de tinte oscuro, de esta forma se mejora mucho el aspecto de los LEDs.

En fotografías, los LEDs tienen un aspecto bastante diferente a cómo se aprecian en la realidad. El brillo que generan, dificulta fotografiar el panel ya que difumina las letras proyectadas. Las siguientes figuras muestran el panel ensamblado así como distintos ejemplos que se visualizan durante la ejecución del programa.

Figura 5.58.- Panel LED para mostrar las emociones

Figura 5.59.- Inicio del programa

Figura 5.60.- Lectura de rostros

Las figuras 5.59 y 5.60, muestran los mensajes de inicio que se muestran al ejecutar el programa. Durante el proceso de importación de las librerías, se mostrará el mensaje "cargando". Cuando aparezca "leyendo", el proceso anterior habrá terminado y comenzará la lectura de imágenes. Las siguientes imágenes muestran las posibles emociones que puede interpretar el programa.

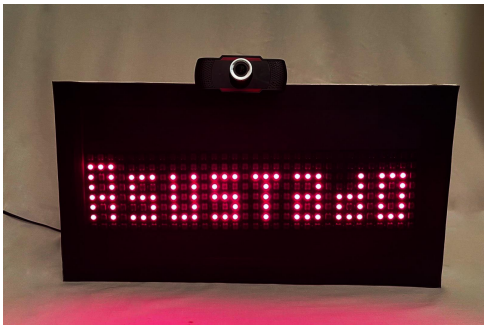


Figura 5.61.- Emoción asustado

Figura 5.62.- Emoción felicidad

Figura 5.63.- Emoción asco

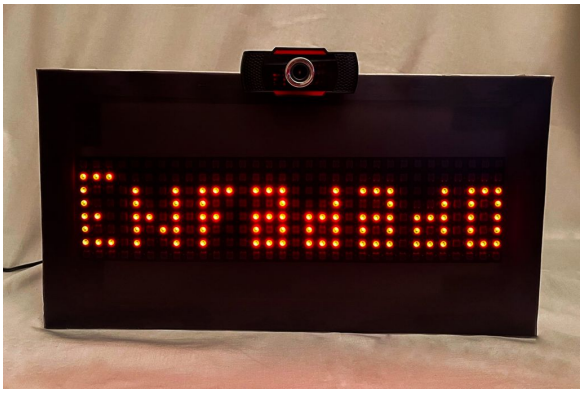


Figura 5.64.- Emoción enfado

Figura 5.65.- Emoción neutral

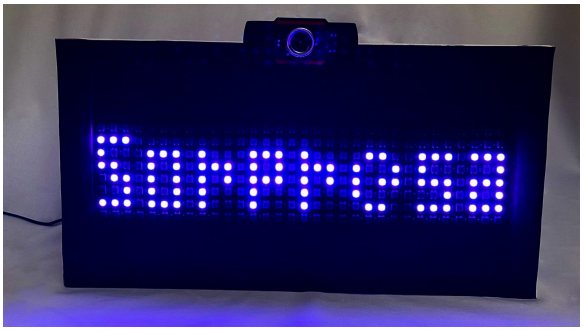


Figura 5.66.- Emoción sorpresa

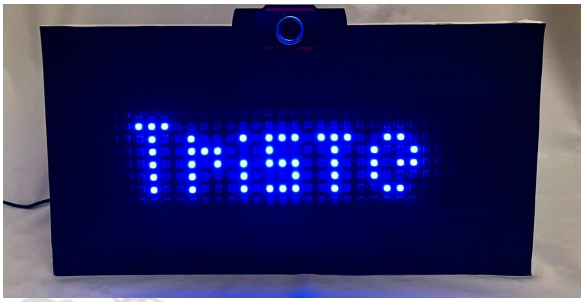


Figura 5.67.- Emoción tristeza

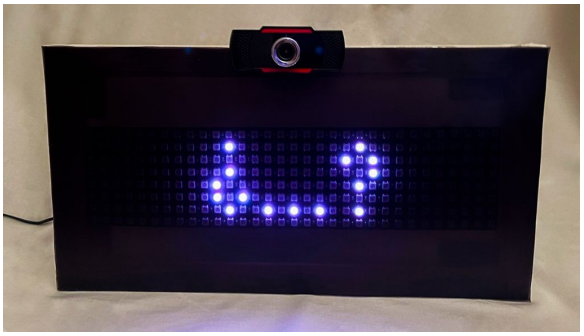


Figura 5.68.- Resultado confuso

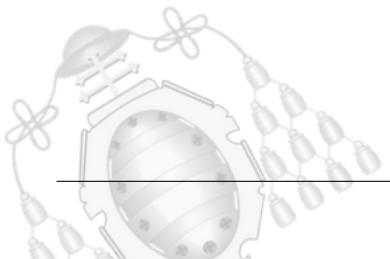
6. Análisis de resultados

En este capítulo se detallarán los diversos modelos que fueron construidos empleando siempre la misma base de datos. Se documentarán los resultados con esquematizaciones de las redes, gráficas que muestran su evolución y las conclusiones relacionadas al éxito o fracaso de cada versión.

Tras evaluar varias estructuras, se aportará una visión global de los resultados, así como explicación a las tendencias generales observadas. Se incluirán además en las conclusiones finales, las limitaciones que se han detectado en el proyecto. Las mejoras para superar estas trabas, se detallarán en el capítulo de trabajo futuro junto a posibles variaciones que otorgarían al proyecto gran potencial.

El análisis de las gráficas será empleado para cada una de las versiones. Analizar el comportamiento de la red en tiempo real es una de las técnicas más útiles y fascinantes del Deep Learning. La herramienta de visualización permite muchas funciones que no pueden ser capturas. Con el propósito de sintetizar cada estudio, se aportarán capturas de los fragmentos gráficos de mayor interés. Además de estudiar los modelos, se pretende que el lector comprenda los efectos que están ocurriendo y sea capaz de interpretar los resultados. Para facilitar la comprensión, las primeras valoraciones corresponden a los peores resultados, que presentarán detalles claros y alarmantes. Estas versiones presentan tendencias muy notorias mientras que, al avanzar en búsqueda de mayor eficiencia, las tendencias son más difusas.

Mejorar los modelos se convierte en una tarea más compleja cuando se rozan las limitaciones del proyecto. Dado el objetivo y el conjunto de datos, la eficiencia y precisión tienen un límite. El objetivo será aproximarse lo máximo posible a él.



6.1.- Modelos VGG

Las primeras versiones desarrolladas para el proyecto se basan en emplear modelos VGG pre entrenados. En términos de Deep Learning, se trata de redes neuronales convolucionales muy profundas. Presentan un conocimiento muy genérico para detectar patrones en imágenes tales como bordes, figuras, formas... Han sido entrenadas con una gran variedad de imágenes de ámbito general. Habitualmente se emplean en proyectos para objetos, personas o animales. Todos los parámetros de estas capas no se entrenan, por lo que los pesos son siempre los mismos.

6.1.1.- VGG16 y Red Neuronal 3 capas

El modelo VGG16 posee 16 capas, donde las 3 últimas representan una red neuronal simple [62]. La siguiente figura 6.2, representa la macroarquitectura de este modelo.

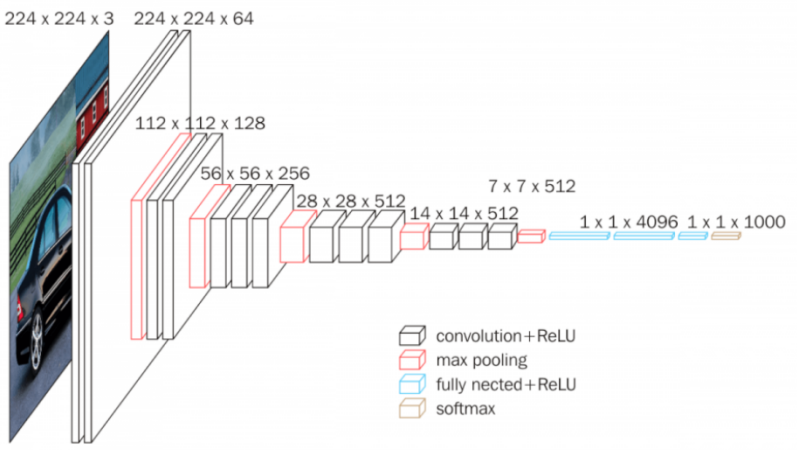


Figura 6.1.- Estructura del modelo VGG16. Fuente: Toward data science [63]

Para el modelo de detección de emociones se planteó eliminar las tres últimas capas que corresponden a la red neuronal simple. De esta forma, se agregó una red neuronal diseñada de forma particular para poder entrenarla.

En el salto a la red neuronal, los datos presentan un formato de 512 puntos, que corresponde a un aplanado del output de las capas convolucionales. Este efecto de transformación de la imagen a una dimensión menor se visualiza en la figura 6.2.

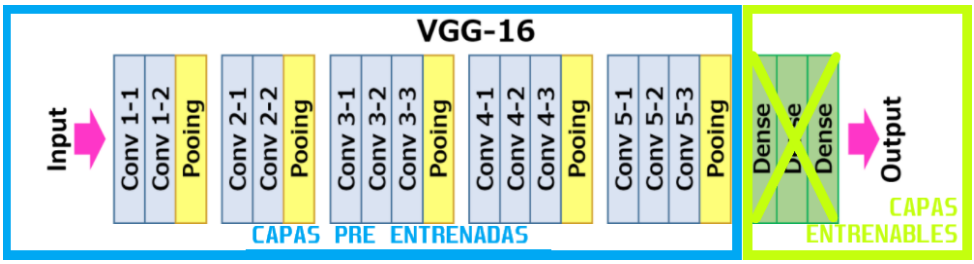


Figura 6.2.- Capas VGG16. Fuente: Toward data science [63]

Las capas entrenables ocupan la posición enmarcada en verde en la figura 6.2. Se diseñaron 3 capas de neuronas dense con 256, 128 y 64 neuras en cada capa respectivamente. Existe una última capa más, donde se encuentran 7 neuronas con función SoftMax que representan el output. La siguiente figura 6.3 representa el diagrama estructural que proporciona Tensorboard de la red.

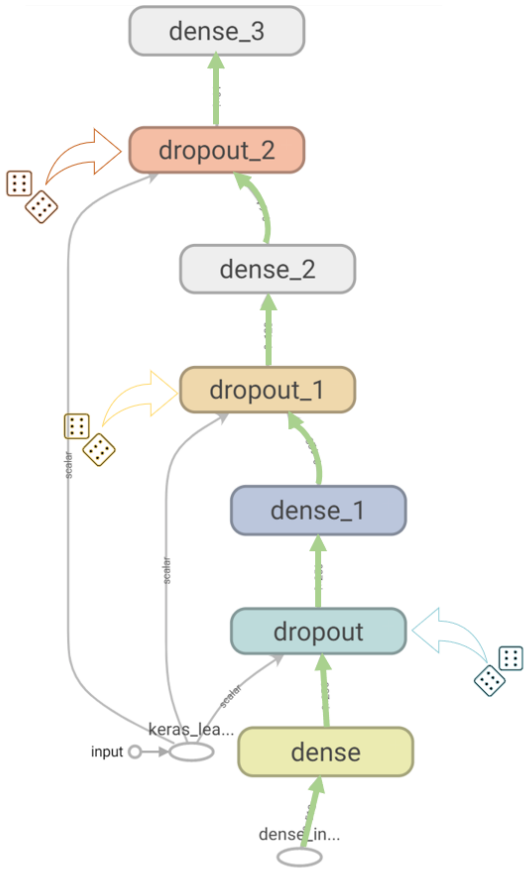


Figura 6.3.- Estructura secuencial de las capas entrenables

En el esquema se señala mediante flechas verdes el ciclo de los datos a los largo de las capas neuronales. Puede observarse que tras cada capa de neuronas se encuentra una función Dropout. Como ya fue explicado anteriormente, esta función permite desactivar de forma aleatoria un número de neuronas. Para este caso, la probabilidad de desactivación es del 50 %. El escalar que entra en cada capa Dropout sirve para resolver qué neuronas deben apagarse en esa iteración. Los dados que se observan en la figura, representan la aleatoriedad de la desactivación, pues para cada iteración, los nodos desactivados cambian aleatoriamente.

Esta función es fundamental para el entrenamiento. Si no se desactivaran neuronas, estas comenzarían a actuar como un filtro que memoriza trayectorias para alcanzar la solución. El resultado sería una particularización al conjunto de datos y la tendencia del entrenamiento desde el inicio iría hacia el overfitting.

6.1.1.1.- Resultados obtenidos

La parte del modelo correspondiente a capas de convolución, como ya fue adelantado anteriormente, no ha sido entrenada. Esto implica que las características que los filtros convoluciones extraen de las imágenes son de propósito general. Los mapas de características generados por las capas convolucionales, aportan información más clara sobre las imágenes. Este output del modelo VGG, será el input de la red neuronal. Se pretende que el nuevo input de la red sea mucho más enriquecedor para ella de lo que era una imagen. Esto se debe a que las redes neuronales simples, poseen muchas dificultades para obtener información de las imágenes. En la figura 6.13, se analiza en tiempo real el entrenamiento de la red.

Conjunto de datos	Precisión máxima	Época
Validación	0,4749	263
Entrenamiento	0,8009	2434

Tabla 6.1.- Resultados obtenidos para el modelo 1

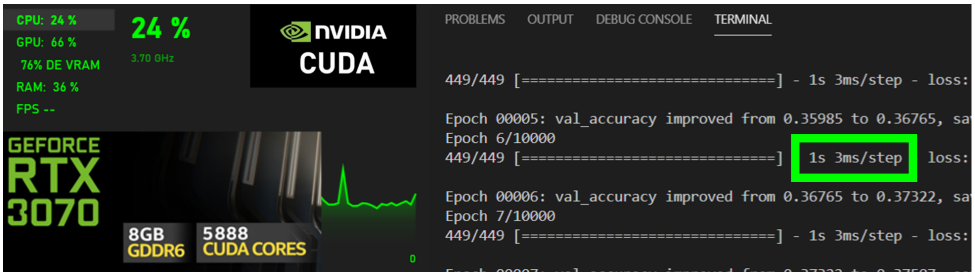


Figura 6.4.- Tiempo por época y carga del entrenamiento mediante GPU

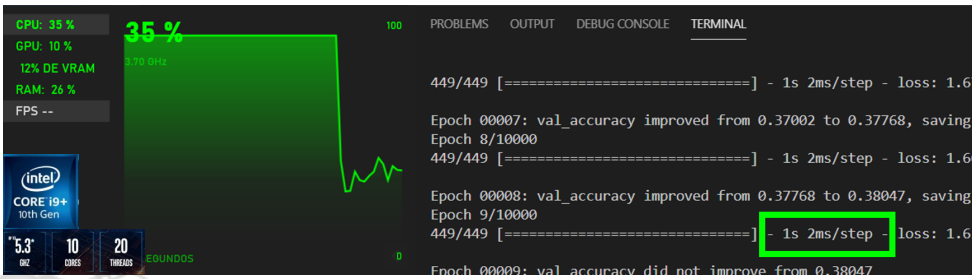


Figura 6.5.- Tiempo por época y carga del entrenamiento mediante CPU

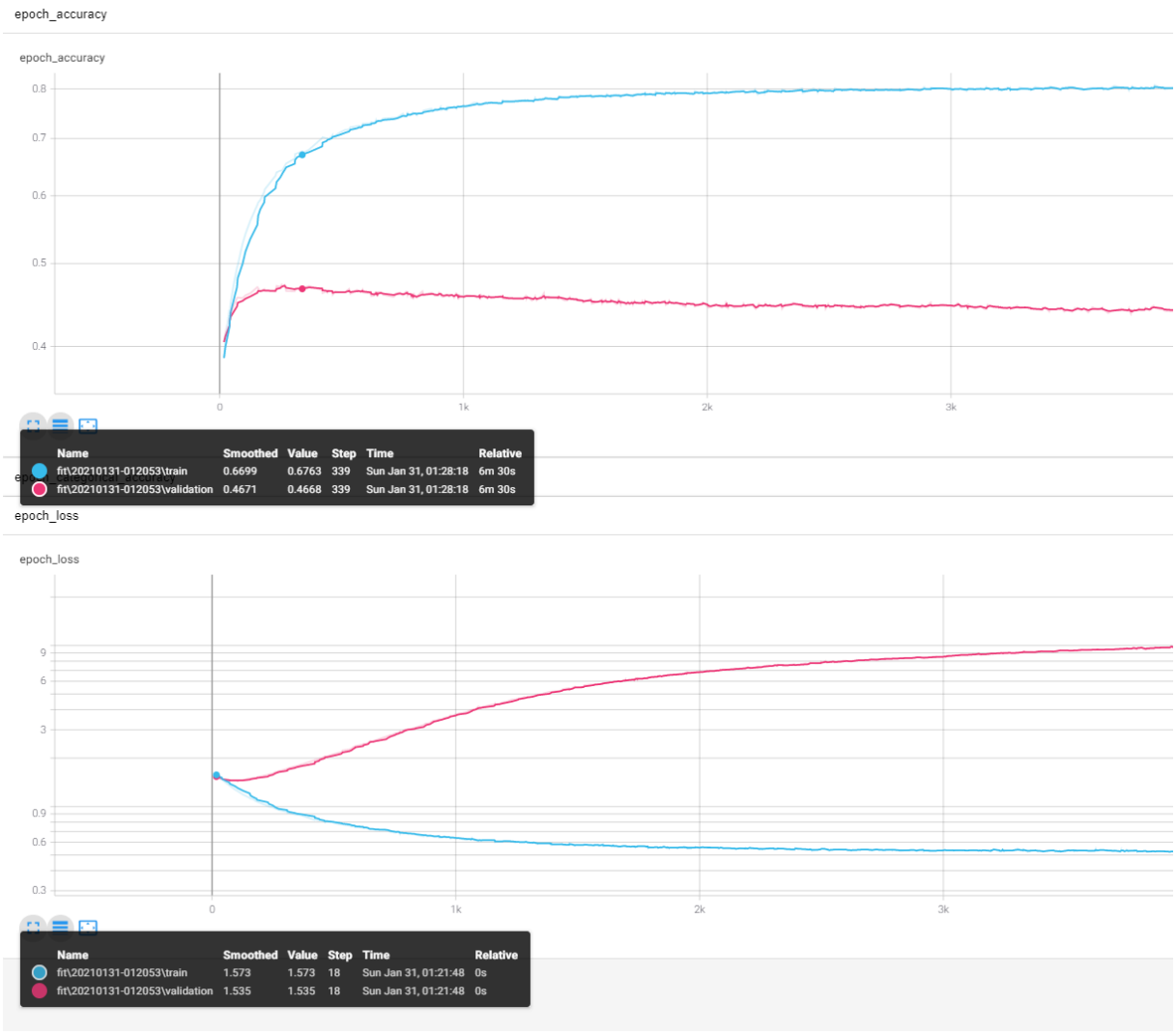


Figura 6.6.- Gráficas generadas en Tensorboard modelo 1

Respecto a los tiempos de entrenamiento, se observan en las figuras 6.4 y 6.5, valores muy similares. Esto sucede porque la red es muy simple, se presentan únicamente dos capas profundas. Los cálculos necesarios para reajustar los parámetros, pueden ser efectuados por el procesador de última generación, incluso ofrece un tiempo por iteración menor que la tarjeta gráfica.

Se observan dos gráficas diferentes, ambas representan una métrica en el dominio temporal, expresado mediante épocas. Para cada gráfica se representan dos funciones, la de color azul, corresponde a los datos de entrenamiento y la rosa, a los de validación.

La gráfica superior evalúa la precisión del modelo y la inferior, la función de pérdidas empleada. Representar el comportamiento del modelo frente a los datos de validación y de entrenamiento por separado, aporta gran información. Los fenómenos de entrenamiento explicados en la sección 5.7, no podrían apreciarse si ambas funciones estuvieran agrupadas.

Para la precisión del modelo, se observa siempre una tendencia logarítmica, que es la naturaleza de un aprendizaje. Al principio la red comienza con un conocimiento aleatorio, por tanto, los primeros ajustes tras las épocas iniciales ofrecen un gran incremento de la precisión. Tras 263 épocas, la precisión comienza a estabilizarse.

Por el contrario, la función de error tiene naturaleza exponencial decreciente. En las primeras inferencias, los aciertos son bajos, por tanto, el error es alto. Existe una relación entre el decrecimiento del error y el incremento de la precisión. Cabe mencionar que estas dos funciones no son inversamente proporcionales como aparentemente parece. Más adelante, en análisis más complejos se detallará este fenómeno.

La precisión máxima de este modelo para el conjunto de datos de pruebas es de **0,4749** mientras que para el conjunto de entrenamiento, es de **0,8009**. Los resultados no son buenos, suceden varios fenómenos al mismo tiempo. La precisión sobre el conjunto de pruebas se estabiliza en un valor bajo, inferior a 0,5 por lo que ya puede conocerse que no presentará un valor muy alto con el trascurso de las épocas.

Sin embargo, con el trascurso de las épocas decrece. Este efecto es verdaderamente negativo y retrata un evidente overfitting. Si a la par se compara con la precisión para el conjunto de entrenamiento, salta a la vista cómo su eficiencia es mucho mayor y se incrementa con el paso de las épocas. El modelo está aprendiendo a asignar emociones a las imágenes del conjunto de datos, siendo un fracaso cuando se enfrenta a las imágenes de pruebas. Todos los modelos tienen una tendencia a ser algo más eficientes frente a los conjuntos de datos, pero en este caso es radical.

El análisis de la precisión ya adelanta el resultado final, un modelo muy ineficiente. Aún así, se analizará la función de error para visualizar su comportamiento. En color azul, se

observa un correcto decrecimiento del error para el conjunto de entrenamiento. Como cabía esperar, el modelo se está convirtiendo en un experto para etiquetar aquellas imágenes que ya conoce.

Si se acota la función a un dominio correspondiente a las primeras épocas, la función de error de validación (representada en color rosa) desarrolla un decrecimiento correcto. Sin embargo, al poco tiempo se dispara delatando una pérdida de conocimiento. Cuanto más se entrena el modelo, más se ajusta al conjunto de datos. El overfitting es tan drástico que el modelo consigue incluso empeorar frente a los datos de validación.

Para este caso no es necesario un análisis profundo, pero se incluye de todas formas la matriz de confusión. La siguiente figura 6.7, muestra la matriz de confusión asociada al modelo.

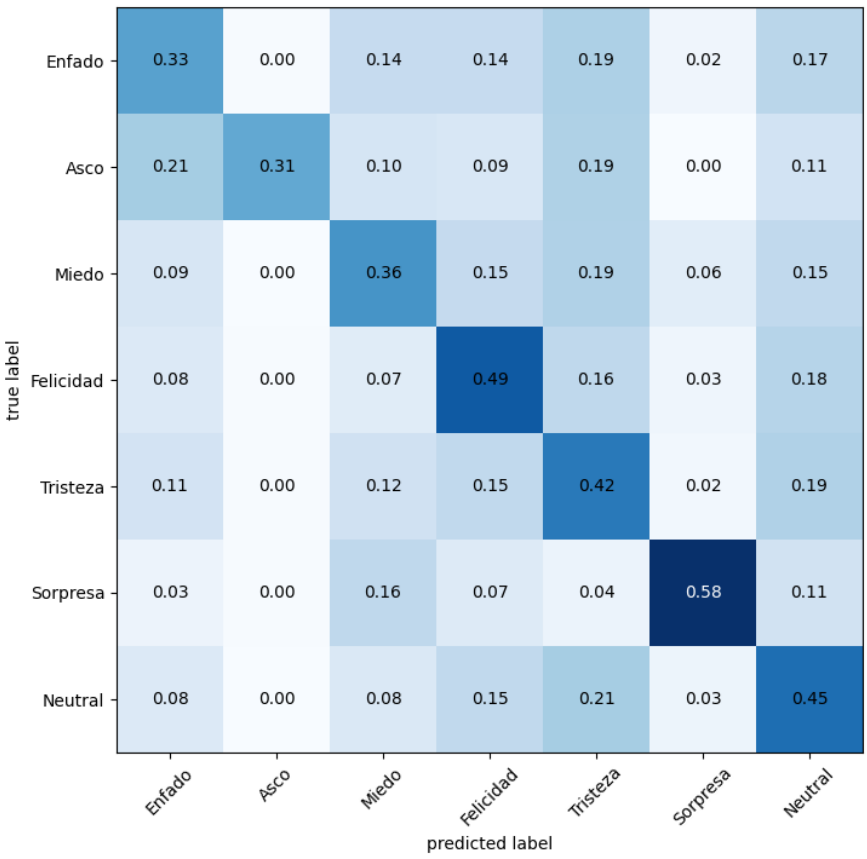


Figura 6.7.- Matriz de confusión modelo VGG16

Puede apreciarse que los aciertos para cada etiqueta, representados en la diagonal principal, son en su mayoría inferiores al 50 %. Las dudas que se plantean para cada clase carecen de relación lógica. Por ejemplo, para una rostro enfadado, se generaran dudas considerables con casi todas las demás clases. Esto muestra que el modelo apenas tiene conocimientos para generalizar y las confusiones que presenta, no son lógicas

6.1.1.2.- Conclusiones

Estos resultados plantean las siguientes conclusiones:

- Entrenar un excesivo número de épocas puede acentuar el overfitting o incluso provocarlo. Pueden ejecutarse entrenamientos largos en búsqueda de aumentos de la precisión, pero han de estar controlados para evitar el overfitting.
- Es posible que el modelo VGG16 no esté realizando un filtrado correcto para las imágenes. Al tratarse de identificar emociones de los rostros, es posible que sus filtros no sean los adecuados. Si fueron diseñados para identificar objetos o otros elementos, puede que no estén extrayendo buenas características de los rostros.

6.1.2.- VGG16 y Red Neuronal 1 capa

Se simplificaron las capas de la red neuronal para comprobar si podía incrementarse la precisión y reducirse el efecto de overfitting. La teoría del Deep Learning indica que una de las causas del sobre entrenamiento es la presencia de demasiados parámetros a ajustar en un modelo.

Los cambios respecto al modelo anterior son, únicamente, la reducción del número de capas y un control sobre las épocas para aportar más claridad con un dominio más corto. De esta forma, la estructura se mantiene similar a la de la figura 6.2.

6.1.2.1.- Resultados obtenidos

Los resultados obtenidos son similares al modelo anterior, incrementando ligeramente la precisión de validación. Se muestra una clara tendencia al overfitting tras apenas trascurrir

unas épocas. Se proyectan las imágenes de esta versión para ofrecer una mejor visión de los efectos ya detallados ampliamente en el caso anterior.

Conjunto de datos	Precisión máxima	Época
Validación	0,4948	650
Entrenamiento	0,8359	1807

Tabla 6.2.- Resultados obtenidos para el modelo 2

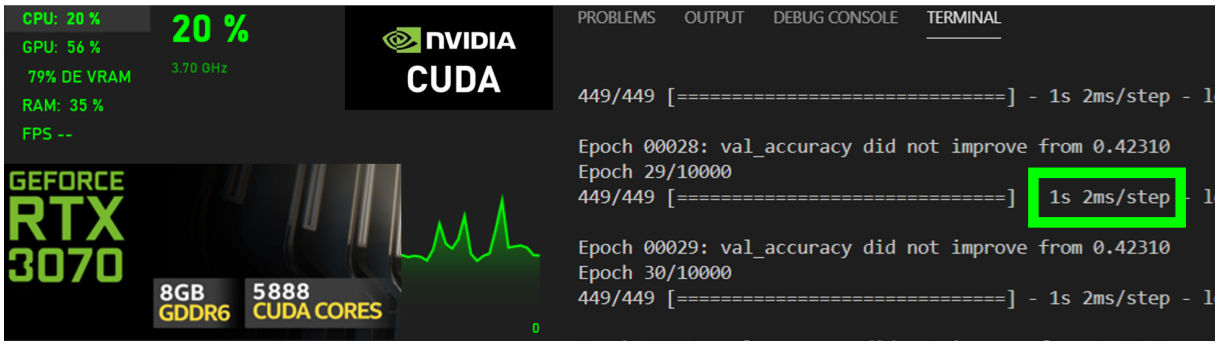


Figura 6.8.- Tiempo por época y carga del entrenamiento mediante GPU

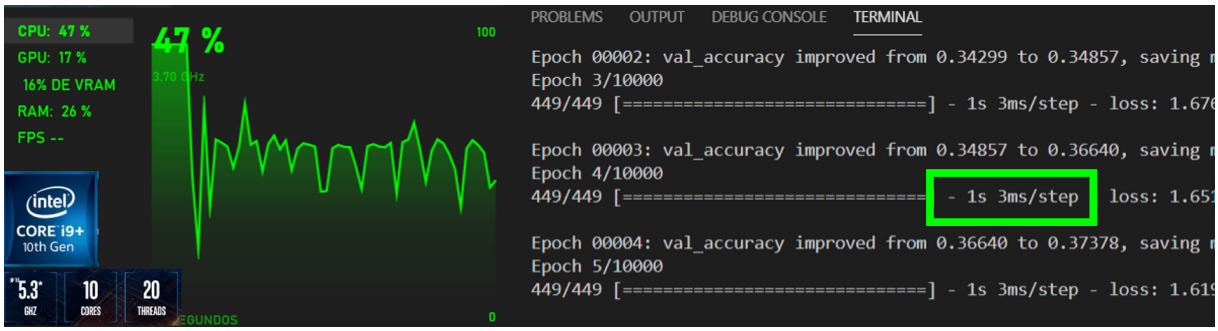


Figura 6.9.- Tiempo por época y carga del entrenamiento mediante CPU

Los tiempos de entrenamiento representados en las figuras 6.8 y 6.9, muestran el mismo efecto que el modelo anterior. Los parámetros entrenables del modelo son bajos por lo que la gráfica no muestra su potencial. El procesador mantiene un buen rendimiento para estos modelos simples.

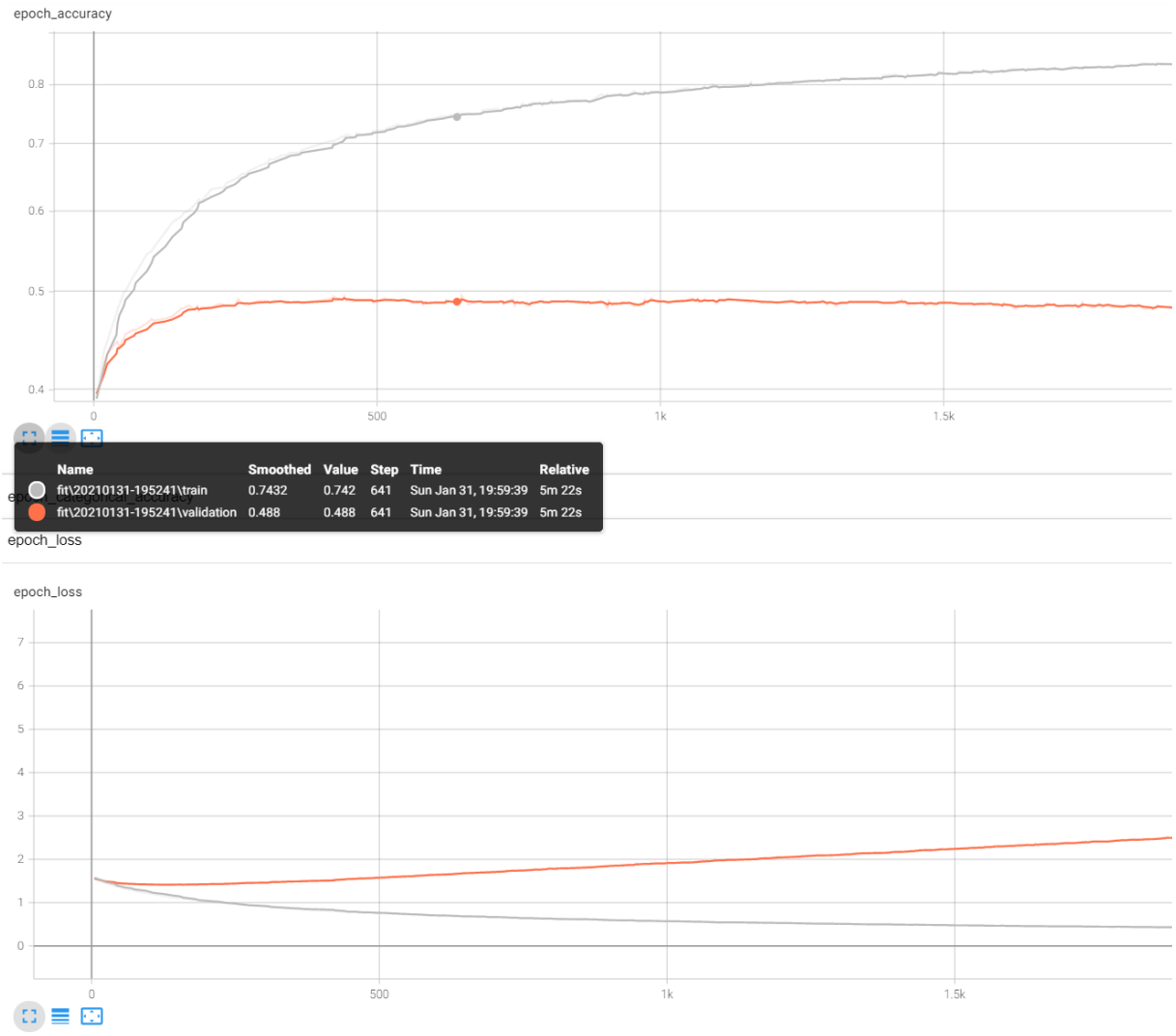


Figura 6.10.- Gráficas generadas en Tensorboard modelo 2

6.1.2.2.- Conclusiones

Al igual que ocurría en el caso anterior, se produce un claro ejemplo de sobre entrenamiento. Reducir los parámetros del modelo incrementó ligeramente la precisión sobre los datos de validación. Por contra, aceleró aún más el efecto del overfitting, llegando a conseguir una precisión de 0,8359 en los datos de entrenamiento en tan solo 1807 épocas.

Como se redujo el número de capas, la función Dropout solo se ejecutó una vez perdiendo efecto en el modelo. La conclusión es que el modelo VGG16 no aparenta ajustarse correctamente a las necesidades de este proyecto. Aporta filtros muy genéricos que

serán eficaces frente imágenes de más dimensiones para detectar formas de animales o personas. Sin embargo, no capta de forma correcta las expresiones de los rostros. Las capas neuronales pueden combinarse para lograr incrementar ligeramente el éxito, pero parten de unas características poco eficientes.

Este modelo representa un buen ejemplo de absoluta memorización de los datos, si el entrenamiento se alarga a 10.000 épocas, la precisión sobre el conjunto de datos puede alcanzar el 95 %. Paralelamente, sus predicciones generalizadas no alcanzan el 50 %.

6.1.3.- VGG19 y Red Neuronal 3 capas

Para observar si se aportaba alguna mejora empleando otro modelo pre entrenado, se desarrolló el mismo ejemplo usando VGG19. La estructura es similar a la empleada en el modelo 6.1.1. Al tratarse del VGG19, se incluyen 3 capas más de convolución, manteniéndose intacto el resto de la estructura.

Como estos ejemplos ya han sido ampliamente descritos, se detallarán de forma breve los resultados.

6.1.3.1.- Resultados obtenidos

Los resultados obtenidos se mantienen en la línea de los ejemplos anteriores. Se obtienen incluso peores cifras para la precisión: “0,4593” en el punto máximo. Se añadió además un incremento de la probabilidad para la función Dropout. Esto produce una mayor desactivación de neuronas consiguiendo frenar más el overfitting. Analizando en detenimiento las primeras épocas, se observa se observan pendientes menos pronunciadas para los datos de entrenamiento. Esto indica que el uso de la función para la desactivación de neuronas produce resultados. De igual forma, el modelo en pocas épocas vuelve a adoptar la tendencia al overfitting. Si se continuara el entrenamiento mayor número de épocas con absoluta seguridad, la curva de precisión para el conjunto de datos continuaría creciendo.

Conjunto de datos	Precisión máxima	Época
Validación	0,4593	323
Entrenamiento	0,71	1362

Tabla 6.3.- Resultados obtenidos para el modelo 3

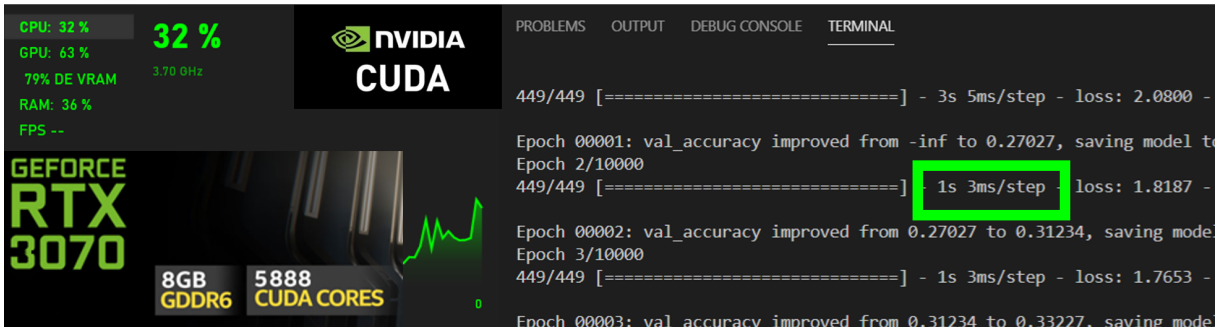


Figura 6.11.- Tiempo por época y carga del entrenamiento mediante GPU

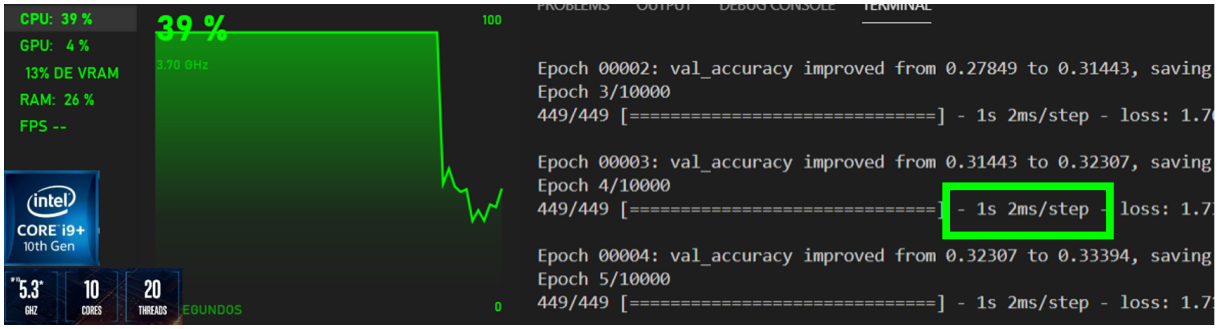


Figura 6.12.- Tiempo por época y carga del entrenamiento mediante CPU

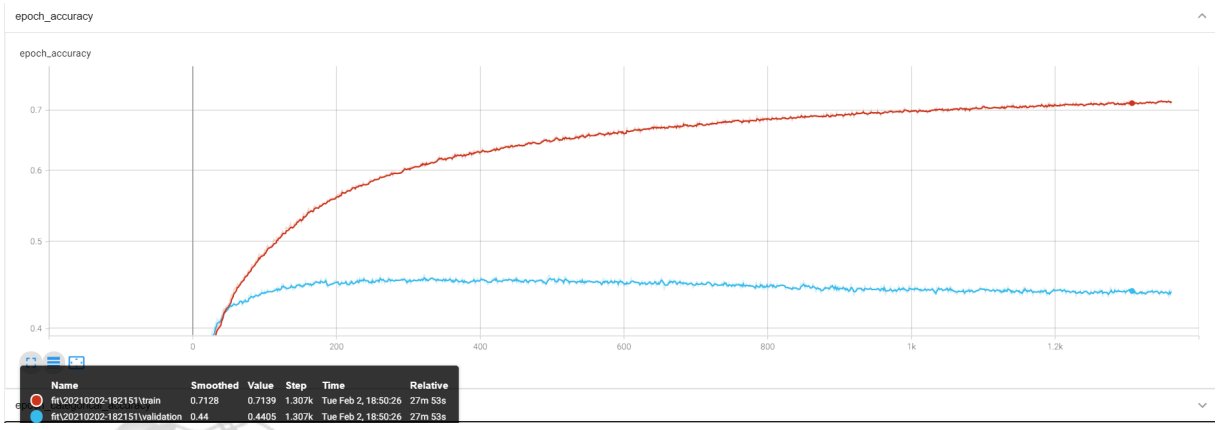


Figura 6.13.- Gráficas generadas en Tensorboard modelo 3

6.1.3.2.- Conclusiones

Los modelos pre entrenados no aportan buenos resultados. Emplean demasiadas capas convolucionales que no aportan una buena extracción de características. Como ya fue mencionado, estos modelos tienen un propósito demasiado general que no se adapta con las necesidades de este proyecto.

Además de las modificaciones de interés ya mencionadas, fueron probadas diferentes configuraciones. Algunos hiperparámetros fueron alterados, así como el número de capas neuronales o sus funciones de activación. Los resultados son bastante semejantes, se consiguen algunas pequeñas mejoras pero dado que el punto de partida no es bueno, el modelo se considera ineficiente.

Sería posible trabajar aún más en la optimización de los hiperparámetros y capas para mejorar estos modelos VGG. Aún así, se considera más apropiado probar otros enfoques donde quizás aparezca mayor rendimiento. Con un modelo más solido, trabajar en los hiperparámetros generará resultados más satisfactorios. La optimización de los modelos VGG para este proyecto podría mejorar algunas cifras, pero no supondrá nunca un cambio drástico.

Estos modelos tienen demasiados parámetros internos debido a su excesivo número de capas convolucionales. Estos modelos VGG exprimen todo su potencial en proyectos que trabajen con imágenes de mayor resolución y pretendan detectar elementos más genéricos. Las características que se buscan en las imágenes de rostros, son muy particulares y el uso de demasiados filtros, no les genera un buen mapa de características.

La red neuronal multicapa parte, por tanto, de una mala interpretación de los datos donde no consigue encontrar buenos patrones comunes. Para emociones como la sorpresa o la felicidad, se mostraba una ligera mejora de la eficiencia respecto a las demás. Los rasgos que las caracterizan son algo más representativos y por consiguiente, se habrán pronunciado más en los mapas de características generados tras las convoluciones. Esto permitía a la red neuronal poder relacionarlos con la etiqueta correcta.

6.2.- Modelos CNN

Los modelos de CNN, redes neuronales convolucionales, emplean un funcionamiento similar a los VGG pero mucho más particularizado. Para estos modelos, se crearán las capas convolucionales que se deseen en cada caso así como el segmento de red neuronal multicapa. Es decir, este modelo será ajustado de forma completa para este proyecto, donde todos los parámetros podrán ser seleccionados. Esto permitirá efectuar muchas más pruebas y tener un mayor control sobre los efectos que se produzcan.

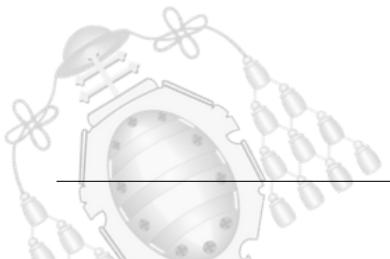
Al mismo tiempo, ralentizará los entrenamientos dado que ahora, todos los parámetros de la red serán entrenados. Los filtros kernel, adaptarán sus valores para encontrar los idóneos para el conjunto de datos.

Dados los resultados de los modelos anteriores, se ha comprobado que aumentar el número de parámetros no es positivo para nuestro caso. Por ello, se platearán modelos mucho más simples. Añadir tantas capas convolucionales, además de no ofrecer buenos resultados, eternizaría los entrenamientos.

Las características de la red neuronal multicapa, se alterarán para optimizarse, pero no suponían el principal problema de los casos anteriores. Es por esto que las expectativas de mejora se enfocan primero a las capas convolucionales.

6.2.1.- Modelo CNN 1

En este modelo se diseña de forma especifica la estructura secuencial de la red neuronal. Se emplearán tres capas convolucionales y una capa de neuronas dense, sin contar la capa de output. El orden de las capas habitual, es aplicar una capa convolucional y después añadir la operación MaxPooling. De esta forma, puede observarse en la tabla 6.4 la estructura.



	Operaciones	Número	Tamaño
Capa 1	Convolución	32	Filtro(3,3)
	MaxPooling	-	(2,2)
Capa 2	Convolución	64	Filtro(3,3)
	MaxPooling	-	(2,2)
Capa 3	Convolución	128	Filtro(3,3)
	MaxPooling	-	(2,2)
Flatten			
Capa1	Neuronas 1024	Activación ReLU	
	Dropout	0,5	
Capa output	Neuronas 7	Activación Softmax	

Tabla 6.4.- Estructura modelo CNN 1

6.2.1.1.- Resultados obtenidos

Tras haber reducido el número de convoluciones, los resultados son muy diferentes. Se encuentra una gran mejoría en la precisión de los datos de validación. El incremento es casi del 20 %, lo que es un gran éxito. La precisión de validación se sitúa en "0,6050", un valor más razonable para comenzar a optimizar. Ha de tenerse en cuenta que las inferencias son realizadas para predecir emociones, lo que no es una tarea tan objetiva como la de otros proyectos. El grado de subjetividad de las imágenes reduce el marco de precisión esperado para este tipo de proyecto. La tabla 6.5, muestra los mejores valores obtenidos y en qué época se alcanzaron.

Conjunto de datos	Precisión máxima	Época
Validación	0,6050	794
Entrenamiento	0,99	148

Tabla 6.5.- Resultados obtenidos para el modelo CNN 1

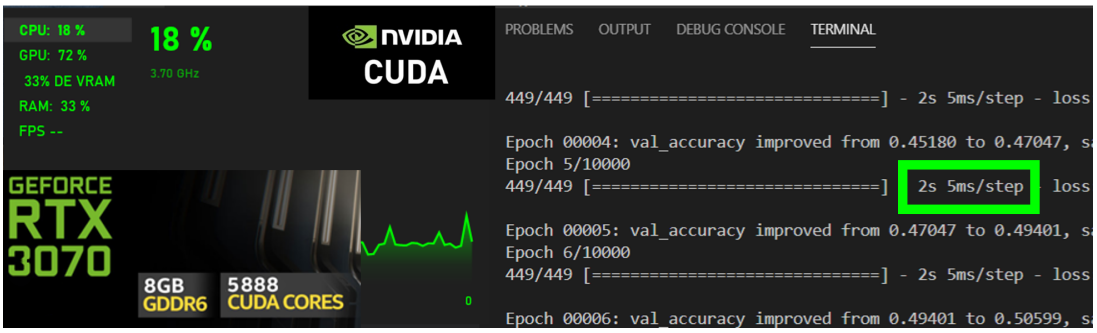


Figura 6.14.- Tiempo por época y carga del entrenamiento mediante GPU

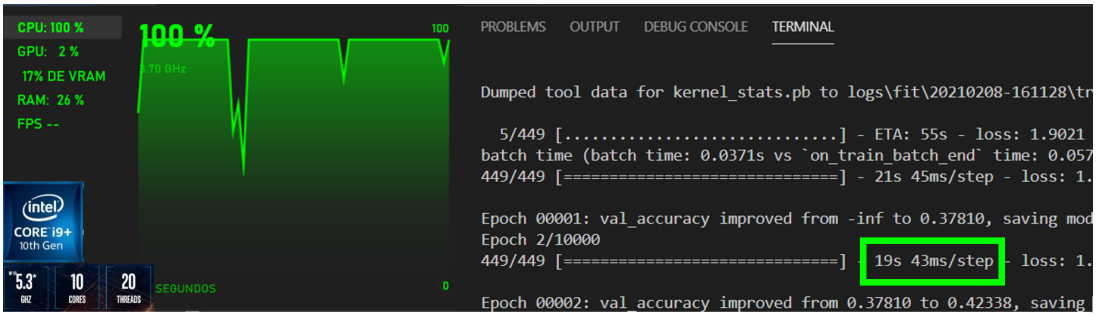


Figura 6.15.- Tiempo por época y carga del entrenamiento mediante CPU

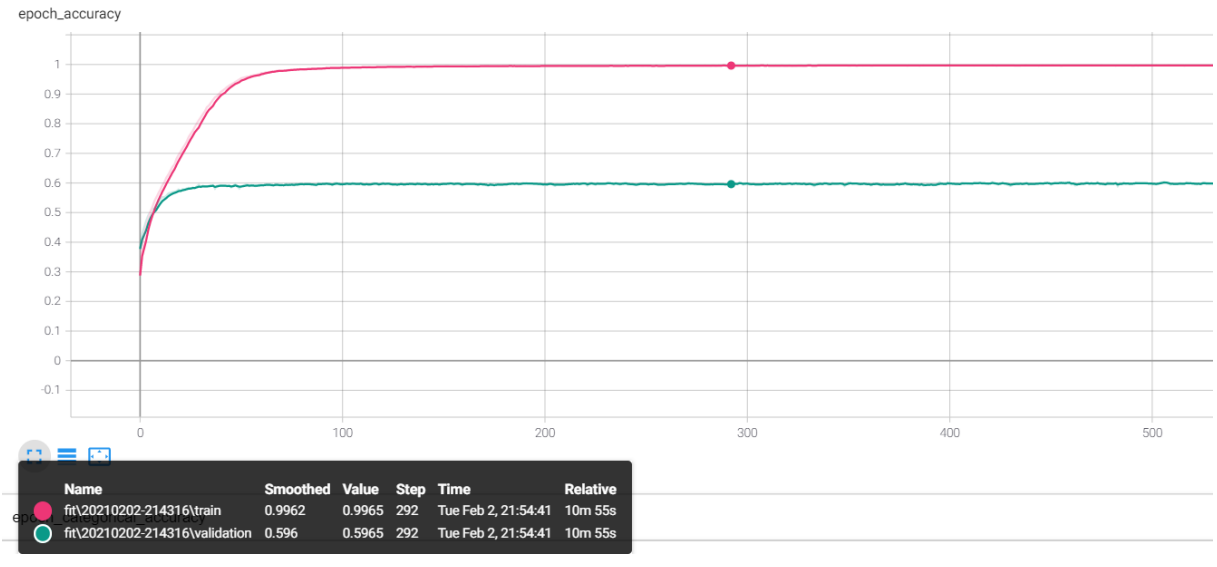


Figura 6.16.- Gráficas generadas en Tensorboard modelo CNN 1: precisión por época

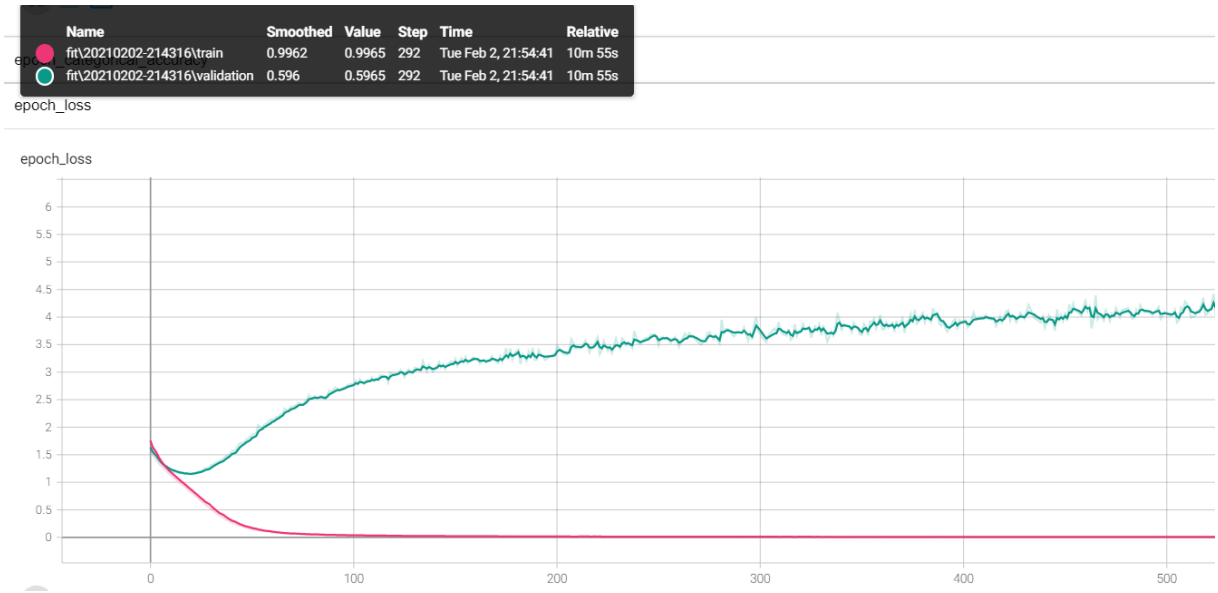


Figura 6.17.- Gráficas generadas en Tensorboard modelo CNN 1: error por época

Los tiempos de entrenamiento y en la carga de los componentes se observan cambios respecto a las redes neuronales simples. Al tratarse de una red convolucional comienza a observarse como el procesador es mucho más ineficiente que la tarjeta gráfica. Incluso tratándose de una de las últimas generaciones, el procesador a 100 % de rendimiento es 10 veces más lento que la GPU. El número de filtros que se aplican en esta estructura no es muy elevado, a pesar de ello, ya se aprecia la diferencia entre la GPU y la CPU. También la tarjeta gráfica comienza a mostrar mayor rendimiento, las operaciones para calcular los parámetros y sesgos le suponen una carga del 72 %. Respecto a la precisión sobre los datos de entrenamiento, está claro que se ha incrementado en exceso, produciéndose un sobre entrenamiento. Para ese aspecto, se sigue manteniendo el efecto que ya se visualizaba en los modelos anteriores. La tendencia del modelo a sobre ajustar los parámetros es clara y su reducción ha de ser un objetivo. Gracias los cheackpoints que se efectúan durante el entrenamiento, puede utilizarse un modelo correspondiente a épocas intermedias con buena precisión. Además, puede controlarse la ejecución para detenerla cuando se observe que no habrá un incremento de la precisión sobre los datos de validación.

Esto se refiere a que los grandes efectos tan pronunciados en estas gráficas, no se corresponderán con el modelo final. Para seleccionar los modelos a utilizar, se controla en

qué época interesa detener el entrenamiento. Emplear el último modelo generado puede no afectar al rendimiento real, pero está claro que no aportará algo positivo. Esto se refiere a que la precisión sobre los datos de validación no cambia, por lo que el modelo continúa manteniendo su eficiencia en las pruebas. Igualmente es un efecto que no aporta rendimiento real, por lo que ha de ser controlado.

Es interesante hacer zoom en la gráfica 6.17 para estudiar los puntos claves que se observaron. Reducir el rango de análisis a 100 épocas, facilita observar las tendencias principales ya que suceden en los inicios del entrenamiento.

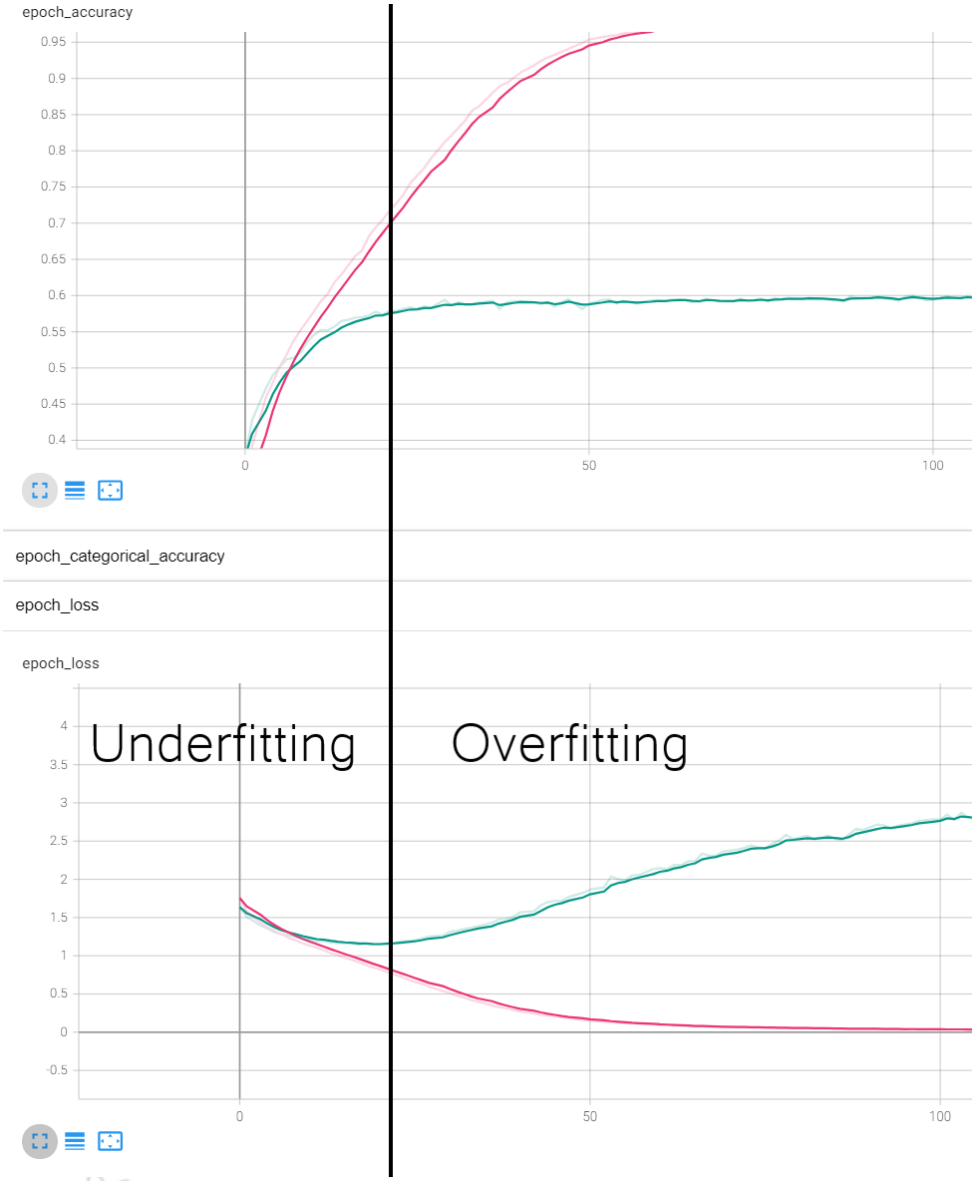


Figura 6.18.- Gráficas acotadas a 100 épocas del modelo CNN 1

En la figura 6.19, se traza una línea vertical en el mínimo de la función de error. La teoría sobre el Deep Learning, ya explicada anteriormente, distingue los dos efectos visibles en la gráfica. El primer tramo del entrenamiento, corresponde a la etapa de underfitting. Este efecto no representa un problema, ya que la función de error decrece. Esta minimización del error implica que se está produciendo un aprendizaje. Tras pasar por el punto mínimo, la función comienza a crecer. Este fenómeno es algo sorprendente si se estudia en comparación a la precisión de los datos de validación. En casos anteriores, se indicaba un claro ejemplo de overfitting cuando la función del error crecía y la precisión decrecía. Sin embargo, en este caso puede observarse que la precisión de validación se mantiene constante. Por tanto surge un caso no tan contemplado en la teoría, un aumento del error al mismo tiempo que se sostiene la precisión. Este fenómeno tan curioso se estudiará en los próximos ejemplos para analizar por qué sucede.

Detener el entrenamiento en la época que indica la línea vertical negra, ofrece un modelo de aproximadamente un 60 % de precisión para datos de validación y un 70 % para los datos de entrenamiento. En este punto, el modelo puede considerarse apto, pues la diferencia entre ambas precisiones es baja. La validación de pruebas siempre tenderá a alzarse más que la de validación, este efecto es esperado y admisible mientras no sea muy pronunciado. A partir de la época 40, esta distancia comienza a hacerse problemática tendiendo a un claro sobre ajuste que se desea evitar.

Dadas las mejoras de este modelo, se genera su matriz de confusión para los datos de validación.

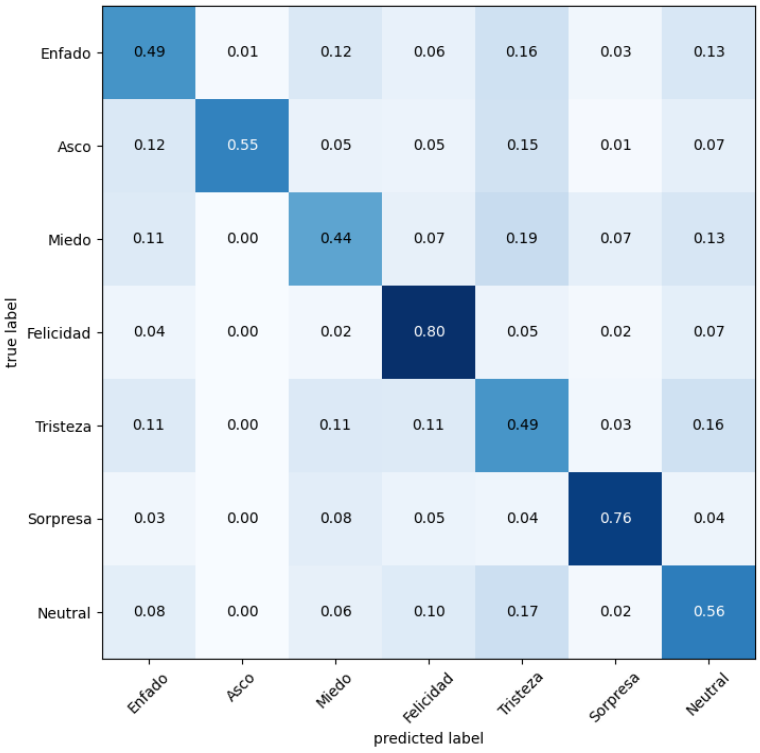
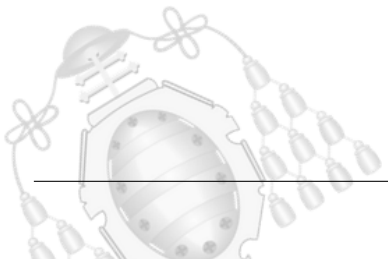


Figura 6.19.- Matriz de confusión modelo CNN 1

El modelo comienza a presentar valores muy buenos para las categorías de felicidad y sorpresa. Al mismo tiempo, pueden analizarse las confusiones producidas, que siguen una tendencia lógica. En el análisis de la dataset ya se planteó el problema de las imágenes ambiguas, donde categorizar un rostro como neutral o triste, estaba sujeto a cierto grado de subjetividad. Este tipo de indecisiones, se visualizan en la matriz de confusión indicándonos que el modelo verdaderamente comienza a tener inteligencia y plantea dudas realistas. En este punto, aparece el dilema de la objetividad de la base de datos, pues en un rostro que presenta miedo, se encuentran patrones relacionados con la tristeza o la sorpresa. También algunas imágenes presentan una emoción sutil, lo que da como resultado una emoción con cierto grado de neutralidad. Que la base de datos este desbalanceada también puede producir repercusión sobre los datos. La influencia de la base de datos sobre los resultados, será un punto de estudio para los siguientes modelos.

6.2.1.2.- Conclusiones

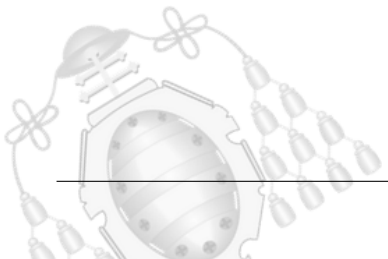
Se observaron incrementos en la precisión de validación, lo que coloca a las CNN en el punto de desarrollo. Se deben optimizar los valores de precisión, comprender los efectos que están ocurriendo y lidiar con el sobre ajuste. Conseguir que la precisión de entrenamiento sea igual a la de validación es un efecto complejo, ya que como se mencionó anteriormente, la precisión sobre los datos de entrenamiento tiende a ser más elevada que la de validación. El objetivo, será reducir la pendiente con la que crece la función de precisión (para los datos de entrenamiento). Si este efecto se logra, el entrenamiento podrá ejecutarse durante más épocas, logrando así que el modelo adquiera más conocimiento.



6.2.2.- Modelo CNN 2

Para este modelo se añade un pre procesamiento de los datos. Para poder realizar un estudio continuista, se mantendrán los hiperparámetros del modelo anterior. Realizar muchas modificaciones dificulta el estudio del modelo, ya que, sería difícil asociar las mejoras observadas con los cambios que fueron realizados. Efectuar cambios graduales permite comprender lo que está sucediendo en los entrenamientos y conocer el impacto de cada hiperparámetro. Cuando las estructuras que favorecen al modelo estén claras, solo se deberá combinarlas para lograr la optimización deseada.

El prepocesamiento de imágenes permite generar nuevas entradas y nutrir más el aprendizaje. Las imágenes que se crean artificialmente presentarán giros, desplazamientos o cambios de perspectiva. Estas transformaciones pueden aportar una mejora en la precisión de validación y en el uso real del modelo. Durante la detección de rostros, las caras capturadas pueden estar de perfil, sufrir algún giro o presentar perspectivas diferentes. La mayor parte del conjunto de datos, muestra imágenes centradas y con ángulos similares. Por ello, aplicar el pre procesamiento, puede suponer mayor eficiencia, no solo en los datos de validación, sino también en su uso real.



	Operaciones	Número	Tamaño
PRE PROCESAMIENTO			
Capa 1	Convolución	32	Filtro(3,3)
	MaxPooling	-	(2,2)
Capa 2	Convolución	64	Filtro(3,3)
	MaxPooling	-	(2,2)
Capa 3	Convolución	128	Filtro(3,3)
	MaxPooling	-	(2,2)
Flatten			
Capa 1	Neuronas 1024	Activación ReLU	
	Dropout	0,5	
Capa output	Neuronas 7	Activación Softmax	

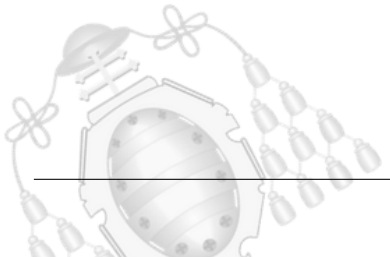
Tabla 6.6.- Estructura modelo CNN 2

6.2.2.1.- Resultados obtenidos

Conjunto de datos	Precisión máxima	Época
Validación	0,6503	119
Entrenamiento	0,6829	148

Tabla 6.7.- Resultados obtenidos para el modelo CNN 2

Se observa un incremento en la precisión de validación de aproximadamente un 5 % respecto al modelo anterior. El pre procesamiento de los datos ha causado un impacto muy positivo. Al mismo tiempo, la precisión de entrenamiento está mucho más controlada manteniéndose con “0,6829” en la época 148.



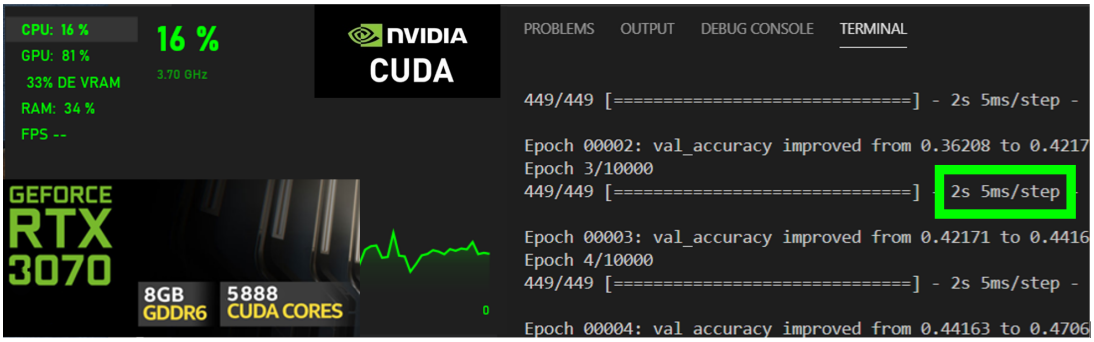


Figura 6.20.- Tiempo por época y carga del entrenamiento mediante GPU

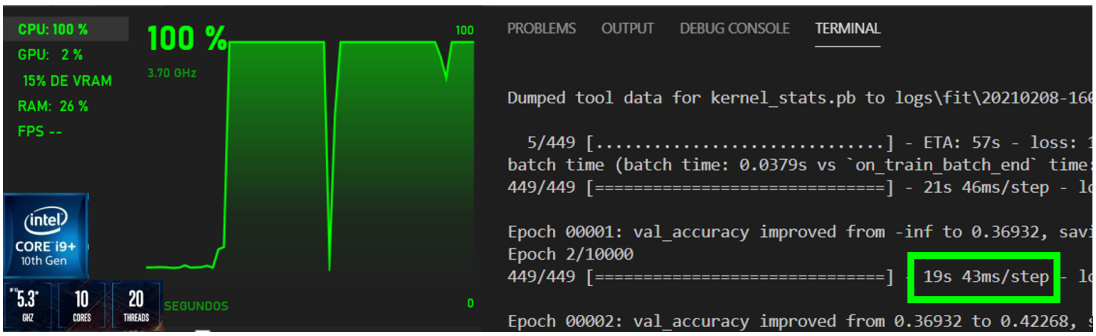


Figura 6.21.- Tiempo por época y carga del entrenamiento mediante CPU

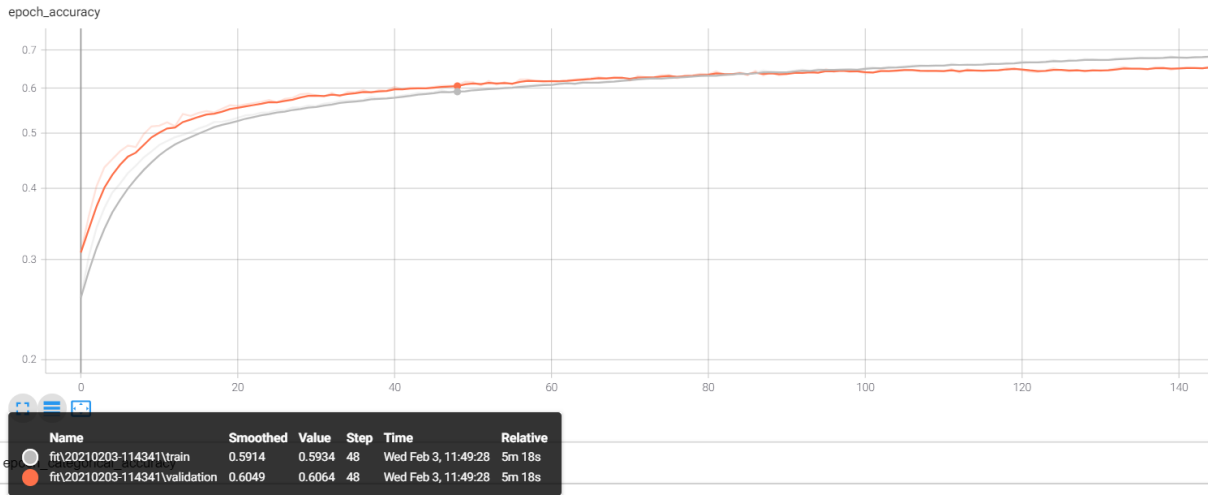


Figura 6.22.- Gráficas generadas en Tensorboard modelo CNN 2: precisión por época

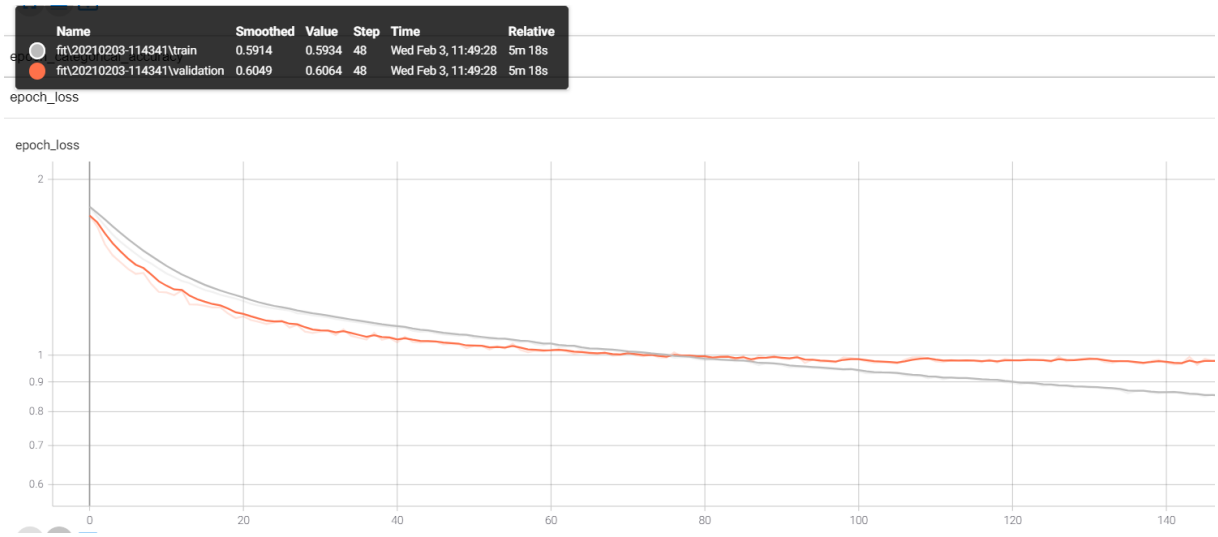


Figura 6.23.- Gráficas generadas en Tensorboard modelo CNN 2: error por época

Respecto a la carga computacional, se observa una tendencia similar a la del modelo CNN 3. El rendimiento de la GPU asciende un 10 % y es mucho más eficiente que la CPU. Aplicar el pre procesamiento de los datos ha generado resultados muy buenos. Se estudiaron aproximadamente 150 épocas, por lo que la precisión sobre los datos de entrenamiento indicada en la tabla , se corresponde a haber finalizado antes el aprendizaje. De haber continuado muchas más épocas, habría aumentado como ocurre en todos los casos. Los resultados y tendencias de interés, se aprecian en este dominio más acotado, es por ello que no se ha alargado más el entrenamiento.

Las gráficas representadas en las figuras 6.22 y 6.23, muestran tendencias muy buenas. Ambas funciones de error presentan una pendiente negativa correspondiente con una minimización del error. Al comienzo del entrenamiento, ambas decrecen más rápido y comienzan a estabilizarse tras varias épocas. Es el primer modelo donde se logra un entrenamiento fructífero durante 150 épocas. Paralelamente, las gráficas de precisión y entrenamiento muestran tendencias muy positivas. La curva de precisión de validación crece de forma progresiva a lo largo de las épocas sin estancarse en las primeras. Esto le permite iterar más sobre el conjunto de datos para adquirir un incremento de precisión respecto a los modelos anteriores.

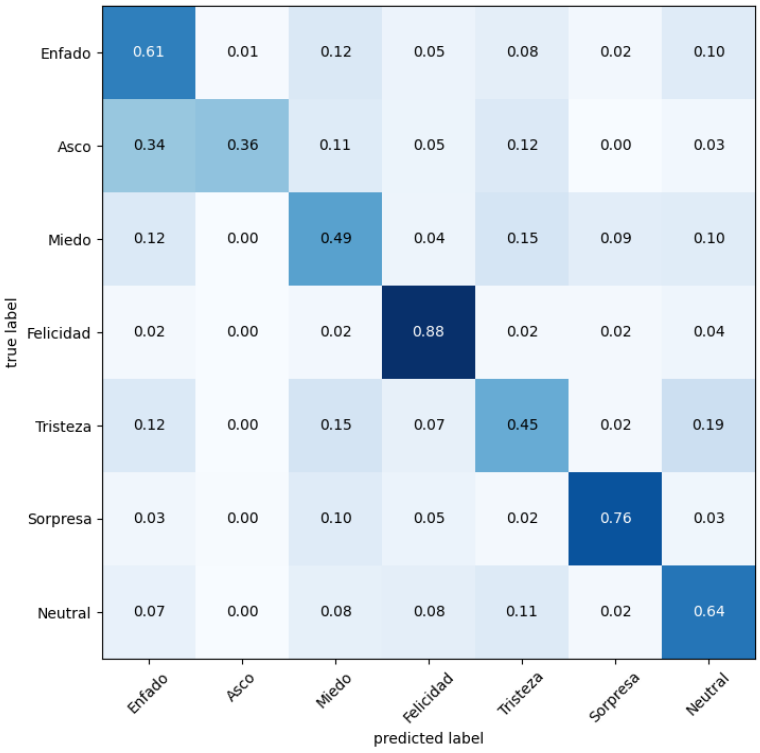


Figura 6.24.- Matriz de confusión modelo CNN 2

En la matriz de confusión, representada en la figura 6.24, se observa un aumento de la precisión para casi todas las categorías. Para “enfado”, “neutral” y “felicidad” se observa un aumento claro. Comienzan a observarse efectos extraños en la categoría de “asco”. El aprendizaje en balance general es positivo, sin embargo, está categoría parece experimentar dificultades extras. Si se recuerda el capítulo de análisis del conjunto de datos, las imágenes de “asco” eran mucho menores que las del resto. Esta desproporción, que ya se venía adelantando en análisis anteriores, comienza ya a ser evidente. Las presencia de la etiqueta “asco” con poco conocimiento, confunde al modelo ya que no tiene apenas información sobre ella. Las mejoras en la precisión del modelo se consiguen incluso con la drástica disminución de la precisión para esta etiqueta.

La dificultad para distinguir entre algunas emociones dada su poca objetividad, sigue presente. Pueden denominarse emociones frontera, caracterizándose por representar más de una categoría y confundir el proceso de entrenamiento. La tristeza y la neutralidad,

presentan un nexo común que confunde al modelo, siendo el causante del 17 % de los fallos para esta categoría. Las imágenes con una muestra de felicidad poco significativa, producen también confusión. El modelo decide catalogarlas como neutral, pues todavía considera que deben ser de esa categoría. Sin embargo, para el conjunto de datos ya son consideradas personas felicites. Este efecto de emociones poco marcadas, supone la causa del error para la categoría neutral. Se aprecia en la existencia de un pequeño porcentaje de confusión para casi todas las demás categorías. Las dos etiquetas donde no se observa este comportamiento son: "asco", que presenta un minúsculo conjunto de datos y "sorpresa", que otorga una diferenciación muy clara respecto a las demás.

En las siguientes gráficas se representa una comparación de este modelo CNN 2 respecto al CNN 1 anterior.

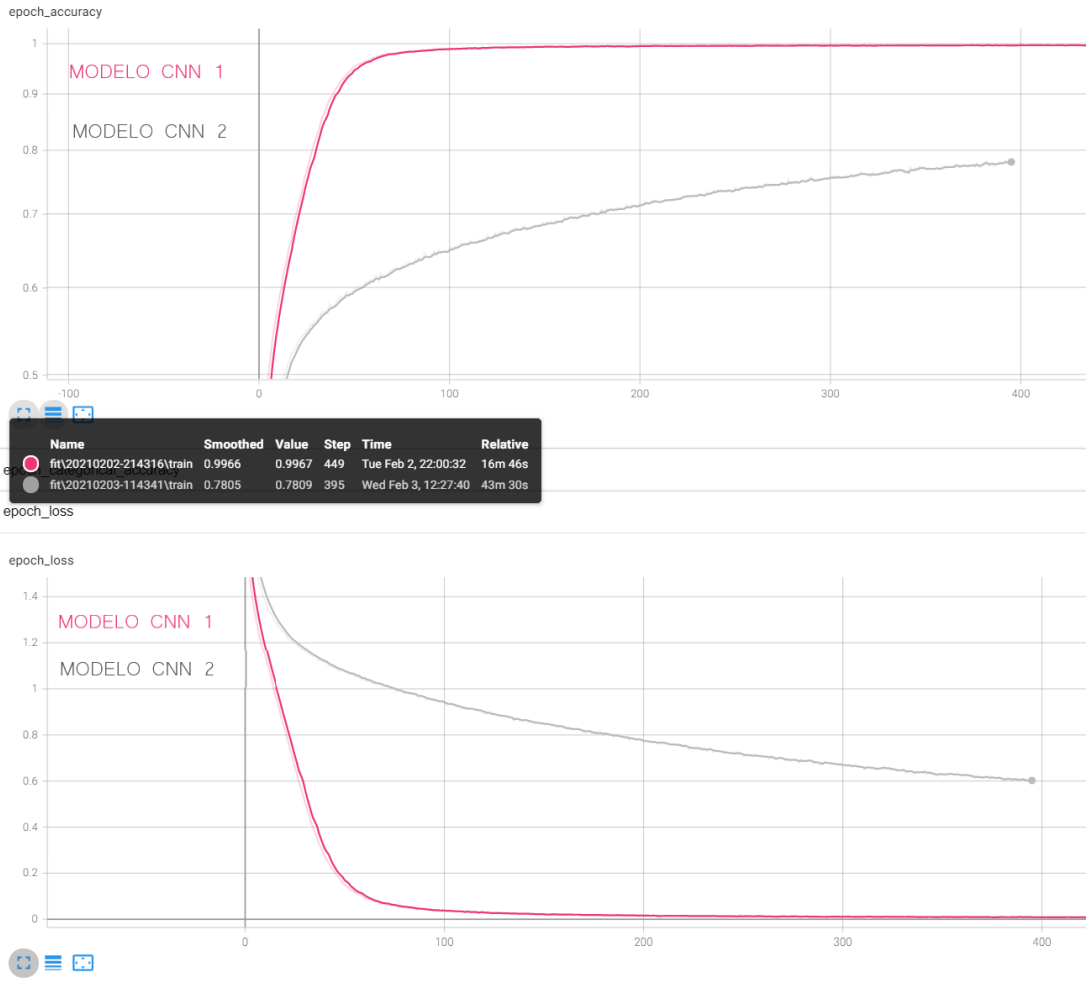


Figura 6.25.- Comparativa precisión de entrenamiento

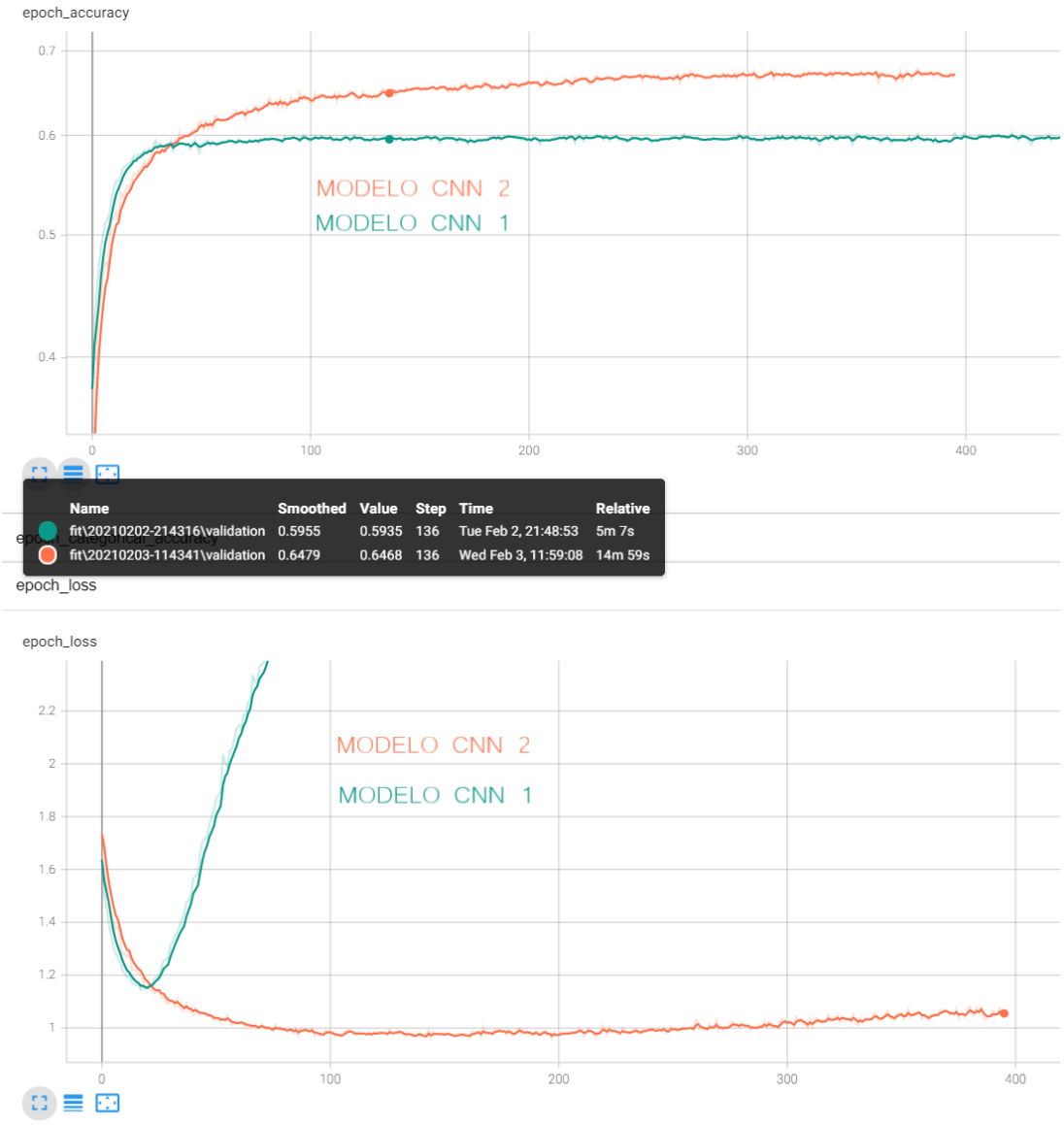


Figura 6.26.- Comparativa precisión de validación

En la figura 6.25, se compara la evolución de la precisión para los datos de entrenamiento que ofrecen ambos modelos a lo largo de las épocas. En la figura 6.26, se realiza la comparativa de la precisión para los datos de validación. Las gráficas demuestran las mejoras ya mencionadas.

6.2.2.2.- Conclusiones

Este modelo que aplica pre procesado a las imágenes ha conseguido unas mejoras muy importantes en los entrenamientos. Para el modelo anterior, la precisión de entrenamiento se encontraba en el 99 % ya en la época 148. Sin embargo, ahora solo presenta un 0,68 % en esa misma época. Además, la curva se mantiene con una pendiente mucho más baja, evitando el claro overfitting que se observaba en el resto de modelos. Este logro, abre la posibilidad de efectuar entrenamientos más largos con propósito de acercar la precisión al %70, que es la cifra objetivo que se ha establecido, dadas las características del conjunto de datos.

6.2.3.- Modelo CNN 3

Para llegar al modelo CNN 3, se han efectuado muchas pruebas previas. Dado que no se observó mejoría ni información de interés en ellas, no serán descritas. El modelo 3 corresponderá a una estructura que ofreció muy buenos resultados. Se aplicaron los preprocesamientos ya antes mencionados y empleados dada su gran eficiencia. Los cambios aplicados consisten en una capa convolucional adicional, incluir funciones Dropout en las capas convolucionales y añadir más complejidad a la red neuronal multicapa. La estructura secuencial que presenta es la que puede observarse en la tabla 6.8

	Operaciones	Número	Tamaño
PRE PROCESAMIENTO			
Capa 1	Convolución	32	Filtro(3,3)
	Dropout	-	0,25
Capa 2	Convolución	64	Filtro(3,3)
	MaxPooling	-	(2,2)
	Dropout	-	0,25
Capa 3	Convolución	128	Filtro(3,3)
	MaxPooling	-	(2,2)
	Dropout	-	0,25
Capa 4	Convolución	128	Filtro(3,3)
	MaxPooling	-	(2,2)
	Dropout	-	0,25
Flatten			
Capa 1	Neuronas 2048	Activación ReLU	
	Dropout	0,5	
Capa 2	Neuronas 1024	Activación ReLU	
	Dropout	0,5	
Capa output	Neuronas 7	Activación Softmax	

Tabla 6.8.- Estructura modelo CNN 3

6.2.3.1.- Resultados obtenidos

Conjunto de datos	Precisión máxima	Época
Validación	0,702	297
Entrenamiento	0,7389	297

Tabla 6.9.- Resultados obtenidos para el modelo CNN 3

Los resultados para este modelo son muy buenos, suponen una mejora casi del 5 % respecto a la versión anterior y optimizan el entrenamiento. El modelo más alto registrado fue el de la época 218, donde la precisión de validación presentaba un valor de"0,702" mientras que la de entrenamiento "0,7389". La diferencia entre ambas se encuentra dentro de lo correcto y esperado en un entrenamiento. Las siguientes gráficas muestran el proceso de evolución de las primeras épocas, donde se observa que la pendiente de la función de precisión, para el tramo inicial, ha incrementado.

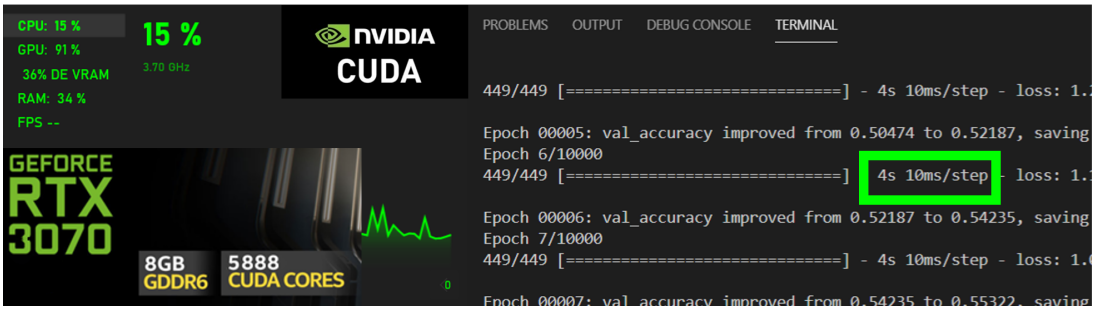


Figura 6.27.- Tiempo por época y carga del entrenamiento mediante GPU

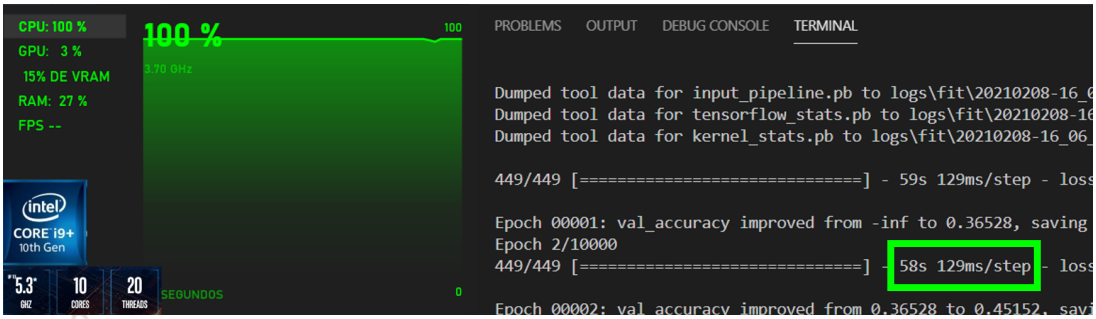


Figura 6.28.- Tiempo por época y carga del entrenamiento mediante CPU

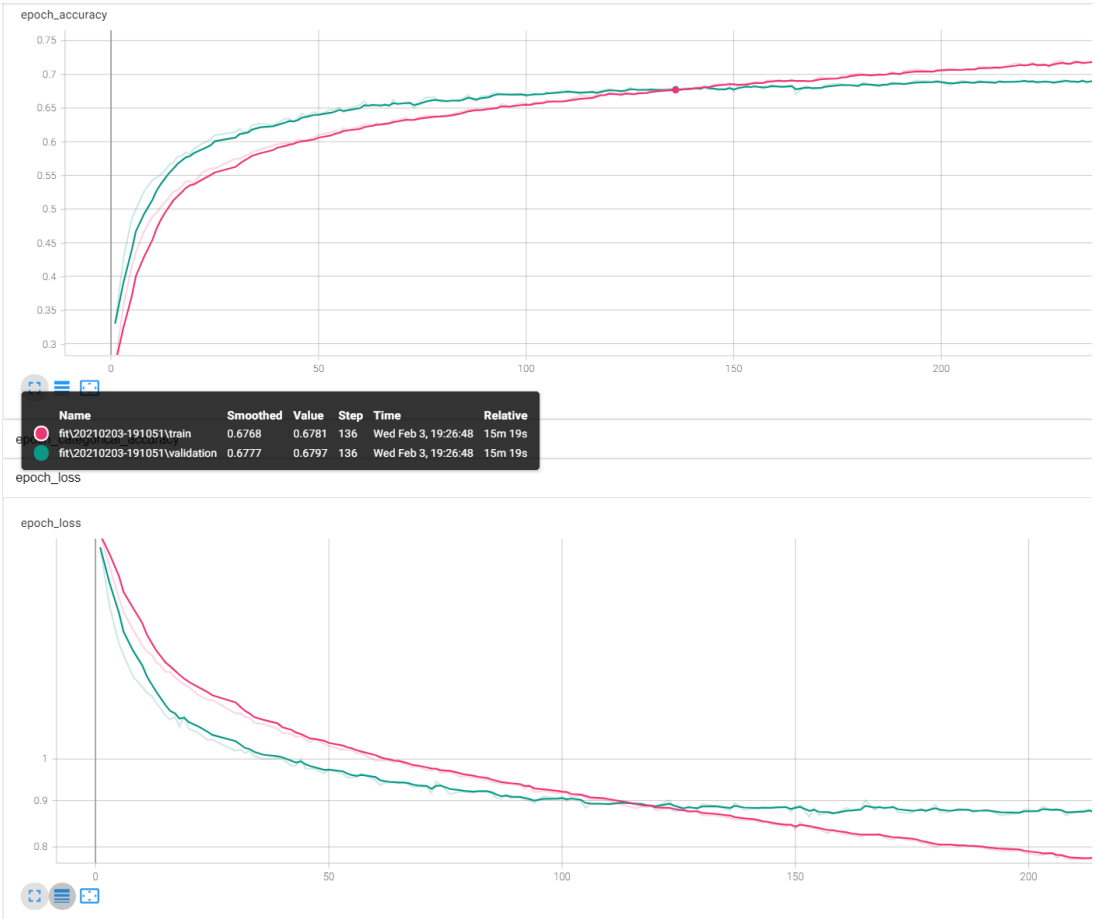


Figura 6.29.- Gráficas generadas en Tensorboard modelo CNN 3

Respecto al procesamiento de los entrenamientos, la diferencia se vuelve abismal. En este caso, al añadir una capa de convolución más que en el modelo anterior, la carga operacional se incrementa considerablemente. Para la GPU, que emplea sus múltiples núcleos en paralelismo, el tiempo por época es de 4 segundos. Sin embargo, para el procesador, este incremento de las operaciones produce un impacto enorme. Pese a continuar al 100 % de su rendimiento, cada época se completa en 58 segundos. Para otros ejemplos, se añadió una capa más, esos ejemplos no fueron detallados ya que no aportaban un a mejora de precisión. En esos ejemplos, pudo observarse que cuanto más profunda era la estructura de la red, más se multiplicaban los tiempos de la CPU, resultando enormes frente a los tiempos que produce la GPU. Esta explicación fue desarrollada en la sección 5.6.1, donde se explican los motivos de estos efectos.

Los resultados para este modelo son verdaderamente buenos. Se alcanzó la cifra del 70 % de precisión para los datos de validación y se controló el overfitting. Este modelo no presentó sobre ajuste ya que ambas precisiones, validación y entrenamiento, presentan valores muy cercanos. Además, puede observarse que la función del error decrece durante muchas más épocas que en el resto de modelos. Al alcanzar aproximadamente las 150 épocas, la función de error se mantiene constante. Lo positivo de esto, es que el error no comienza a crecer de nuevo como ocurría en casos anteriores.

En la figura 6.31, se muestra una comparativa del error de validación del modelo CNN 2 frente a este modelo CNN 3. En la gráfica puede observarse cómo el error comienza a ascender para el modelo 2, mientras que se mantiene con una tendencia constante en el modelo 3.

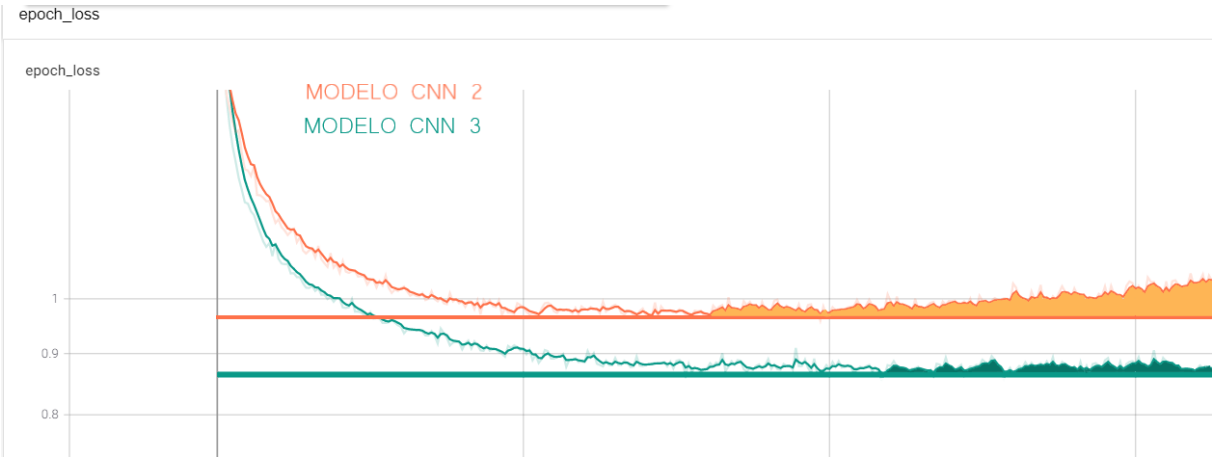


Figura 6.30.- Comparativa error de validación modelo 2 y 3

El área sombreada representa el incremento del error, para el modelo 3, el ruido puede parecer un aumento del error, pero a lo largo de las épocas, este vuelve al valor mínimo mostrando que se mantiene constante.

Para la función de error del modelo 2, el ruido también está presente, sin embargo, el valor crece. Una vez que alcanzó el mínimo, la función comienza a crecer. Si el entrenamiento se prolongara durante más épocas el efecto sería mucho más notorio.

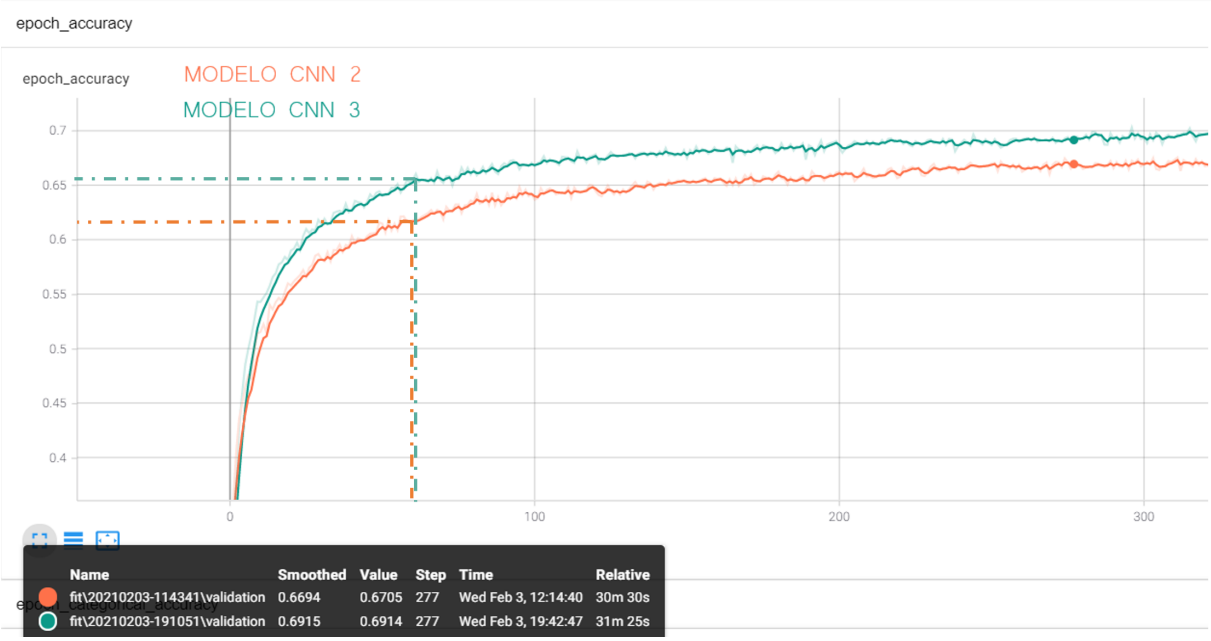


Figura 6.31.- Comparativa precisión de validación modelo 2 y 3

La figura 6.31, muestra la comparativa de la precisión de validación para ambos modelos. Se observa claramente la mejora que este modelo aporta respecto al anterior. Analizando los valores puede aparentar que no es una diferencia importante, pero en términos de optimización, lo es. Para un ingeniero de datos lograr estas modificaciones suponen muchas pruebas, ya que las tendencias son, en muchas ocasiones, sutiles y difíciles de analizar.

En la figura 6.32, se representa la matriz de confusión para el modelo. Los resultados son muy positivos, ya que se observa un incremento de precisión para todas las categorías. Emociones como “feliz” o “sorpresa” muestran un precisión excelente, del 89 % y 86 % respectivamente. Para el resto de categorías se obtienen buenos valores exceptuando la emoción del “miedo”. Tras analizar la causa de estas confusiones, empieza a observarse una tendencia clara, el modelo presenta “inteligencia”. Su análisis comienza a encontrar semejanzas entre emociones, que en la base de datos se plantean de forma totalmente independiente.

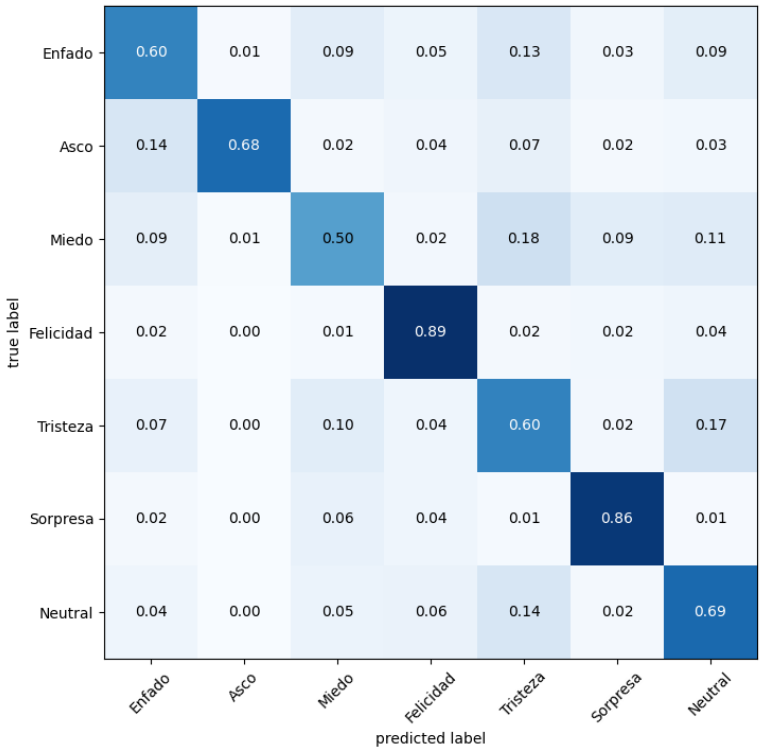


Figura 6.32.- Matriz de confusión modelo CNN 3

6.2.3.2.- Conclusiones

Este modelo ha logrado un gran éxito en comparación a todos los anteriores. Para lograr estas cifras y alcanzar la precisión de 70 % se han probado muchas estructuras. Se describe este caso ya que fue el más exitoso y característico. Para alcanzar este modelo, se trabajó con muchas más variantes anteriormente, del orden de 20 a 30 configuraciones diferentes. Tras numerosos ensayos y pruebas, las mejoras que se observan en los otros, eran demasiado bajas.

Una precisión del 5 % supone una gran diferencia cuando se efectúan estos procesos de optimización. Tras probar muchas modificaciones, se incluye esta versión como recopilación de las anteriores menos relevantes. Alcanzada la precisión del 70 %, puede considerarse que la base de datos ya presenta una limitación para incrementar el éxito. Como ya fue mencionado anteriormente, la función que calcula el error es

"categorical_crossentropy_loss". La optimización de un modelo que emplea esta función, se efectúa reajustando las categorías que presentaron más confusión. Las imágenes frontera, es decir aquellas que representan una mezcla de emociones, son interpretadas por el modelo como tal. Sus resultados para dichos casos, pueden ser por ejemplo: 60 % tristeza y 40 % neutral. El modelo está siendo inteligente, mostrando dudas ante una imagen frontera, sin embargo, al tratarse de una base de datos donde solo se espera una solución, esto es penalizado. La función de error castiga a la rama de la estructura que pretendía aportar dudas de una imagen que aparentaba neutralidad y tristeza al mismo tiempo. Sin embargo, estas dudas son reales, cualquier humano interpretando esa imagen estaría de acuerdo.

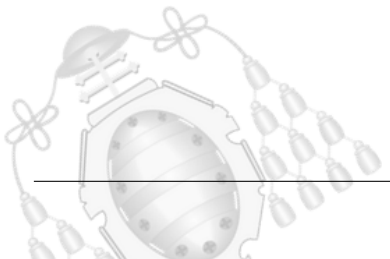
Estas conclusiones relacionadas con la eficiencia máxima y la limitación de la base de datos, serán detalladas en las conclusiones globales.

6.3.- Modelos con reducción de categorías

Para comprobar la influencia de la base de datos sobre la eficiencia, se efectuarán unas pruebas alterando las categorías presentes. Como ya se introdujo en los análisis anteriores, algunas emociones se encuentran en un punto frontera difícil de clasificar. Esto supone dudas para el modelo en el proceso de inferencia. Mediante la función Softmax, la red neuronal proporciona la probabilidad asociada a cada categoría. Es decir, muestra qué porcentaje de cada emoción encuentra en una imagen. Como la base de datos describe únicamente una etiqueta solución, estos intentos de mostrar varias emociones en cada imagen son rechazados y castigados por la función de error.

Para los siguientes modelos, se aplicaron cambios en la base de datos en cuanto a las categorías usadas se refiere. El objetivo es ver cómo afectan estas variantes a las predicciones. Si el modelo encuentra, por ejemplo, la emoción de "tristeza" muy relacionada con el miedo, se observarán mejoras al eliminar una de ellas.

Se empleará para todos los casos el modelo CNN 3 que ya presentaba las características más eficientes. Su estructura se representa en la tabla 6.10. Serán necesarias algunas modificaciones en la base de datos, así como cambios en las etiquetas. También las matrices de confusión y el programa de prueba se construyen de formas distintas. Como estas modificaciones son únicamente cuestiones de formato, no serán comentadas. El enfoque de esta sección es únicamente estudiar la eficiencia que provocan estos cambios y compararlos con los resultados previos.



	Operaciones	Número	Tamaño
PRE PROCESAMIENTO			
Capa 1	Convolución	32	Filtro(3,3)
	Dropout	-	0,25
Capa 2	Convolución	64	Filtro(3,3)
	MaxPooling	-	(2,2)
	Dropout	-	0,25
Capa 3	Convolución	128	Filtro(3,3)
	MaxPooling	-	(2,2)
	Dropout	-	0,25
Capa 4	Convolución	128	Filtro(3,3)
	MaxPooling	-	(2,2)
	Dropout	-	0,25
Flatten			
Capa 1	Neuronas 2048	Activación ReLU	
	Dropout	0,5	
Capa 2	Neuronas 1024	Activación ReLU	
	Dropout	0,5	
Capa output	Neuronas 7	Activación Softmax	

Tabla 6.10.- Estructura del modelo para las alteraciones del dataset

6.3.1.- Modelo 6 emociones: eliminando “asco”

La primera prueba que se efectúa es eliminar la categoría “asco”. Inicialmente podía observarse que estaba desbalanceada respecto a las demás categorías. Eliminando esta emoción del conjunto de datos de entrenamiento y de validación, se observará el efecto que desencadena. La distribución de la base de datos se muestra en la figura 6.35.

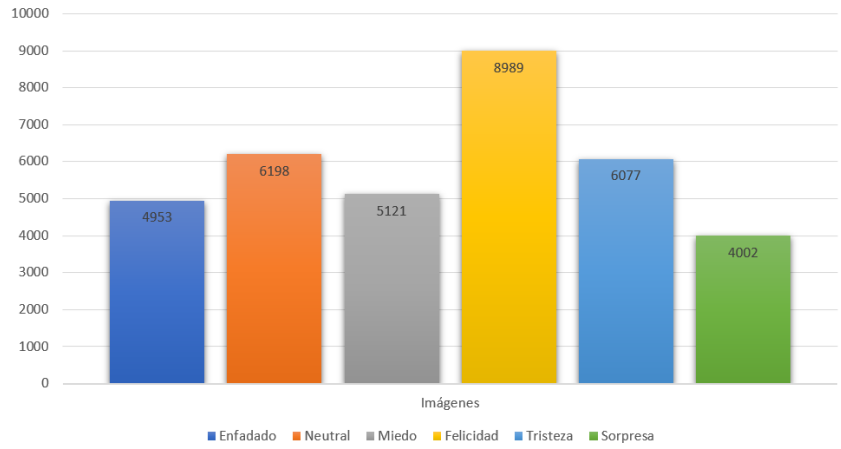


Figura 6.33.- Dataset sin la emoción “asco”

6.3.1.1.- Resultados obtenidos

Conjunto de datos	Precisión máxima	Época
Validación	0,702	303
Entrenamiento	0,7422	303

Tabla 6.11.- Resultados obtenidos sin la categoría “asco”

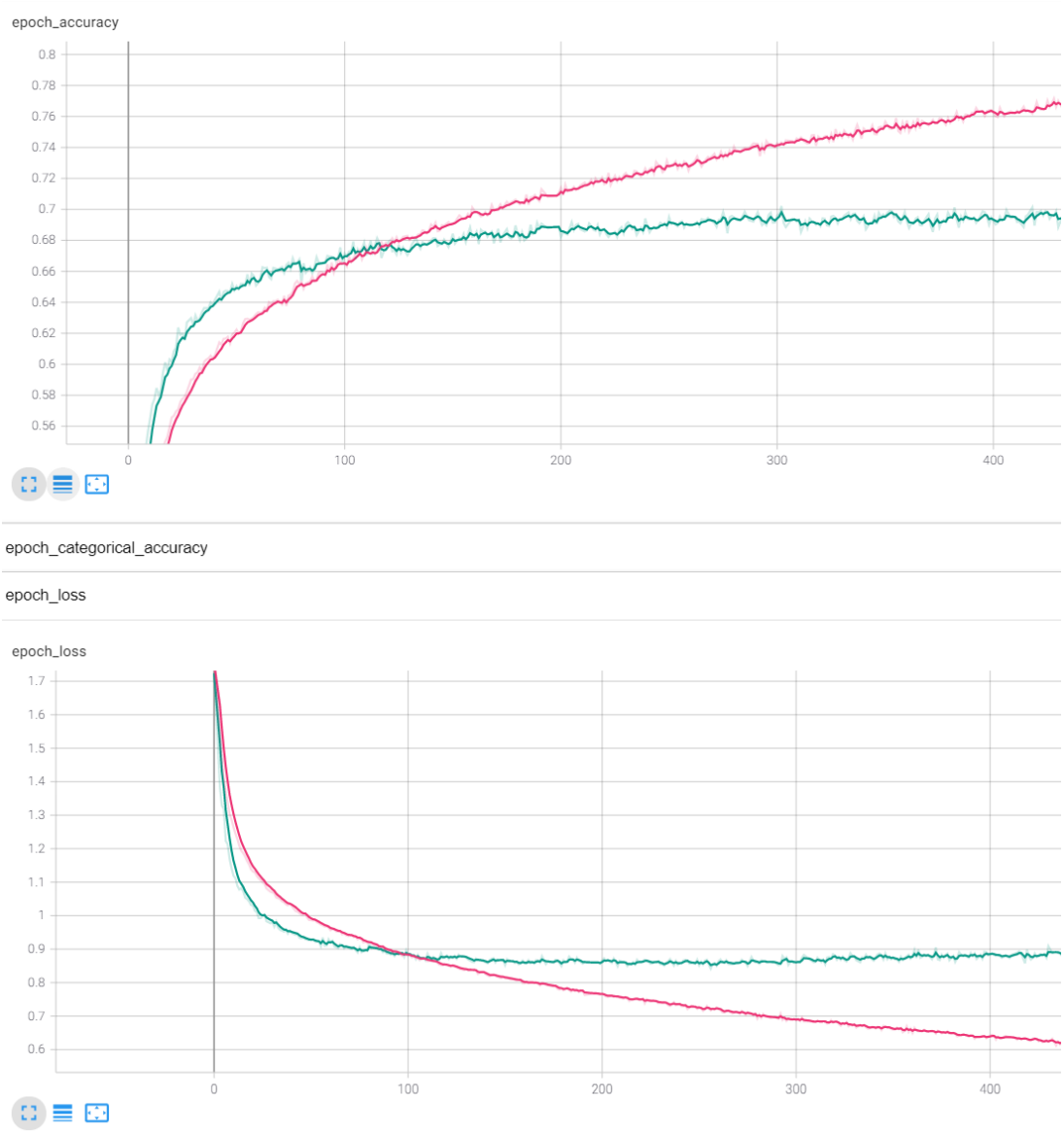


Figura 6.34.- Gráficas generadas en Tensorboard modelo sin “asco”

Los resultados obtenidos, son muy semejantes a los modelos anteriores. Eliminar esta categoría, no aportó apenas diferencias en la precisión. Si se continuara el entrenamiento, tendería a los valores obtenidos en el último modelo. Como se observa en las gráficas de la figura 6.34, las tendencias son correctas, pero se esperaba un incremento de la precisión.

6.3.1.2.- Conclusiones

Rompiendo las expectativas, el modelo no ha aportado ninguna mejora. Se incluye este ejemplo en el análisis debido al interés que supone analizar el desbalanceo que producía en la base de datos. La teoría indica que tener pocas muestras de una categoría produce efectos negativos, pero para este caso, no se cumple. Unas 547 imágenes representaban la emoción de "asco", suponían tan solo un 1,52 % de los datos totales. La explicación de este efecto está en que las pocas imágenes que describían esta categoría eran bastante claras. Además, estudiando la matriz de confusión del modelo CNN 3, se observa que únicamente aportaba confusiones importantes con la emoción "enfadado". Al existir pocas entradas para validar las caras de "asco", apenas causaba efectos ni positivos ni negativos en el modelo. Esto es un ejemplo interesante, representa un logro para el modelo, ha podido mantener dentro de la media una categoría con tan poca representación en el conjunto de datos. Esto muestra que el modelo es inteligente, adquirió la capacidad de reconocer esta emoción con muy pocas imágenes en comparación al resto.

6.3.2.- Modelo 6 emociones: eliminando “miedo”

La siguiente emoción que se va a estudiar es la del “miedo”. Esta categoría, representa bastantes confusiones ya que los efectos que podríamos catalogar como miedo, en el dataset están más representados por sorpresa. Las imágenes que se etiquetan como ”miedo” corresponden a rostros que muestran también ápices de tristeza, y por ello, suponen una gran confusión. En la matriz de confusión del modelo completo, 6.32, se pueden apreciar estas tendencias. En el **18 %** de las ocasiones, el modelo confunde la imagen de miedo con tristeza. Los rostros que aparentemente se visualizan con un miedo sutil, son un conflicto para el modelo, pues con bastante lógica, el modelo los intenta catalogar como tristeza o incluso neutralidad al considerarlos poco característicos. También sufre confusiones más bajas para los rostros de sorpresa, una confusión lógica dado que en muchas imágenes que representan miedo, existe un factor de sorpresa.

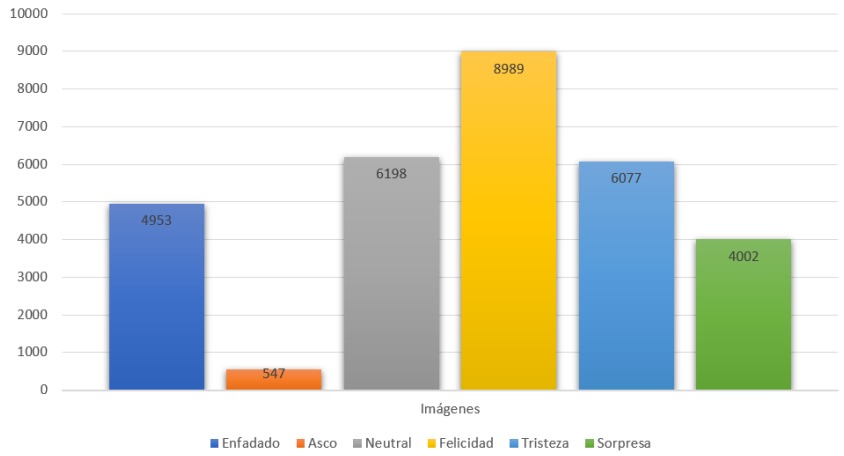


Figura 6.35.- Dataset sin la emoción “miedo”

6.3.2.1.- Resultados obtenidos

Conjunto de datos	Precisión máxima	Época
Validación	0,7634	280
Entrenamiento	0,7974	280

Tabla 6.12.- Resultados obtenidos sin la categoría “miedo”

Para el caso de eliminar el miedo del modelo, los efectos son muy positivos. El incremento de la precisión es de aproximadamente un **6 %**. Es una cifra muy elevada, retirar esta etiqueta, elimina muchos conflictos de clasificación para el modelo, otorgándole mayor eficiencia en esas categorías conflicto que ya fueron mencionadas.

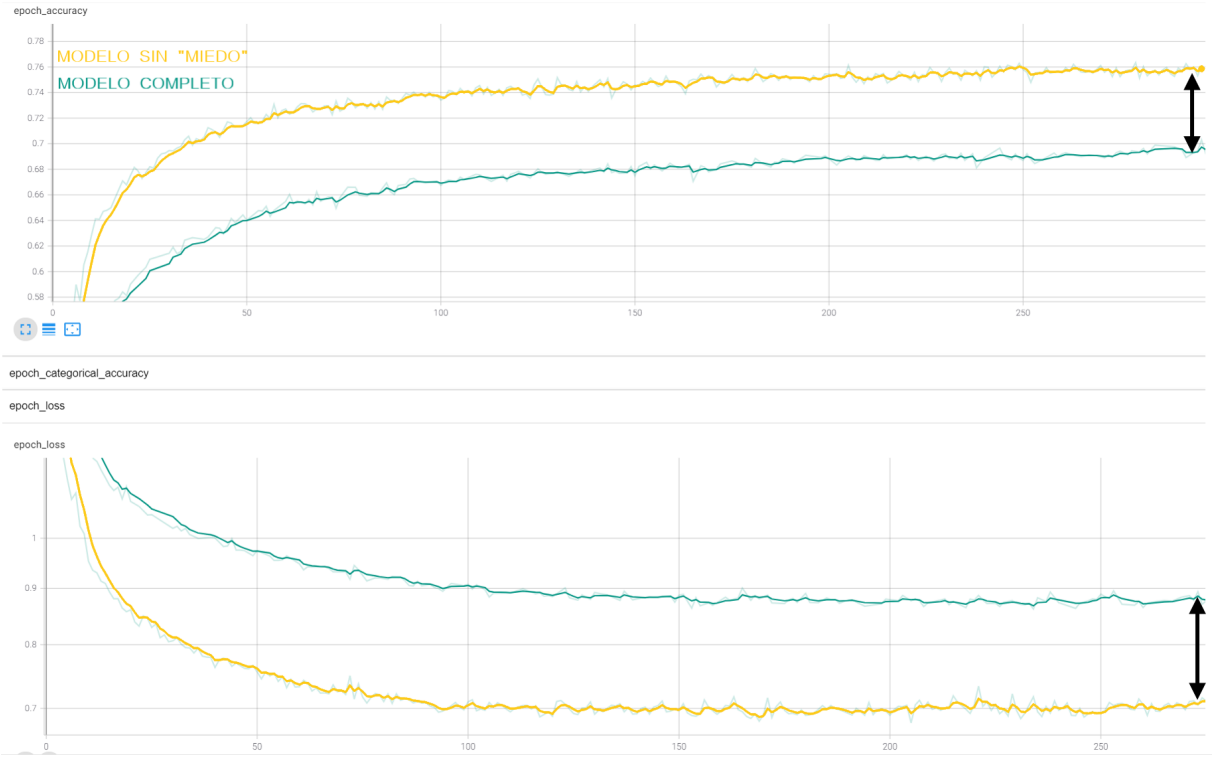


Figura 6.36.- Comparativa gráficas de validación CNN 3 completo y sin “miedo”

Las figuras 6.36, representan las gráficas del modelo CNN 3 con todas las etiquetas y el mismo modelo para seis etiquetas. La mejora es evidente y las curvas mantienen unas tendencias correctas. En la gráfica de precisión de validación, se observa esa clara diferencia desde las primeras épocas, manteniendo la precisión en crecimiento. Para la función del error, también se observa la mejora respecto al modelo anterior. Además, también logra mantenerse constante tras alcanzar el mínimo global.

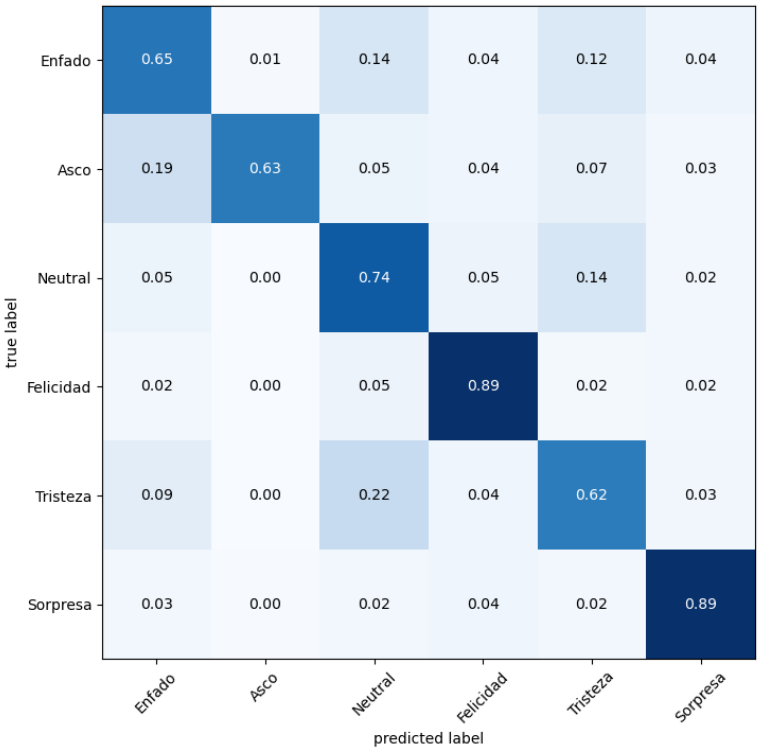


Figura 6.37.- Matriz de confusión modelo sin "miedo"

La matriz de confusión representada en la figura 6.37 representa unos datos muy buenos. De forma general, asciende la precisión para todas las categorías. La categoría "miedo" no aporta mucha utilidad dado que dentro de la emoción de "sorpresa", se contemplan rostros que reaccionan ante algo inesperado. Las etiquetas de "miedo" es muy abstracta y ambigua, pues como ya se mencionó, todos los rostros sorprendidos tienen un ápice de miedo. También añade la dificultad de diferenciar algunos rostros que se encuentran en un punto intermedio entre miedo y tristeza. Eliminando la etiqueta de "miedo", se ve incrementada la eficiencia de estas categorías.

6.3.2.2.- Conclusiones

La mejora del modelo es muy elevada respecto a la modificación anterior. Incrementar un *6 %* la precisión quitando únicamente una emoción, es una cifra muy buena. Esto implicaba que la categoría presentaba muchas imágenes difusas, que dificultaban la

asignación correcta. Estos resultados, dan mayor peso a la hipótesis de que la base de datos presenta limitaciones que impiden al modelo prosperar. Conseguir estas mejoras retirando categorías implica que verdaderamente, el modelo se veía comprometido para dar una única respuesta. Su formato de salida, la función Softmax, aporta un formato diferente al que la base de datos proporciona. Surge aquí la hipótesis de sí el modelo está mostrando mayor capacidad de la que se pretendía.

En el diseño de la base de datos, se decantaron por etiquetar cada imagen con una única emoción, en el formato denominado "one-hot-vector". Como ya fue mencionado, esto implica que el resultado que desea recibir la función "categorical_crossentropy_loss", sea con un 100 % de seguridad. El modelo penaliza todas las respuestas que aporten algún grado de duda, pese a ser correctas. Estas conclusiones e hipótesis, serán más detalladas en las conclusiones finales.

6.3.3.- Modelo 4 emociones

Como prueba final, para observar los incrementos de las precisiones, se construye un modelo con las cuatro emociones más características. Las emociones que se utilizarán son "enfado", "sorpresa", "neutral" y "felicidad". La base de datos tendrá la distribución mostrada en la figura 6.38.

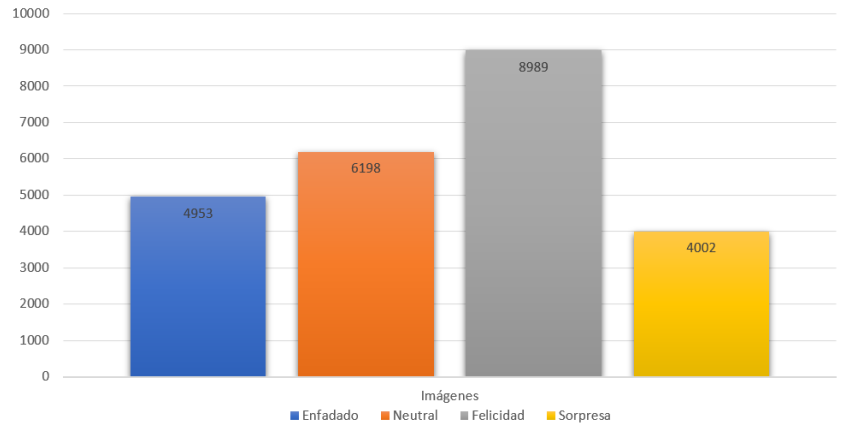


Figura 6.38.- Dataset para 4 emociones"

6.3.3.1.- Resultados obtenidos

Conjunto de datos	Precisión máxima	Época
Validación	0,853	277
Entrenamiento	0,869	277

Tabla 6.13.- Resultados obtenidos para 4 emociones

Los resultados para 4 emociones son muy positivos. Al eliminarse las categorías más difusas y mantenerse las más deterministas, la mejora es enorme. Estas 4 emociones, son quizás las menos ambiguas dentro del conjunto de datos. Cualquier persona que las interprete, puede dar un resultado objetivo, sin generarse dudas. Este formato de solución, donde la respuesta es objetiva, se observa claramente en este ejemplo.

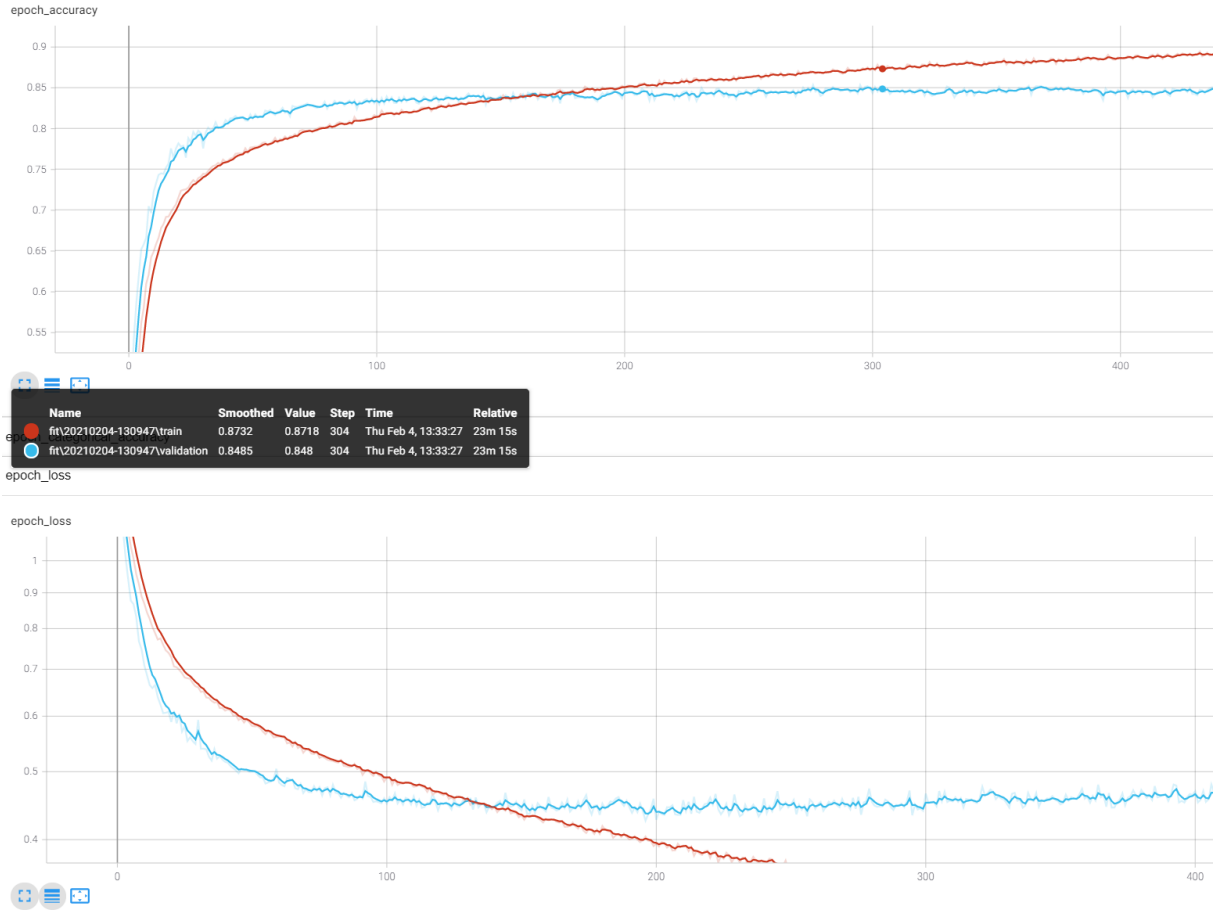


Figura 6.39.- Gráficas de entrenamiento y validación para el modelo con 4 emociones

La figura 6.39, muestra los valores obtenidos para este modelo. En color naranja, se representan las gráficas referentes a los datos de entrenamiento, mientras que en color azul, las referentes a los datos de validación. Puede observarse como alcanza un valor de pico del 85 %, manteniéndose constante tras esa época. Si se continúa el aprendizaje, tiende al sobre entrenamiento. En el punto de máxima precisión, correspondiente a la época 277, los valores tanto para validación como pruebas, son prácticamente iguales. El modelo presenta características muy buenas en todos los aspectos, a excepción de la tendencia al sobre ajuste, que comienza a ser clara a partir la época 300.

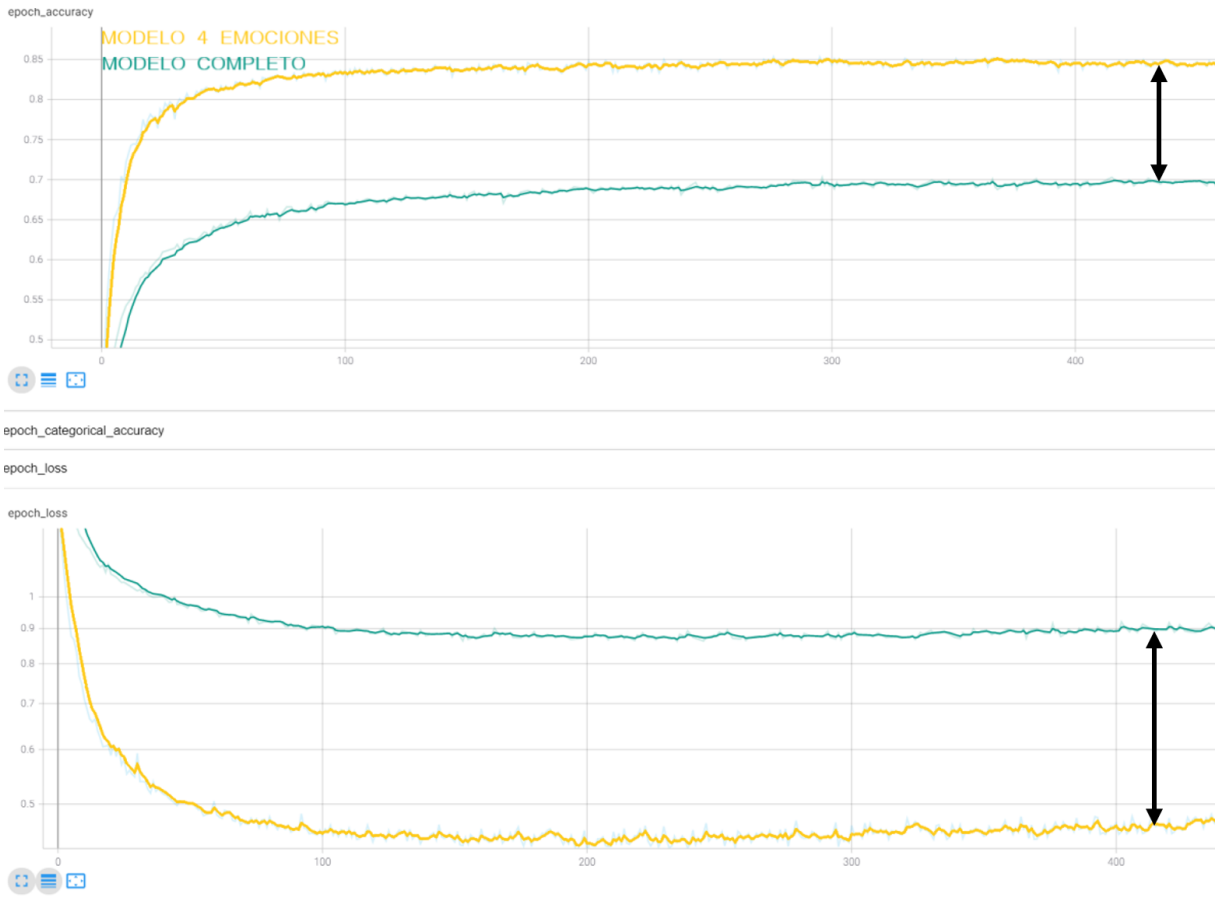


Figura 6.40.- Gráficas comparativas entre el modelo CNN 3 completo y con 4 emociones

En la figura 6.40, se muestra una comparativa del modelo CNN 3 con 7 categorías frente a 4. En ella se aprecia de forma clara la mejora en cuanto a precisión.

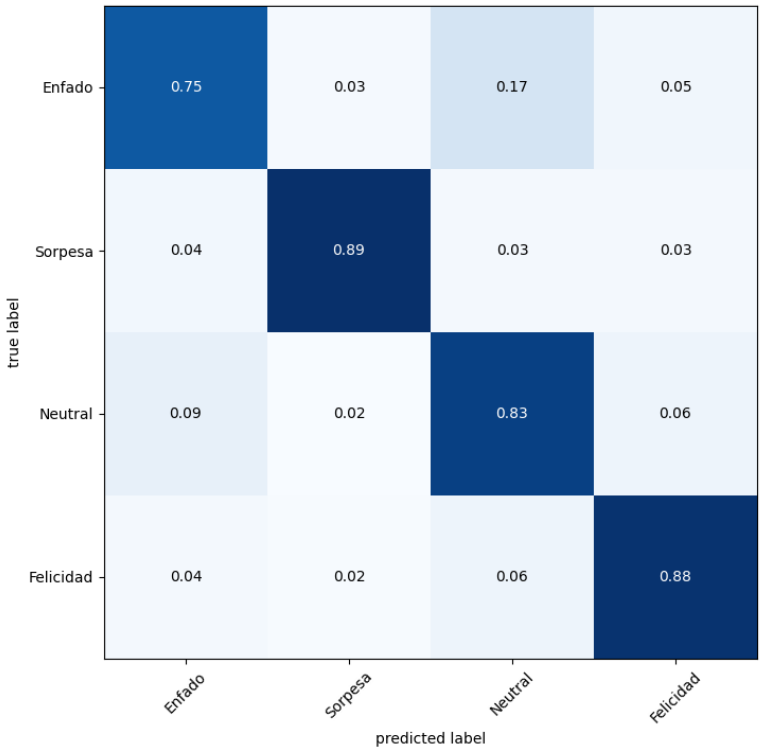


Figura 6.41.- Matriz de confusión 4 emociones

En la matriz de confusión, representada en la figura 6.41, se observan las altas precisiones para todas las categorías. Ha de tenerse en cuenta, que se trata de un proyecto de interpretación de imágenes donde además, las soluciones son un concepto algo abstracto como son las emociones. Los resultados no pueden compararse con otras predicciones sobre datos mucho más objetivos y claros. Dado el formato de los datos del proyecto, las inexactitudes del dataset y el cierto grado de ambigüedad de una emoción, un **85 %** de precisión es un valor muy elevado.

6.3.3.2.- Conclusiones

El modelo ha mejorado de forma abismal al reducir las categorías a 4. Es cierto que el mero hecho de reducir el número de respuestas, aporta mejoras. Sin embargo, los cambios tan drásticos se deben a más factores. La eficiencia para estas categorías, manifiesta que ante imágenes claras y con respuestas objetivas, el modelo muestra una gran inteligencia.

El formato de la solución que aporta la base de datos, no es óptimo ya que limita la interpretación de las imágenes. Al igual que una persona, cuando clasifica un rostro, puede aportar diferentes datos sobre la imagen. De un rostro con un sonrisa sutil, un humano podrá considerar que la imagen se encuentra entre feliz y neutral, decantándose finalmente por la primera opción. La base de datos, al estar en formato "one-hot-vector", no permite estos análisis llegando incluso a penalizarlos.

6.4.- Comparativa global

Como análisis global se observó una tendencia muy clara, la base de datos subestima la capacidad de la red neuronal. Se apreciaba en los primeros ejemplos, un crecimiento del error acompañado de una precisión de validación constante. Ese efecto, no comprendido en su momento, correspondía a la inteligencia del modelo. La red intentaba mejorar pese a las penalizaciones que le impartía la función de error. Incluso en algunos casos, el modelo aumentaba su precisión y el error, en lugar de disminuir, aumentaba también. Este efecto se asocia a más respuestas correctas pero con menor "confianza".

Se adelantaba en esas secciones, que la función de error no era inversamente proporcional a la de precisión. Las gráficas de precisión, muestran la media de aciertos, es decir, los aciertos totales en función de todas las pruebas ejecutadas. Sin embargo, además del acierto, el error también mide la confianza con la que se consiguió. Esto implica que un resultado como (1,0,0,0,0,0,0), produce un error 0. Para el mismo caso, la respuesta (0.5,0.4,0.1,0,0,0,0), produce un error muy grande para la categoría dos, ya que pretendió aportar una proporción que no le correspondía.

Cuando comienzan los entrenamientos, la red neuronal verdaderamente aprende a analizar las imágenes con cierto rasgo humano. De todo ese aprendizaje, desarrolla la tendencia más humana posible: ¡encuentra en una imagen rasgos de distintas emociones!. Es decir, comienza a mostrar las dudas que mostraría una persona al analizar las mismas fotografías. Si la base de datos se hubiera construido en otro formato, la CNN tendría un enorme potencial. La base de datos podría aportar las soluciones de cada imagen con varias

etiquetas posibles a la vez o con un formato proporcional. De esta forma, una imagen que muestre una felicidad sutil, podría haberse etiquetado como (0.80,0.20,0,0,0,0,0).

Con el formato "one-hot", el modelo lucha en cada entrenamiento para mantener sus proporciones. Es incapaz de asumir que en cada rostro se observa una emoción con un 100 % de determinación. Esta categorización es útil cuando se trata de averiguar algo más objetivo. Por ejemplo, si se pretende analizar un animal para saber si es un perro o un gato, en este caso existe una realidad muy obvia detrás de las soluciones, que tras penalizaciones, el modelo terminará encontrando.

En algunos casos, como ya se mencionó, el modelo consigue más eficiencia con un aumento del error. Metafóricamente, en esas situaciones, el modelo consigue esquivar las penalizaciones que le impone el error y logra mayores aciertos. Todo este flujo de correcciones y respuestas termina estabilizándose, adquiriendo un valor constante, donde el modelo ya no puede mejorar más. Pese a mucho entrenamiento, el modelo es incapaz de ver un 100 % de felicidad en una imagen donde solo hay una ligera sonrisa, porque es ir en contra de lo que ha aprendido.

Se proyecta en la siguiente figura 6.42 la precisión de validación para todos los modelos explicados en este capítulo. En ella, se aprecia la evolución gradual que ha experimentado el modelo a lo largo de todas las modificaciones comentadas.

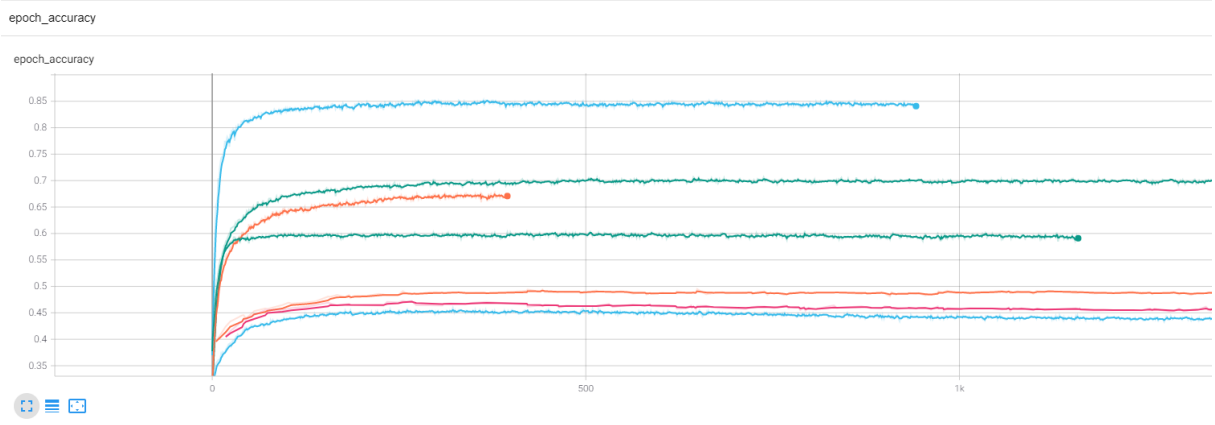


Figura 6.42.- Agrupación de las gráficas correspondientes a la precisión de validación

Finalmente, para mostrar todos los entrenamientos realizados a lo largo del proyecto, se plasman todas las gráficas existentes en la figura 6.44. Con esta figura, se pretende mostrar la cantidad de análisis que se pueden realizar de un mismo proyecto en búsqueda de su optimización. La clave del trabajo del ingeniero de datos, está en todas esas imágenes. Se realizaron aproximadamente 300 entrenamientos a lo largo del proyecto.

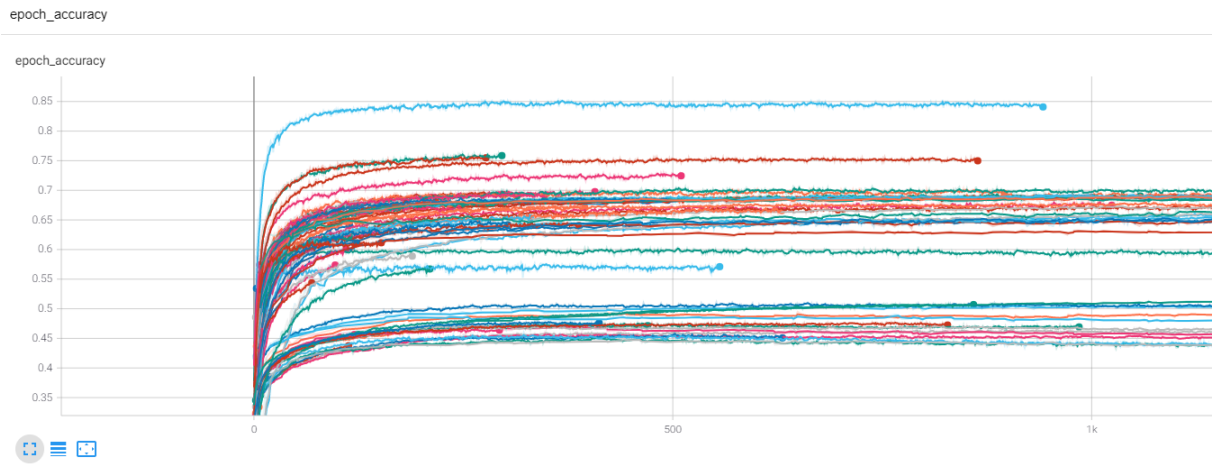


Figura 6.43.- Agrupación de las gráficas correspondientes a la precisión de validación de todas las pruebas efectuadas

6.5.- Consumo del modelo

En esta sección, se estudian los tiempos de ejecución del mejor modelo en diferentes dispositivos. Se pretende observar el comportamiento de los dispositivos al ejecutar inferencias masivas, pudiendo efectuarse así una comparativa. El objetivo del modelo, en su uso real, será efectuar pocas inferencias por segundo. Para mostrar las emociones en tiempo real, la Raspberry Pi 4, ofrece una buena experiencia de uso. La capacidad que proporcionará un PC será superior, pero no resulta necesaria. La portabilidad y coste reducido de la Raspberry le otorgan muchos puntos a favor que no dispone un PC. Al mirar a la cámara, la Raspberry nos muestra cada segundo la emoción que presenta el usuario. Si se desea construir un sistema que realice una gran cantidad de inferencias por segundo, es evidente que la Raspberry Pi no es el dispositivo indicado para ello.

Se efectuará le evaluación del conjunto de prueba y del conjunto de entrenamiento. Además, se realizará una validación cruzada, es decir, se empleará todo el conjunto de datos. Se observarán los tiempos de ejecución que emplea cada dispositivo. Si por su estructura general y su coste, ya podía intuirse la gran diferencia entre ellos, se efectuó una comparativa real. Mediante el sitio web Cpu-Monkey [64], se generaron las siguientes imágenes:

6.5.1.- Comparativa de las CPUs

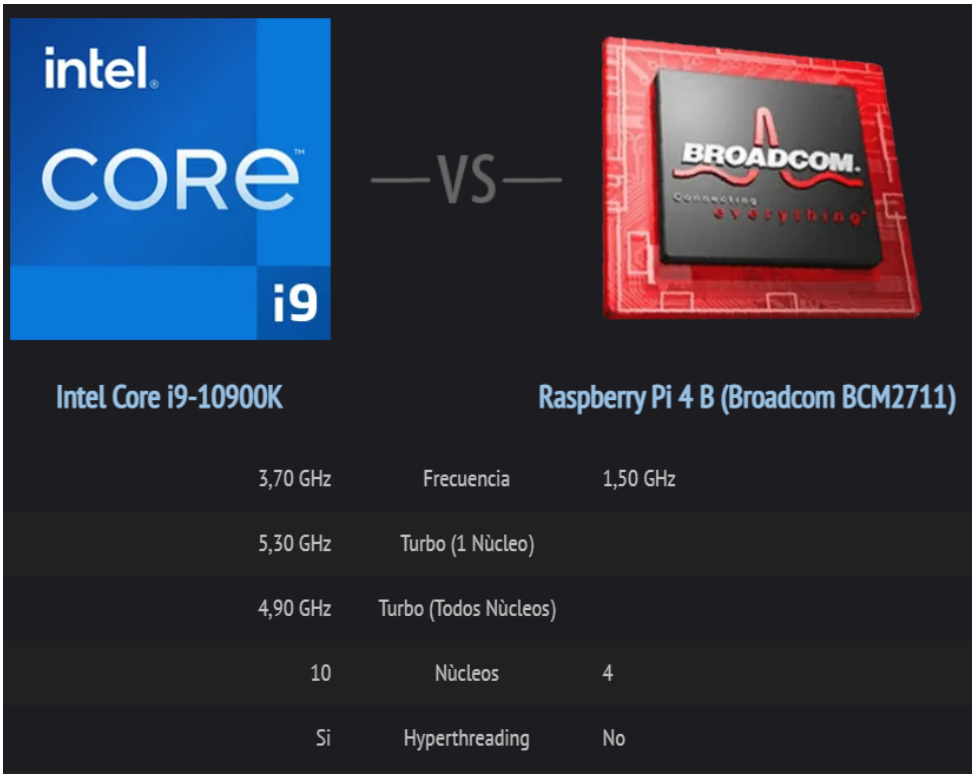


Figura 6.44.- Comparativa de especificaciones del intel i9 10900k y el Broadcom BCM2711

En cuanto a las especificaciones, la diferencia es importante. Desde el número de núcleos a las frecuencias de operación de estos. En cuanto a efectuar las inferencias, la tarjeta gráfica NVIDIA, también puede realizarlas mediante CUDA. Sin embargo, como la carga computacional es mucho menor que la del entrenamiento, también el Intel i9, efectuará la inferencia de forma rápida, tal y como se verá más adelante. Aún así, la GPU continúa siendo más rápida en este tipo de operaciones.

Con esto, se pretende indicar que la Raspberry Pi no presenta herramientas CUDA de aceleración, por lo que empleará su CPU, que es mucho menos potente que la de un PC estándar. Se destacaron tres comparativas benchmark que resultaban de interés para esta tarea.

La figura 6.45 muestra la prueba Geekbench 5 (Single-Core). En esta, se utiliza intensivamente la memoria del sistema, involucrando únicamente un núcleo.

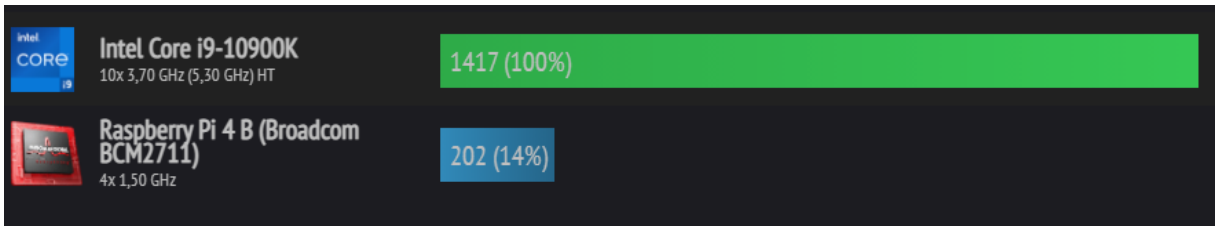


Figura 6.45.- Comparativa Geekbench 5, 64bit (Single-Core)

La figura 6.46 muestra la prueba Geekbench 5 (Multi-Core). El procedimiento es similar a la prueba anterior pero empleando todo los núcleos y el hyperthreading.

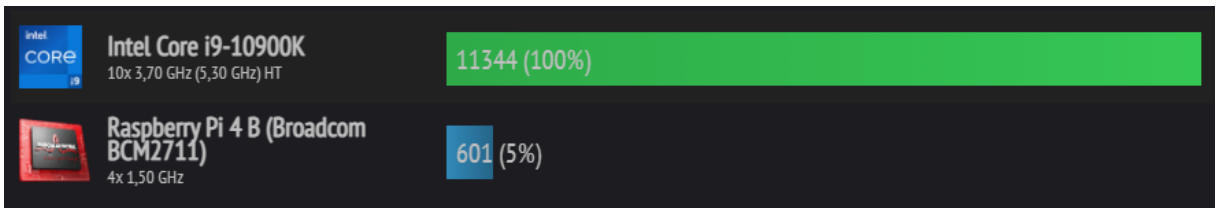


Figura 6.46.- Comparativa Geekbench 5, 64bit (Multi-Core)

La figura 6.47 muestra cuántos mil millones de operaciones de punto flotante puede realizar por segundo la iGPU. Es el rendimiento de la unidad gráfica interna del procesador.

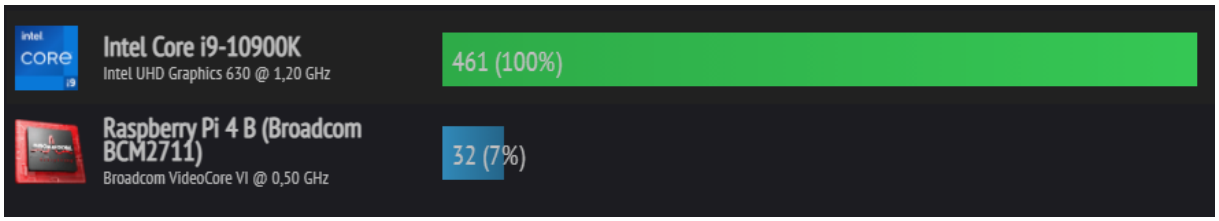


Figura 6.47.- Comparativa iGPU - Rendimiento FP32 (GFLOPS de precisión simple)

6.5.2.- Resultados mostrados en el cálculo de las inferencias

Se elaboró un código que efectúa la evaluación del conjunto de datos de pruebas y del conjunto de datos de entrenamiento. Además, efectúa una validación cruzada mezclando todas las imágenes del dataset. El programa, mide los tiempos de ejecución de cada tarea y los imprime por pantalla en segundos.

Para la Raspberry Pi 4 modelo B, con 4 GB de RAM. Se obtuvieron los resultados representados en la figura 6.48.

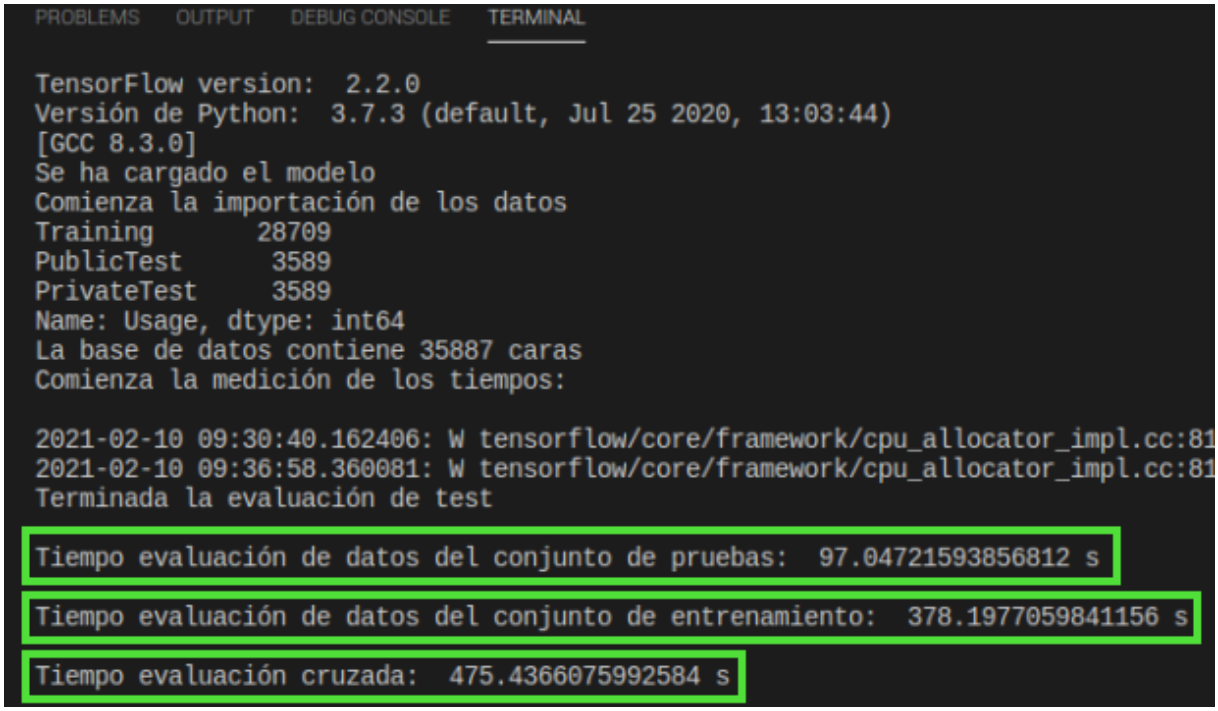


Figura 6.48.- Tiempos de ejecución en la Raspberry

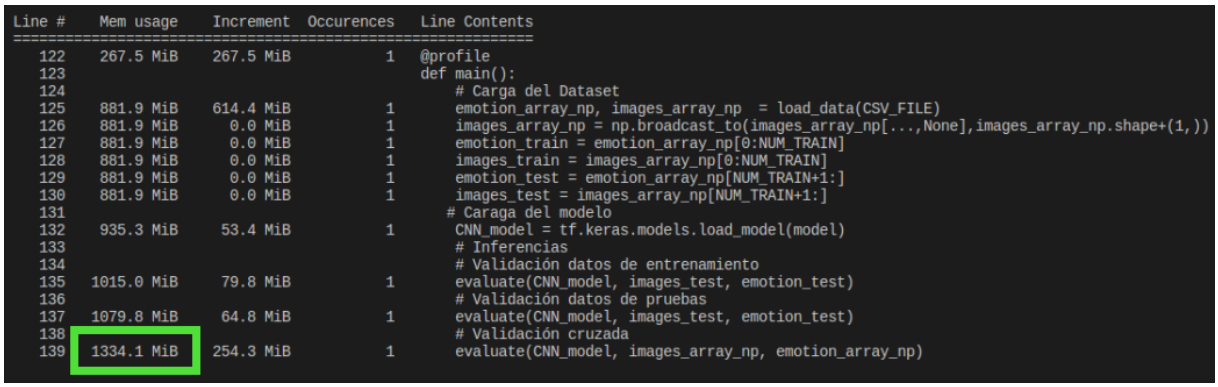


Figura 6.49.- Consumo de RAM en la Raspberry

En las figuras 6.48 y 6.49, se observa que, para evaluar el conjunto de pruebas, se tardó aproximadamente **1 minuto y 37 segundos**. Para el conjunto de entrenamiento, el tiempo total fue de **6 minutos y 18 segundos**. La validación cruzada se efectuó en **7 minutos y 55 segundos**. El consumo de memoriza RAM fue aproximadamente de **1334,1 MB**.

Para el PC empleando la CPU, Intel i9, se observaron los resultados mostrados en las figuras 6.50 y 6.51.

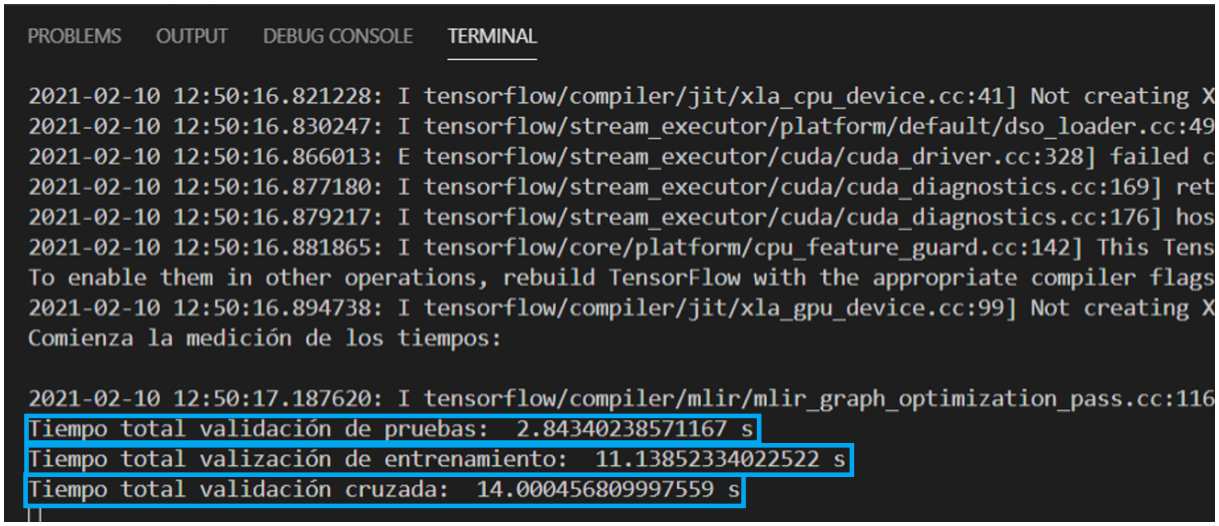


Figura 6.50.- Tiempos de ejecución en el PC mediante CPU

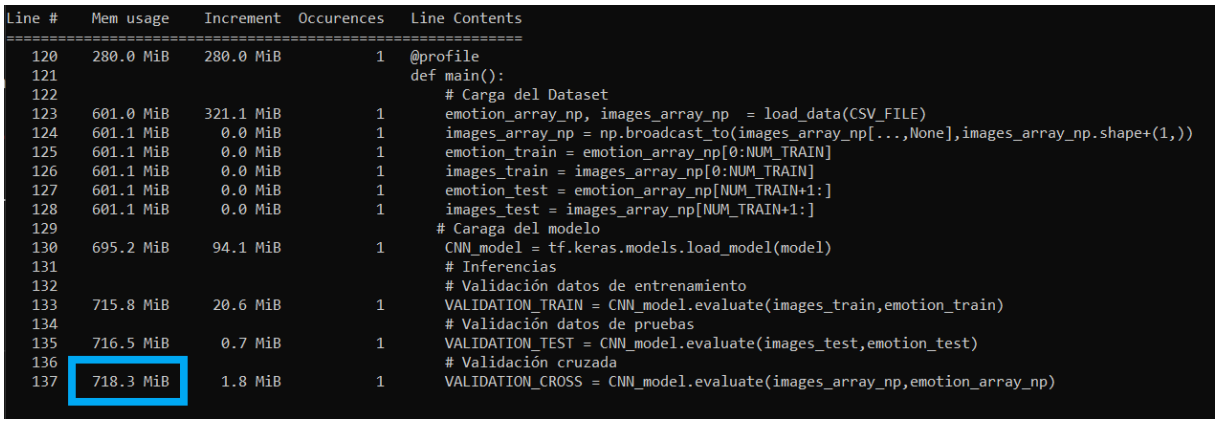


Figura 6.51.- Consumo de RAM en el PC mediante CPU

Los tiempos son muy superiores a los del procesador de la Raspberry, efecto esperado dada la enorme diferencia de capacidad y coste. El tiempo para evaluar el conjunto de datos de pruebas fue de **2,84 segundos**. Para el conjunto de entrenamiento, se efectuó en **11,13 segundos**. Finalmente la validación cruzada se efectuó en **14 segundos**. El consumo de RAM, sin tener en cuenta el consumo del programa (331,2MB), fue de aproximadamente de **718,3 MB**.

Para el PC, empleando las herramientas de CUDA y ejecutándose mediante GPU, se obtuvieron los resultados observados en las figuras 6.52 y 6.53.

```
2021-02-10 12:45:48.661906: I tensorflow/core/platform/windows/subprocess.cc:308] SubProce

2021-02-10 12:45:48.669203: I tensorflow/core/platform/windows/subprocess.cc:308] SubProce

2021-02-10 12:45:48.679051: I tensorflow/core/platform/windows/subprocess.cc:308] SubProce

2021-02-10 12:45:48.686279: I tensorflow/core/platform/windows/subprocess.cc:308] SubProce

2021-02-10 12:45:48.694655: I tensorflow/core/platform/windows/subprocess.cc:308] SubProce

Tiempo total validación de pruebas:  1.3281521797180176 s
Tiempo total valización de entrenamiento:  2.149329900741577 s
Tiempo total validación cruzada:  8.086166858673096 s
```

Figura 6.52.- Tiempos de ejecución en el PC mediante GPU

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
   120    279.9 MiB    279.9 MiB           1   @profile
   121                                         def main():
   122                                             # Carga del Dataset
   123    602.3 MiB    322.4 MiB           1       emotion_array_np, images_array_np  = load_data(CSV_FILE)
   124    602.4 MiB      0.0 MiB           1       images_array_np = np.broadcast_to(images_array_np[...,None],images_array_np.shape+(1,))
   125    602.4 MiB      0.0 MiB           1       emotion_train = emotion_array_np[0:NUM_TRAIN]
   126    602.4 MiB      0.0 MiB           1       images_train = images_array_np[0:NUM_TRAIN]
   127    602.4 MiB      0.0 MiB           1       emotion_test = emotion_array_np[NUM_TRAIN+1:]
   128    602.4 MiB      0.0 MiB           1       images_test = images_array_np[NUM_TRAIN+1:]
   129                                            # Caraga del modelo
   130   1050.2 MiB    447.8 MiB           1       CNN_model = tf.keras.models.load_model(model)
   131                                             # Inferencias
   132                                             # Validación datos de entrenamiento
   133   2652.8 MiB   1602.6 MiB           1       VALIDATION_TRAIN = CNN_model.evaluate(images_train,emotion_train)
   134                                             # Validación datos de pruebas
   135   2658.2 MiB      5.4 MiB           1       VALIDATION_TEST = CNN_model.evaluate(images_test,emotion_test)
   136                                             # Validación cruzada
   137   2662.7 MiB      4.5 MiB           1       VALIDATION_CROSS = CNN_model.evaluate(images_array_np,emotion_array_np)
```

Figura 6.53.- Consumo de RAM en el PC mediante GPU

Los tiempos mejoran aún más respecto al procesador. Al mismo tiempo, se ve cómo el cálculo en paralelo que efectúa la GPU aumenta considerablemente el consumo de RAM. Para efectuar la evaluación sobre los datos de prueba, se obtuvo un tiempo de **1,32 segundos**. Para el conjunto de entrenamiento, **2,14 segundos** mientras que para la validación cruzada fueron **8,08 segundos**. La memoria RAM que consumió este proceso fue **2662,7 MB**.

6.6.- Pruebas mediante videojuego

Para visualizar como sería el funcionamiento del panel en un simulador de conducción, se incorporó durante el uso de un videojuego. Durante todo el periodo de conducción, se analizaba la emoción que mostraba el conductor. Se tomaron fotografías en momentos críticos de la conducción con la información que mostraba el panel para esos instantes.

Mientras una persona conducía en el videojuego, el modelo era capaz de registrar diferentes emociones. Obviamente, al tratarse de un juego, las emociones no son tan marcadas como serían en un simulador real. Aún así, en momentos críticos, el panel mostraba información de interés. Elevando el volumen del juego y utilizando un volante con pedales, se dota al juego de mayor realismo. De esta forma, los efectos que produce su conducción, le producen una reacción mucho más significativa. Algunos de los resultados de interés, se muestran en las siguientes imágenes.

Figura 6.54.- Pantalla de conducción y panel para la emoción “neutral”

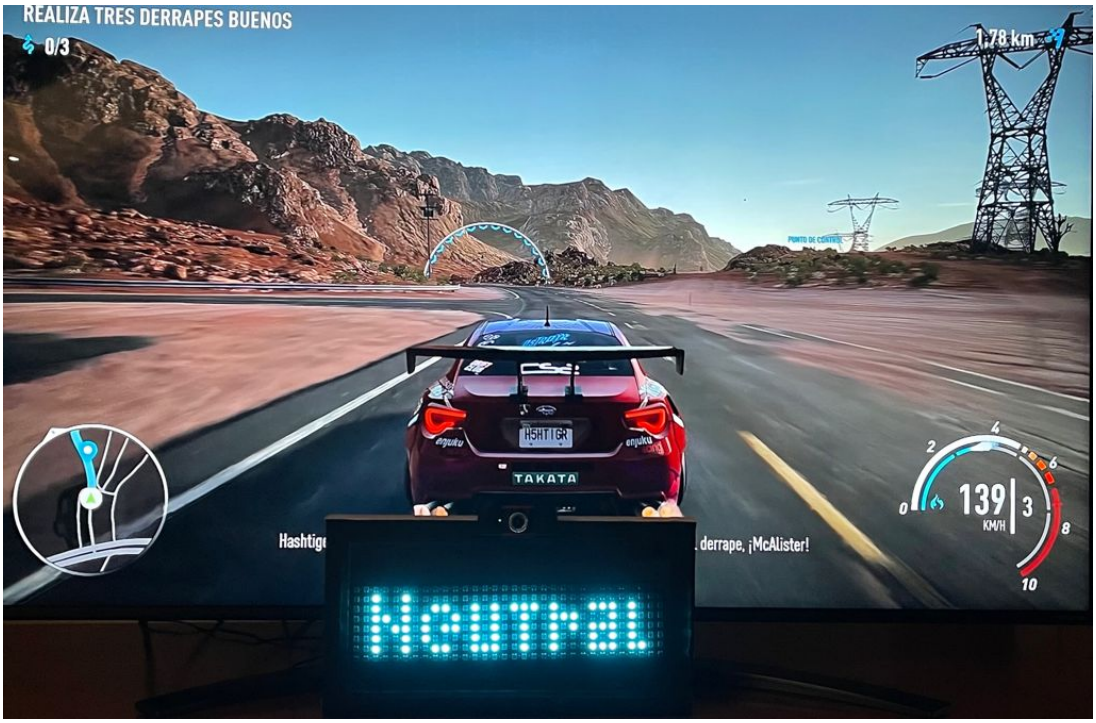


Figura 6.55.- Pantalla de conducción y panel para la emoción “neutral”

Figura 6.56.- Pantalla de conducción y panel para la emoción “sorpresa”

Figura 6.57.- Pantalla de conducción y panel para la emoción “sorpresa”

Figura 6.58.- Pantalla de conducción y panel para la emoción “enfadado”

Figura 6.59.- Pantalla de conducción y panel para la emoción “enfadado”

Figura 6.60.- Pantalla de conducción y panel para la emoción “feliz”

7. Conclusiones y trabajo futuro

7.1.- Conclusiones

Respecto a los objetivos planteados al comienzo del proyecto, el resultado es muy positivo. El alcance del trabajo ha cumplido con los objetivos planteados. La Inteligencia Artificial existente en los modelos creados, logra interpretar de forma correcta las emociones significativas que muestra el conductor. Este proyecto es un comienzo y sirve de prueba de que esta tecnología, tiene mucho futuro por delante. Los retos plateados para la precisión del modelo se lograron. Incluso la reducción de etiquetas mostró precisiones muy elevadas, que suponen algo muy positivo para este desarrollo. Además, se consiguió ejecutar las predicciones en el dispositivo de menor coste computacional, lo que representa otro objetivo cumplido.

Aplicando el trabajo futuro mencionado en la sección 7.2, estos modelos dotara a los simuladores de la funcionalidad que necesitan para desarrollar unos análisis más específicos. En este proyecto se planteo el enfoque del automóvil, pero esta línea de desarrollo, puede tener mucho futuro en otros campos como la aviación o la salud. Por ejemplo, detectar el aspecto de máscara, que es uno de los síntomas del Parkinson. El paciente muestra un rostro muy particular caracterizado por ser poco expresivo.

En cuanto a la realización del proyecto, el trabajo ha sido verdaderamente enriquecedor. Las áreas exploradas durante la elaboración, han aportado una gran experiencia para el autor. En los inicios, gestionar la Raspberry Pi, aportó conocimiento acerca de los protocolos y configuraciones estudiadas en el Grado. Conectarse al router de la compañía para abrir los puertos, aportarles un dominio global a los equipos o configurar protocolos como el SSH, han sido tareas interesantes. Además, se adquirió conocimiento sobre Visual Studio Code y sus extensiones para programar en dispositivos de forma remota.

Sincronizar los datos generados con la nube supuso una funcionalidad extra muy interesante. Los dos ordenadores lograban ejecutar entrenamientos al mismo tiempo y

volcaban la información a la nube. Configurar protocolos como WOL y llevarlos a nivel de Internet fue otro proceso muy curioso. Durante el Grado, siempre efectuamos prácticas sobre routers del laboratorio. Realizar configuraciones sobre los routers domésticos fue una oportunidad para aplicar conocimientos aprendidos en clase.

A nivel de la Inteligencia Artificial, el enriquecimiento ha sido descomunal. Se comenzó el proyecto desde un nivel nulo, ya que en el Grado de Telecomunicaciones no se trabaja con este tipo de programación. Es un logro haber conseguido comprender los algoritmos que se emplean, aprender sobre las APIs que los introducen y ser capaz de construir todas las estructuras. El aprendizaje ha sido un proceso muy largo y gradual. Leer mucho código de otros usuarios que trabajan con la librería, comprender los problemas de otras personas, aprender de las soluciones que aportan otros usuarios... es un trabajo día a día que nutre tu conocimiento. Poco a poco, comienzas incluso a poder resolver las dudas que publican otros usuarios.

En cuanto a la utilización de diferentes versiones de Python y APIs, se ha adquirido una gran experiencia. Trabajar con entornos de desarrollo mediante Anaconda, se ha vuelto algo rutinario y necesario. Es una herramienta que te permite combinar muchas versiones y permite efectuar numerosas pruebas. En cuanto a efectuar los entrenamientos de los modelos con la GPU, ha sido otro proceso muy complicado. Las configuraciones necesarias para que se sincronicen las librerías con la tarjeta gráfica son algo complejas. Llevó tiempo comprender su funcionamiento para lograr ponerlo en práctica. Además, no conforme con efectuar todas las configuraciones para la NVIDIA 2070, se hizo después para la 3070. Todas las versiones habían cambiado, Python, Tensorflow, CUDA todo debía ejecutarse en una versión superior. En ese punto, fue donde se puso a prueba el conocimiento, pues instalarlo todo fue infinitamente más rápido que la primera vez.

Respecto a la herramienta Tensorboard, ha sido un acierto aprender a utilizarla e incorporarla en el análisis del trabajo. Se encontraron numerosos proyectos relacionados, donde los usuarios apenas extraían información de sus modelos. Pretendían mejorar algo de lo que solo conocían tres datos finales. La visualización de los datos, a mi criterio, te otorga toda la capacidad para comprender y mejorar.

Configurar la Raspberry Pi para que se pudiera ejecutar el programa de predicción, también supuso un trabajo enriquecedor. El panel LED, apenas posee documentación asociada y mediante código que existía para otros paneles, se pudo reconstruir y conseguir que funcionará en el del proyecto. Las funciones de proyectar texto, colores, o frases en movimiento fueron agrupadas para crear una pequeña API que pudiera incluirse en el programa de predicción. Al mismo tiempo, se diseño un reloj para poder dar otro uso al panel.

El proceso de optimización de los modelos fue muy enriquecedor, probar tantas versiones diferencias así como leer sugerencias de otros usuarios, aporta un gran conocimiento. Esperar la mejora gradual se convirtió en algo muy satisfactorio: Observar las gráficas con la curiosidad y la emoción de si alcanzarían el valor deseado.

Por último, ver el modelo en marcha, ejecutándose y proyectando la información en el panel LED, vuelve tangible todo el trabajo realizado.

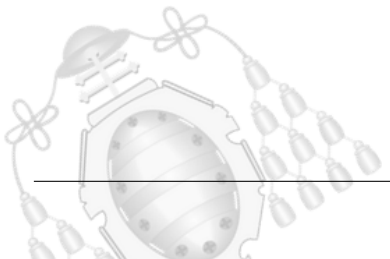
7.2.- Trabajo futuro

Como trabajo futuro para el proyecto existen múltiples implementaciones que podría ser interesante estudiar. Las diferentes vías de trabajo planteadas son:

- **Transformar el conjunto de datos**: se trata de un proceso complejo y de conocimientos avanzados. Se pretende transformar el formato "one-hot vector": (1,0,0,0,0,0,0) a otro más eficiente como "target enconding". Conseguir que las imágenes describan más información en sus soluciones podría aportar una gran mejora al modelo. Estudiar esa evolución puede ser un punto de investigación muy interesante. Se dice que la transformación es compleja porque deberá realizarse de forma automática o mediante otro modelo de inteligencia. Cambiar manualmente las 35.888 entradas no es una opción para una única persona. Esto sería un proceso tedioso y además las respuestas no serían objetivas. El cansancio del usuario produciría cambios en los criterios a lo largo del proceso. Conseguir que un modelo proyecte sus resultados Softmax como solución de las entradas puede ser una de las

alternativas más interesantes. Permitir a un modelo, construir las nuevas soluciones de la base de datos en formato Softmax, aportaría un dataset más natural para la Inteligencia Artificial. Tras la reconstrucción del dataset, un humano podría eliminar las soluciones erróneas.

- **Registro de las predicciones**: cuando se pretenda incluir en el simulador de conducción sería interesante incorporar una recogida de datos. Con esto, el modelo realizaría predicciones que colocaría en una función temporal. Tras haber terminado la simulación, los datos del simulador y el registro de emociones se podrán unir para construir los resultados finales. Podrán asociarse los diferentes puntos conflictivos de la conducción con las emociones que se experimentaban antes, durante y después.
- **Autosuficiencia en la Raspberry Pi**: podría ser interesante construir un sistema que permitiera a la placa descargar los modelos de un servidor. De esta forma, mediante, por ejemplo un servicio REST, el programa podría dinámicamente actualizar el modelo. Si se deseará incorporar como dispositivo embarcado con conectividad a Internet, el modelo podría actualizarse cada día, manteniéndose siempre actualizado.



8. Manual de usuario y manual de instalador

8.1.- Manual de usuario

Inicialmente podrá descargarse el proyecto completo del repositorio de GitHub [1]. Para utilizarlo correctamente, recuerde descomprimir el archivo "dataset.rar" que se encuentra dentro del directorio "Entrenamiento_modelo".

8.1.1.- Uso en PC

Para ejecutar el programa con las próximas instrucciones será necesario tener ya instalado el entorno de trabajo de Anaconda. Este proceso se explica en el manual de instalador. Además, será necesaria una WebCam para la interpretación de las emociones en tiempo real. Habiendo descargado todos los archivos necesarios, el directorio tendrá un contenido semejante al de la figura 8.3.

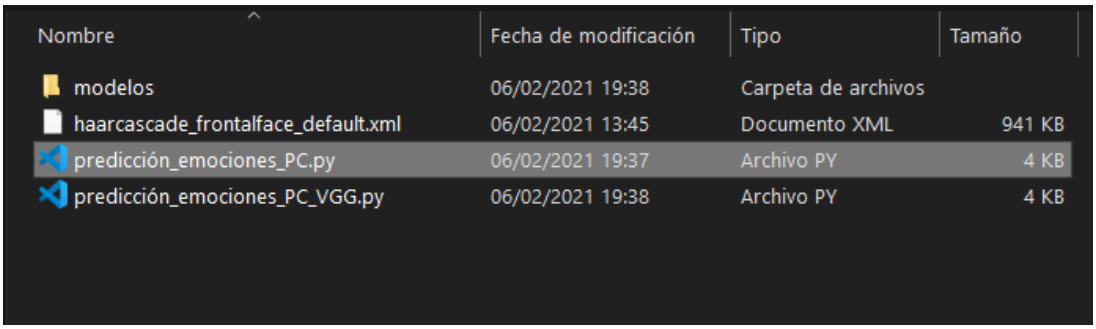


Figura 8.1.- Contenido de los directorios para la ejecución

Abriendo con Visual Studio Code el programa de predicción de emociones, podrán seleccionarse algunos parámetros de interés e iniciar la lectura. La vista del usuario será semejante a la mostrada en la figura 8.2

[1] https://github.com/IgnacioPrieto/Reconocimiento_emociones

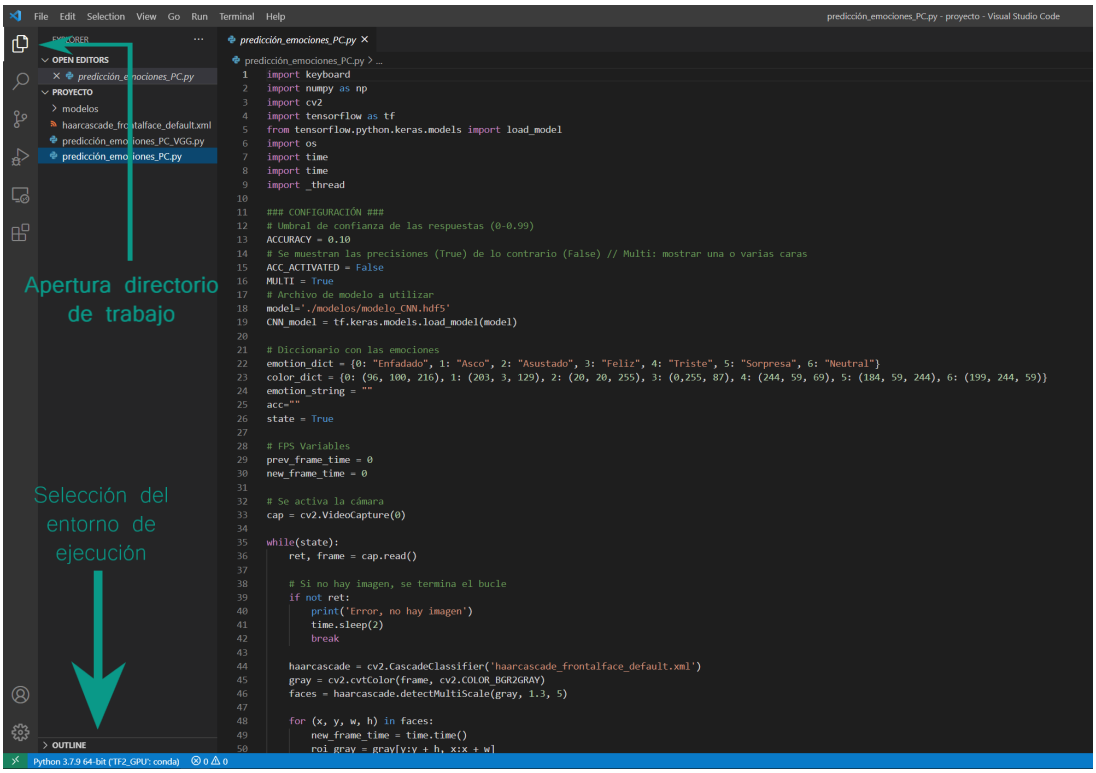


Figura 8.2.- Visual Studio Code interface

En la figura 8.3, se muestran las tres modificaciones que pueden realizarse sobre el programa. “ACCURACY”, al igual que ocurría en la Raspberry, permite fijar un filtro para evitar predicciones poco precisas. “ACC_ACTIVATED” es un booleano que permite indicar si se desea visualizar por pantalla la probabilidad de éxito asociada a la respuesta. Finalmente, la variable“MULTI”, es otro booleano que adapta el formato en el que se visualizan los datos. Si varios rostros van a formar parte de la imagen, es más representativo establecerla en “True”, de lo contrario, será “False”.

```
### CONFIGURACIÓN ###
# Umbral de confianza de las respuestas (0-0.99)
ACCURACY = 0.10
# Se muestran las precisiones (True) de lo contrario (False) // Multi: mostrar una o varias caras
ACC_ACTIVATED = False
MULTI = True
```

Figura 8.3.- Variables globales que permiten modificaciones

El programa podrá ejecutarse con “F5” o mediante la pestaña run. Si el entorno de ejecución presenta todos los requisitos necesarios, se deberá abrir una ventana con el

vídeo que está grabando la WebCam. En tiempo real se valorarán las emociones que muestre el usuario tal como se visualiza en la figura, 8.4.

Figura 8.4.- Prueba del programa del PC

8.1.2.- Uso en Raspberry Pi

Para utilizar el programa de detección de las emociones en la Raspberry Pi, el procedimiento es sencillo. Primero debe encenderse el dispositivo pulsando el interruptor que posee el cable de alimentación. En los casos donde no exista el interruptor, la Raspberry se encenderá automáticamente al conectarse a la corriente. Para acceder al control de la placa existen cuatro posibles metodologías:

- **Conexión HDMI**: si se conecta la placa a un monitor mediante un cable HDMI, su funcionamiento se corresponde al de un equipo convencional. Pueden realizarse todas las acciones de forma local. Si se desea transferir algún modelo, puede utilizarse un dispositivo USB.

- **Terminal SSH**: puede controlarse mediante el protocolo SSH. Desde la propia terminal podrán ejecutarse los comandos necesarios para gestionar el dispositivo. Si el protocolo SSH ya está activado en la placa, solo se necesita la IP que tiene el dispositivo en la red local. Han de conocerse el usuario y la contraseña del dispositivo, ya que serán solicitados durante el proceso de conexión. Si se configura en el router una dirección IP estática para el dispositivo, su acceso será mucho más práctico. Además, como funcionalidad adicional, si fueron abiertos los puertos en el router como fue explicado en la memoria, es posible utilizarla desde Internet. En combinación con un proveedor de servicios DNS, como no-ip, puede acceder se mediante una nombre. Estas últimas modificaciones no son recomendadas ya que comprometen la seguridad de los equipos.
- **Conexión VNC**: es un programa que vincula nuestros equipos cliente y servidor y permite el control remoto. También será necesario que esté activado este protocolo.
- **Visual Studio Code Remote**: es la versión más recomendada, emplea SSH y facilita mucho las tareas de depuración del código. Una vez configurado, podrá ejecutarse el código y depurarse como si se tratará de archivos locales.

Se detallará cómo efectuar la conexión para el último caso. Cuando se ha instalado Visual Studio Code y la extensión remote, aparecerá un símbolo en la parte inferior izquierda como se muestra en la figura 8.5.

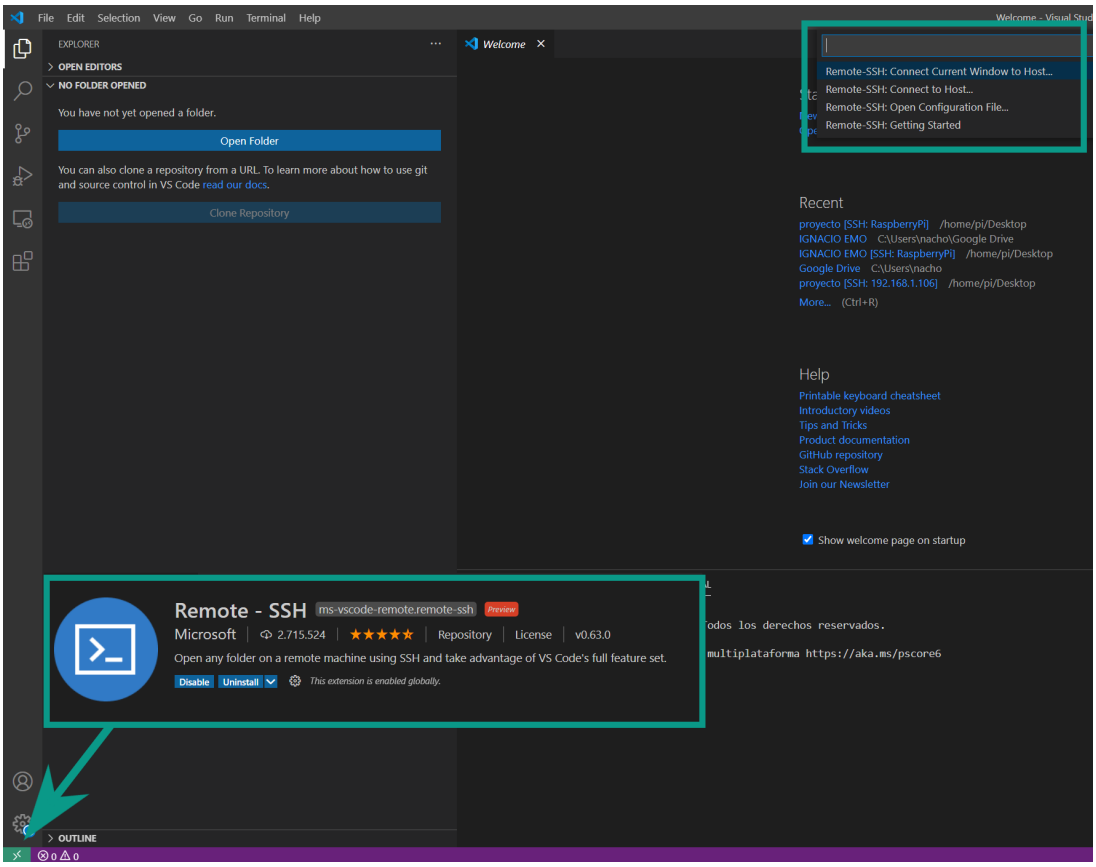


Figura 8.5.- Gestionar la Raspberry Pi desde VSC

Mediante ese menú podrá iniciarse la conexión remota. Inicialmente se deberá completar un archivo de configuración. Se debe especificar el nombre de usuario y la dirección IP del dispositivo al que se pretende conectar. La figura 8.6, muestra los diálogos emergentes que aparecen.

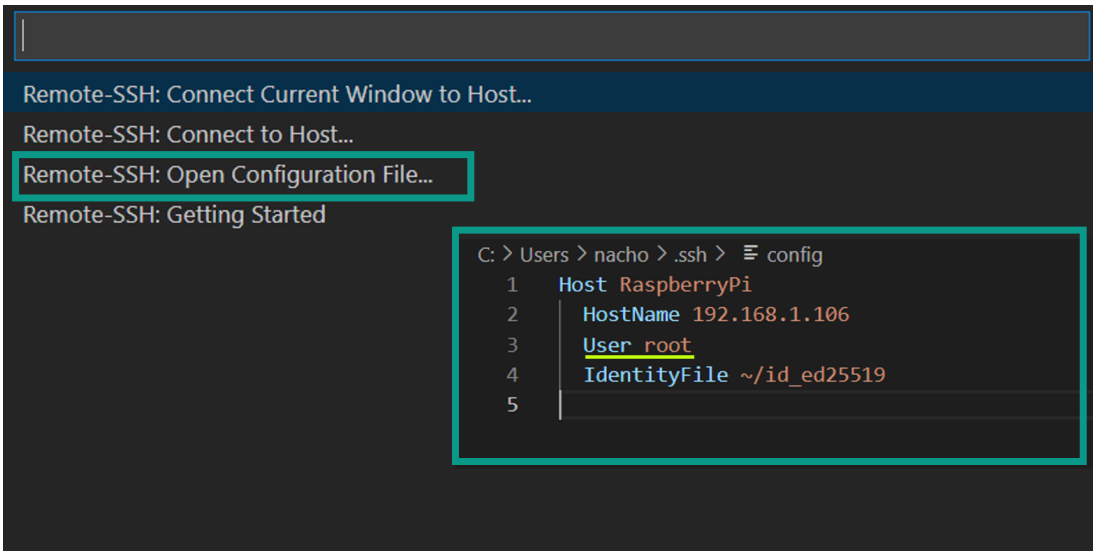


Figura 8.6.- Configuración del dispositivo para efectuar la conexión

Es importante iniciar sesión como usuario root, ya que así podrá depurarse el código. Tras completar la información solicitada, podrá seleccionarse "Remote-SSH: Connect to Host...". Se abrirá un desplegable como el de la figura 8.7.

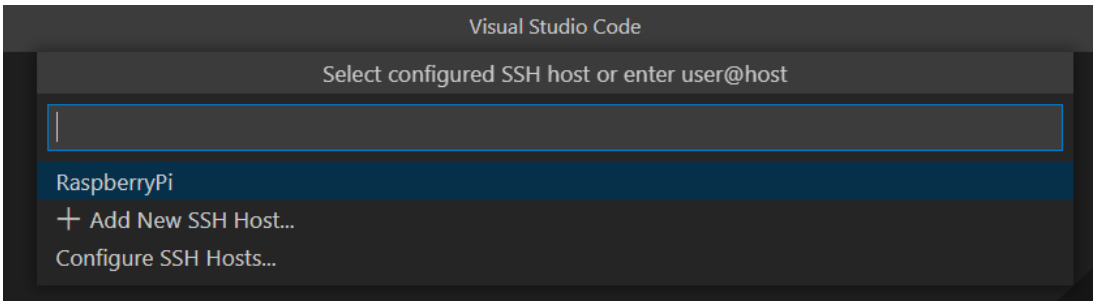


Figura 8.7.- Conexión con el dispositivo configurado

Cuando se ha seleccionado el dispositivo al que se desea conectar, se descargará un archivo en el dispositivo de destino para que pueda ejercer de servidor. Tras realizar la conexión de forma exitosa, aparecerá en la parte inferior izquierda algo semejante a lo que se visualiza en la figura 8.8.

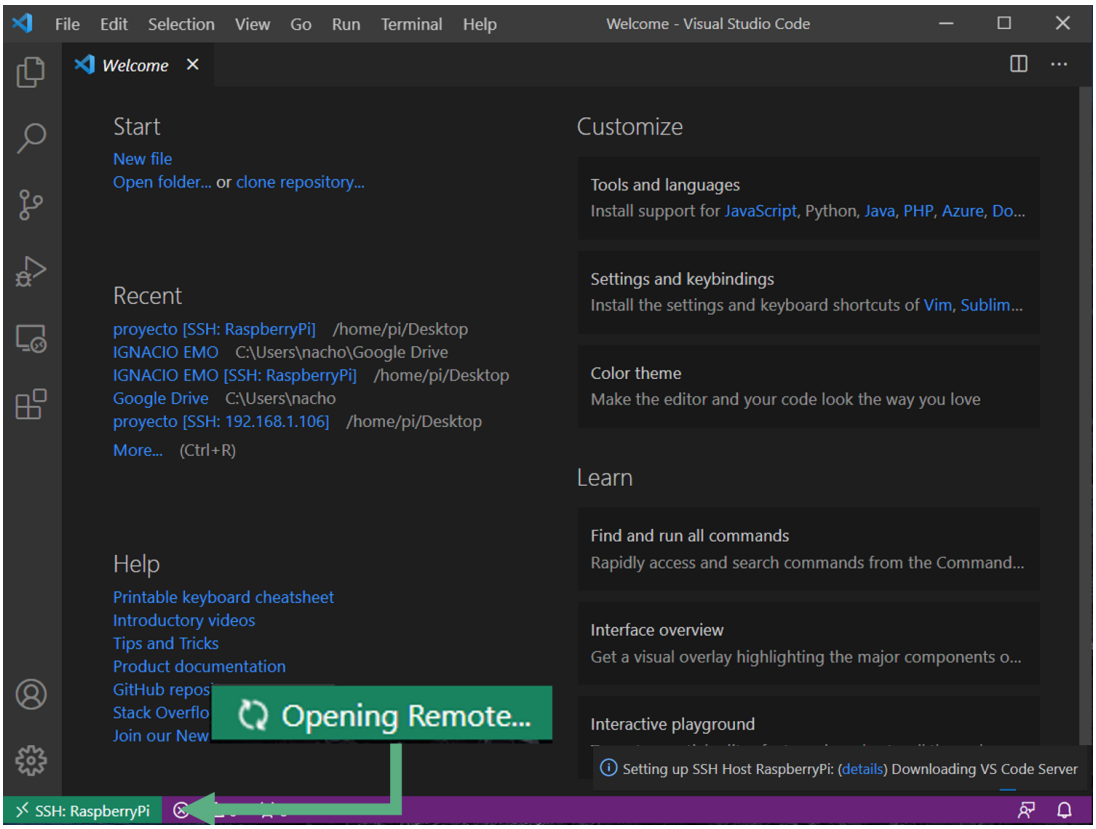


Figura 8.8.- Conexión lograda con el dispositivo remoto

Ahora es posible navegar por las carpetas de la Raspberry para seleccionar la que interese. Situándose en la carpeta del proyecto, podrán ejecutarse todos los códigos disponibles. También es posible depurarlos. Si se desea incorporar un modelo nuevo, puede arrastrarse a la carpeta del proyecto y se enviará a la Raspberry.

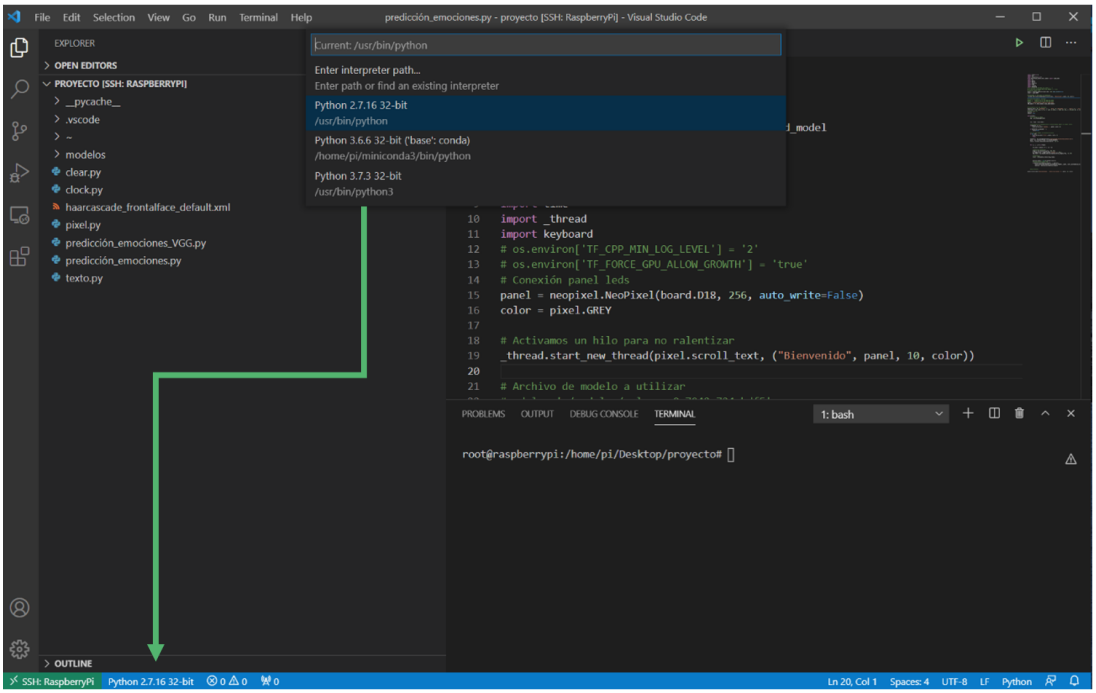


Figura 8.9.- Conexión lograda con el dispositivo remoto

En la figura 8.9 se muestra cómo se visualiza una carpeta como directorio de trabajo y dónde se cambia el entorno de ejecución. La región que señala la flecha permite seleccionar en qué entorno de la Raspberry se desea ejecutar el código. Esto es útil si se emplea el software Miniconda, que facilita trabajar con diferentes entornos de ejecución. En la sección izquierda, pueden visualizarse todos los archivos ".py" que pueden ejecutarse. Haciendo click en el archivo, se visualizará el código en la ventana de edición. Presionando la tecla **F5**, el código se ejecutará. También es posible mediante el desplegable superior "Run" y "Start Debbuging". Si se desea ejecutar mediante consola de comandos, también es posible. Utilizando el programa Putty se establece la conexión SSH con la Raspberry. Ha de accederse con usuario root y su respectiva contraseña. La figura 8.10, muestra la conexión mediante Putty.

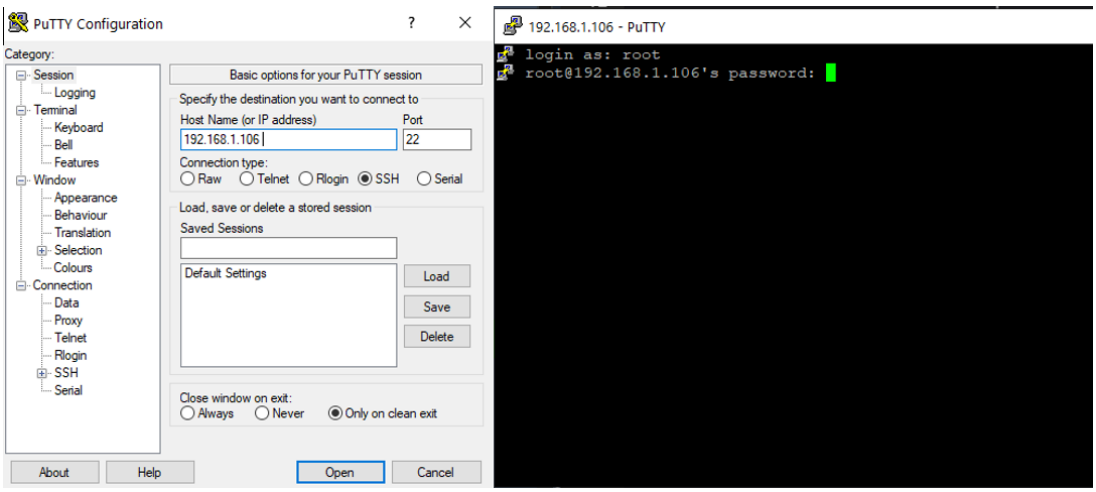


Figura 8.10.- Conexión SSH mediante putty

Cuando la conexión se haya establecido con éxito, deben ejecutarse los siguientes comandos:

- *cd ..*
- *cd home/pi/Desktop/proyecto/*
- *python3,7predicción*$_e$*mociones.py*

Con esto, el panel de LEDs se iluminará y comenzará la ejecución del programa. Es posible alterar parámetros del código para obtener con un indicador la precisión de cada prueba o si se desea aplicar un filtro. Estas modificaciones se encuentran al inicio del código. "ACC_ACTIVATED" es un booleano que activa la proyección de la precisión visible en la parte inferior del panel LED con una dinámica de colores. "ACCURACY" es el valor del filtro, puede utilizarse para limitar la confianza de las respuestas. De esta forma si se introduce, por ejemplo, "0.2", cuando la probabilidad de la categoría seleccionada sea inferior a ese valor, no será mostrada.

8.2.- Manual del instalador

En este manual se detallará como entrenar un modelo, así como conseguir efectuarlo desde la GPU. También se indicarán las características necesarias para ejecutar los archivos de entrenamiento.

8.2.1.- Instalación entorno Anaconda

Para crear el entorno de Anaconda, se facilita en el enlace de todos los contenidos un archivo de extensión ".yml". Con este archivo, podrá crearse una copia del entorno de trabajo de forma sencilla. El procedimiento a realizar dentro del software Anaconda se visualiza en la figura 8.11.

Clonar el repositorio permite instalar todo lo necesario para ejecutar el proyecto. El entrenamiento podrá ser posible con la CPU si se finaliza aquí el proceso. Si en vez de un entorno de trabajo se prefiere utilizar el propio sistema operativo, se proporcionará también un archivo "requirements.txt" compatible con PIP. Se puede crear un modelo propio con la CPU, ejecutando pocas épocas y con paciencia. De esta forma, también podrá monitorizar la evolución de las épocas. Si desea agilizar el proceso considerablemente, deberá instalar los controladores de su GPU que se explicarán a continuación.

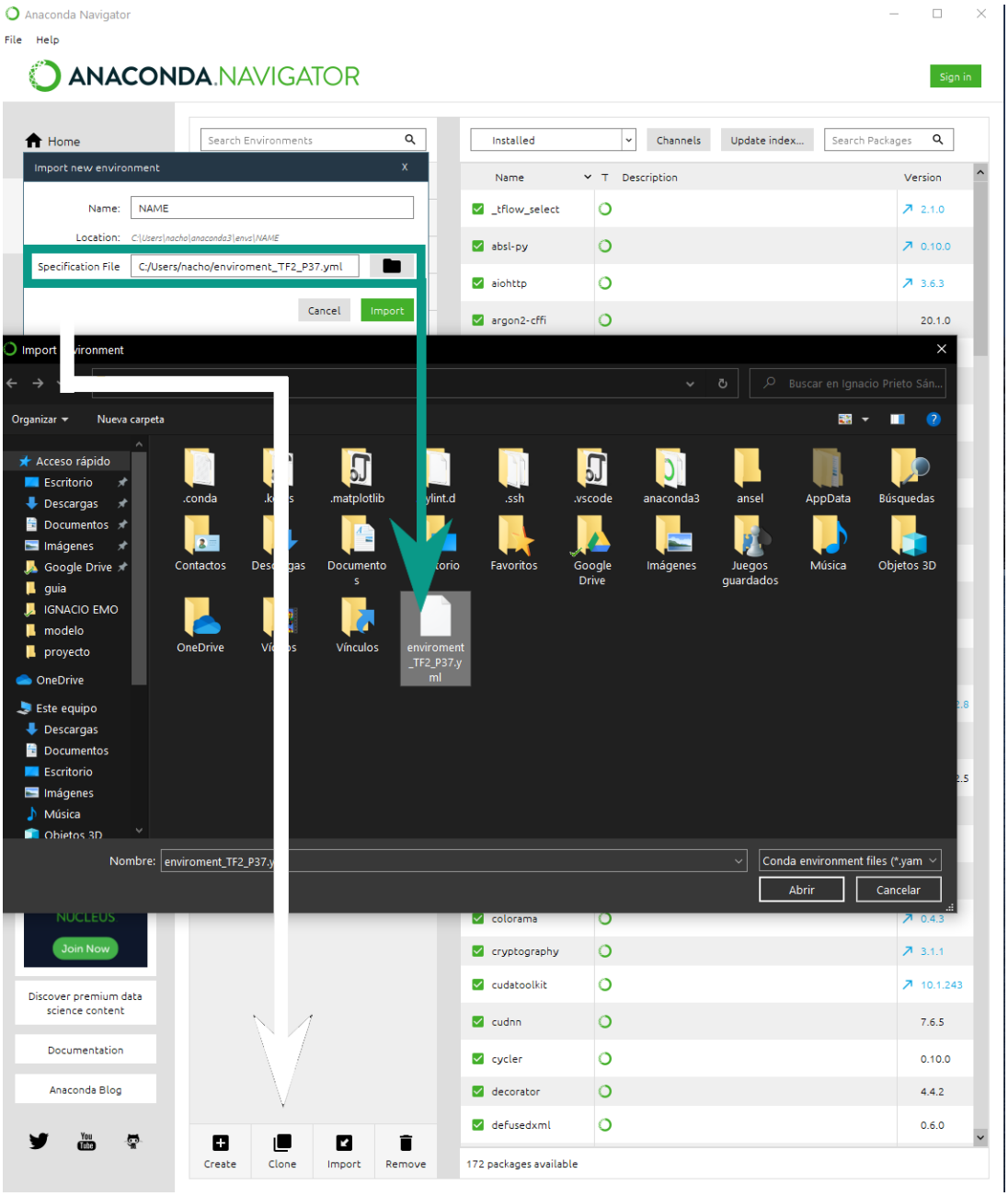


Figura 8.11.- Clonar un entorno de Anaconda

8.2.2.- Instalación GPU toolkit

Para poder ejecutar entrenamientos mediante GPU, debe tener una tarjeta gráfica NVIDIA compatible. No es un proceso sencillo, ya que depende de muchos factores y pueden complicaciones diferentes según el hardware empleado. Se proporcionarán enlaces donde se guía cualquier instalación. En concreto, se concretará más la instalación para

las versiones utilizadas en este proyecto. Aunque las versiones que el usuario posea sean distintas, la guía será igualmente útil para familiarizarse con los formatos.

La tarjeta gráfica va ligada a las versiones de las herramientas necesarias. La tabla representada en la figura 8.12, muestra la compatibilidad entre las versiones de Tensorflow, Python, Cuda y cuDNN. Para efectuar la instalación en un orden lógico, primero, debe identificar que versión de CUDA es compatible con su tarjeta gráfica. Esto puede encontrarlo en el siguiente enlace [65]. Además, dentro de la documentación de Tensorflow aparece una guía de instalación. La guía que más detalla el proceso es la que proporciona la documentación oficial de NVIDIA, pero es muy extensa.

GPU

Versión	Versión de Python	Compilador	Herramientas de compilación	cuDNN	CUDA
tensorflow_gpu-2.4.0	3.6 a 3.8	MSVC 2019	Bazel 3.1.0	8.0	11.0
tensorflow_gpu-2.3.0	3.5 a 3.8	MSVC 2019	Bazel 3.1.0	7.6	10.1
tensorflow_gpu-2.2.0	3.5 a 3.8	MSVC 2019	Bazel 2.0.0	7.6	10.1
tensorflow_gpu-2.1.0	3.5 a 3.7	MSVC 2019	Bazel 0.27.1-0.29.1	7.6	10.1
tensorflow_gpu-2.0.0	3.5 a 3.7	MSVC 2017	Bazel 0.26.1	7.4	10

Figura 8.12.- Tabla de compatibilidad entre TensorFlow, Python, CUDA y cuDNN. Fuente: documentación Tensorflow [66]

En la web de NVIDIA pueden descargarse de forma gratuita CUDA y cuDNN. En este enlace [67] están disponibles todas las versiones de CUDA toolkit. Tras su descarga, se ejecutará un ".exe" que guía la instalación. Cuando el software esté instalado, se habrá creado el directorio en la siguiente ruta: "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA".

Tras haber instalado y ubicado la carpeta anterior, puede instalarse cuDNN. Para poder hacerlo, será necesario registrarse en NVIDIA e identificarse como desarrollador. Tras este proceso, se puede descargar cualquier versión del enlace [68]. Para el caso de cuDNN, la descarga se corresponde con un archivo comprimido en formato ".zip". Su incorporación en las herramientas de NVIDIA es manual, y debe ejecutarla el usuario.

Figura 8.13.- Correspondencia de archivos cuDNN con los directorios de CUDA

La tarea que debe realizar el usuario es copiar el contenido de cada directorio del archivo de cuDNN al directorio con el mismo nombre de CUDA. Es decir, por ejemplo el contenido de la carpeta “bin” de cuDNN ha de ser copiado dentro del directorio “bin” de CUDA. En la figura 8.13 se muestra esta relación entre los directorios, guiando la tarea a realizar.

Tras finalizar la instalación de estos dos componentes, han de ser revisadas las variables de entorno. Durante la instalación, la mayor parte de ellas se crean de forma automática, pero es posible que alguna no lo haga. En la figura 8.14 se muestra el estado de las variables. Algunas se repiten porque, en el caso de este equipo, hay dos versiones instaladas. El caso normal es que solo haya una única versión por lo que no se observará esa duplicidad.

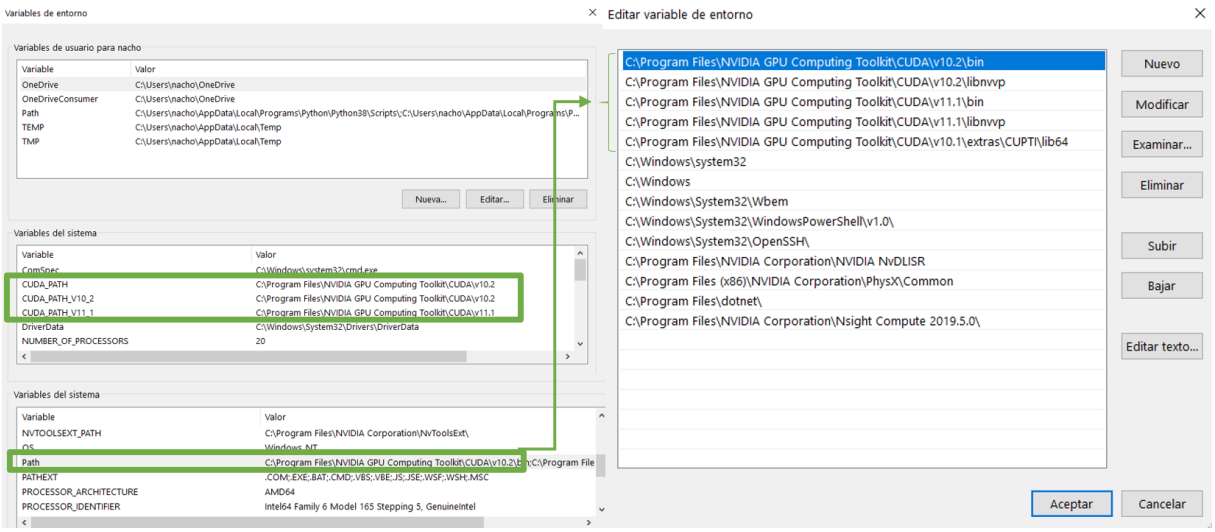


Figura 8.14.- Variables de entorno para CUDA ToolKit

8.2.3.- Ejecución de los entrenamientos

Cuando las herramientas anteriores se hayan instalado, puede ejecutarse el código de entrenamiento sobre la GPU. Para saber si se estaconectando exitosamente la GPU a CUDA y cuDNN, se deben comprobar los mensajes que se proyectan en la terminal. Al iniciar el programa y ejecutar el entrenamiento, la terminal comenzará a proyectar muchos mensajes. Según la versión y potencia de la tarjeta, las respuestas son diferentes. Cuantos más núcleos de CUDA tenga, más subprocesos iniciará. De esta forma, al inicio, solo se visualizarán innumerables líneas de todos los subprocesos que se están iniciando. Si se observa el comienzo de este arranque, se debe observar algo semejante a lo mostrado en la figura 8.15. Bibliotecas de enlace dinámico (dll) abriéndose exitosamente indican que CUDA se está iniciando de forma correcta. Finalmente mostrará la GPU que detectó en el equipo y su capacidad.

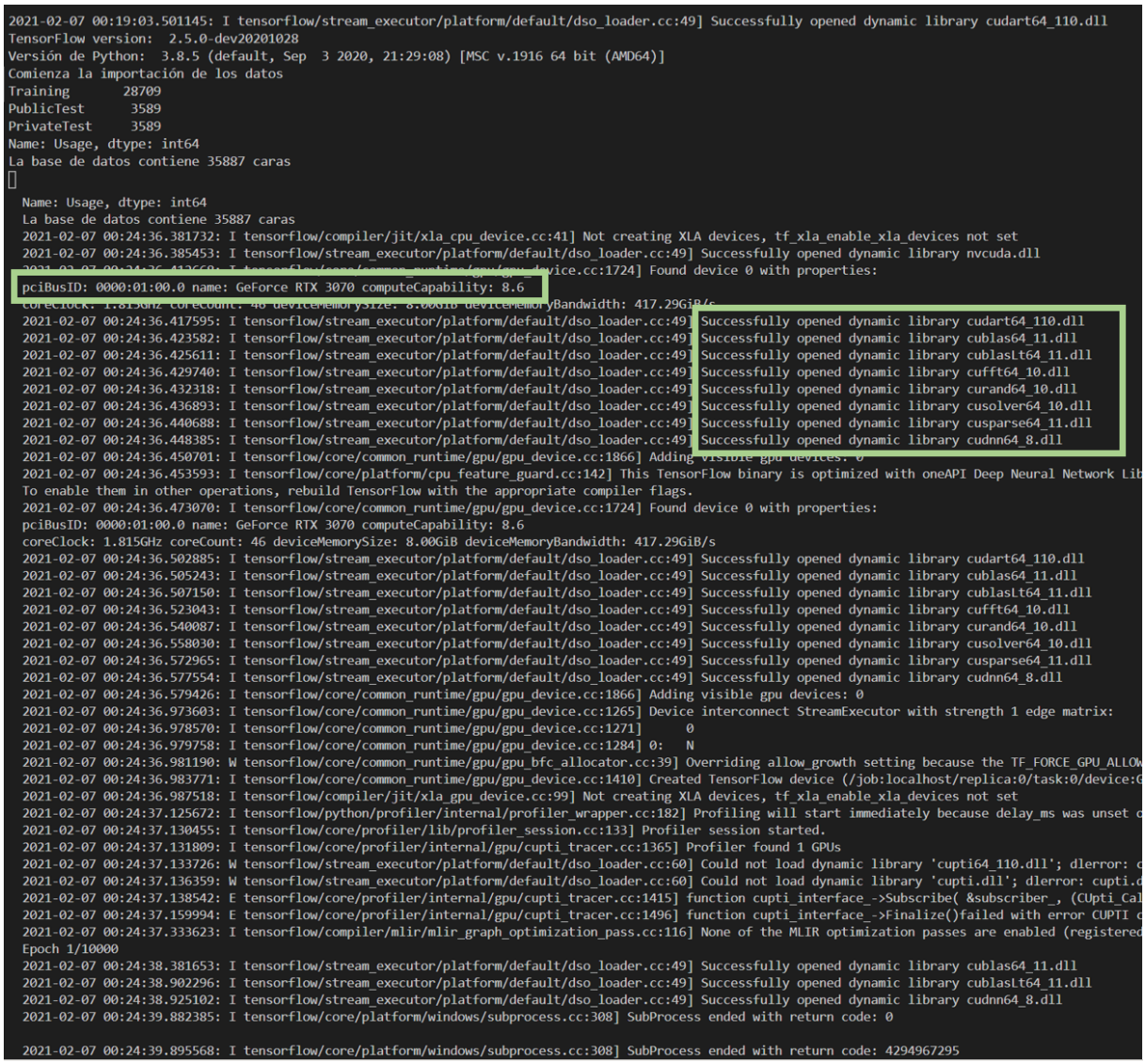


Figura 8.15.- Respuesta terminal arranque CUDA NVIDIA 3070

Interpretar este fragmento es fundamental, permite observar si falta algún archivo “dll” o una variable de entorno para llegar hasta este. Este análisis es el principal mecanismo de localización de errores y fuente para buscar su solución. La librería CUPTI no es necesaria por lo que, no cargarla, no es un problema. Cargar CUDA para la GPU NVIDIA 2070 se visualiza en la figura 8.16.

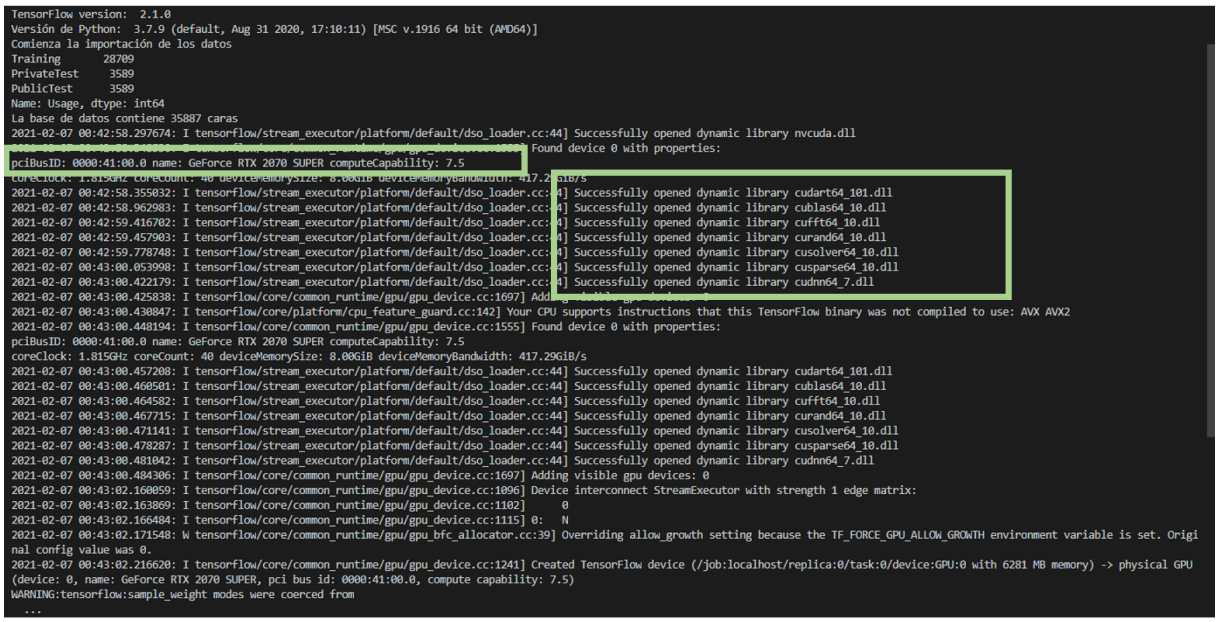


Figura 8.16.- Respuesta terminal arranque CUDA NVIDIA 2070

Cuando el entrenamiento se haya iniciado, en la terminal debe aparecer el progreso de cada época. Si se está ejecutando sobre la tarjeta gráfica, el tiempo para cada una será del orden de 15s. Además, si se visualiza el rendimiento de la GPU, se verá como asciende a más del 50 % de carga. La figura 8.17 muestra la terminal una vez iniciadas las épocas y el rendimiento de la GPU.

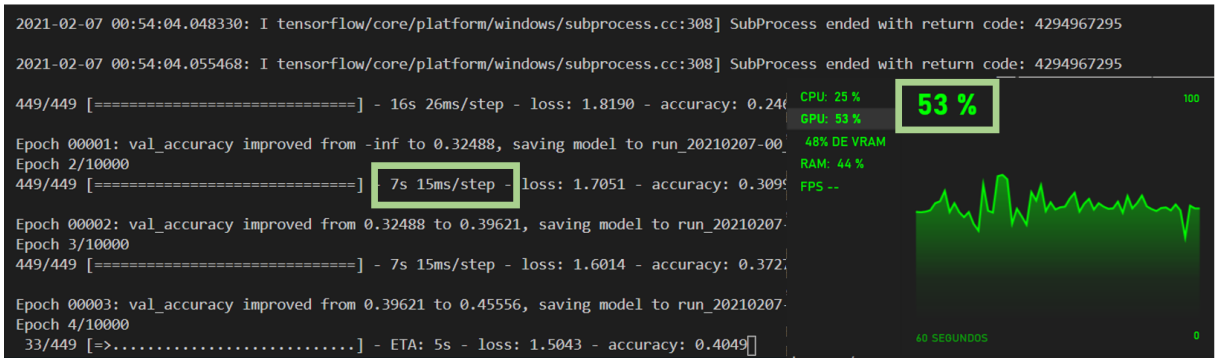


Figura 8.17.- Respuesta terminal inicio del entrenamiento y rendimiento

8.2.4.- Monitorización del entrenamiento

Para monitorizar el entrenamiento, debe ejecutarse un comando en la terminal. Si la terminal está ubicada en el directorio del proyecto, "tensorboard –logdir=./logs" lanzará Tensorboard de forma local. La plataforma será accesible desde un navegador mediante la dirección "http://localhost:6006/ ". Este proceso se efectuará de forma automática, aunque dependiendo de la situación, podría no hacerlo. Para que el código pueda hacerlo de forma automática deberá tener instalado Python y Tensorflow en su equipo (independientemente del entorno de ejecución).

Con unas simples instrucciones de instalación, "pip install tensorboard" y "pip install tensorflow", el sistema ya es capaz de ejecutarlo (pip3 podría ser necesario). Las versiones no tienen incompatibilidades y no suelen generar conflictos. Se ejecuta Tensorboard para leer los ficheros que se van rellenando durante el entrenamiento. Al no tener interacción directa con el código, no hay incompatibilidad.

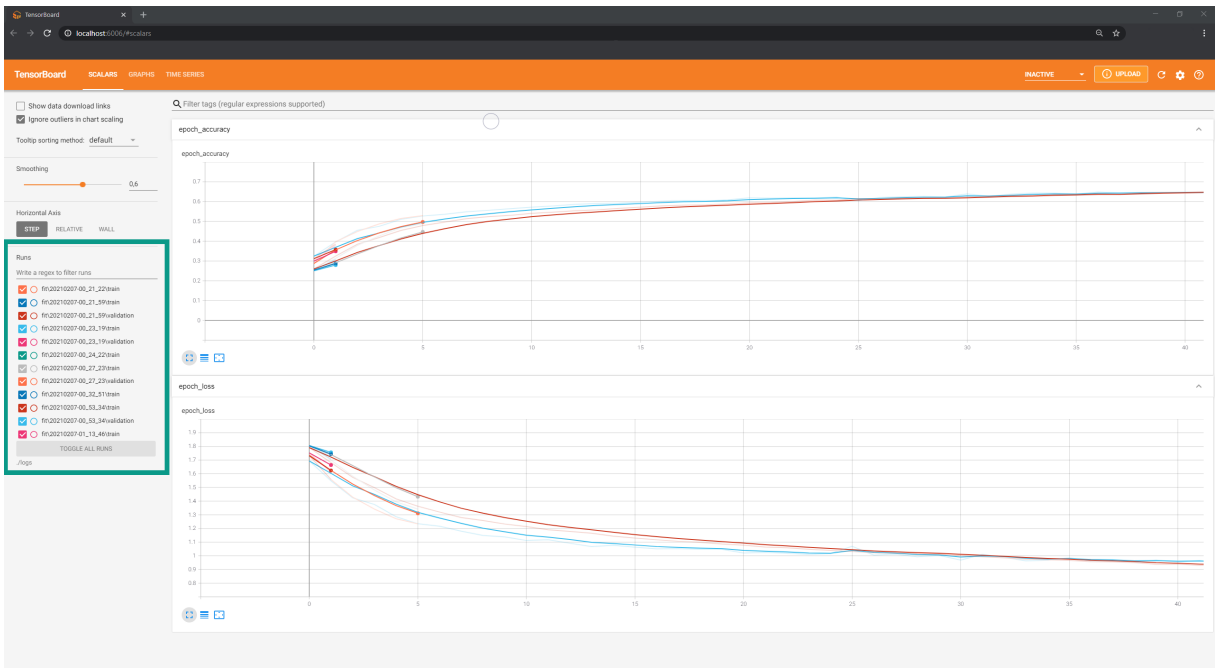


Figura 8.18.- Vista de Tensorboard iniciado en un navegador

Si se ejecuta con éxito, la herramienta se cargará en el navegador. En la figura 8.18 se muestra como se visualiza. La zona enmarcada de color verde muestra todos

los registros existentes. Los más recientes, irán cambiando en tiempo real ya que corresponden al entrenamiento en ejecución. También están accesibles todos los registros de los entrenamientos pasados. Esto facilita la comparación y estudio. Las gráficas son totalmente adaptables permitiendo desplazarse a lo largo de las épocas, hacer zoom, etc...

Cuando el entrenamiento haya finalizado, el modelo estará disponible como un archivo ".hdf5". Para utilizarlo en el programa de inferencia, debe incorporarse en el directorio correspondiente detallado en el manual de usuario. Deberá indicarse dentro del programa el nombre del modelo que se desea ejecutar. Para ello, se especifica su nombre y ruta como se visualiza en la figura 8.19.

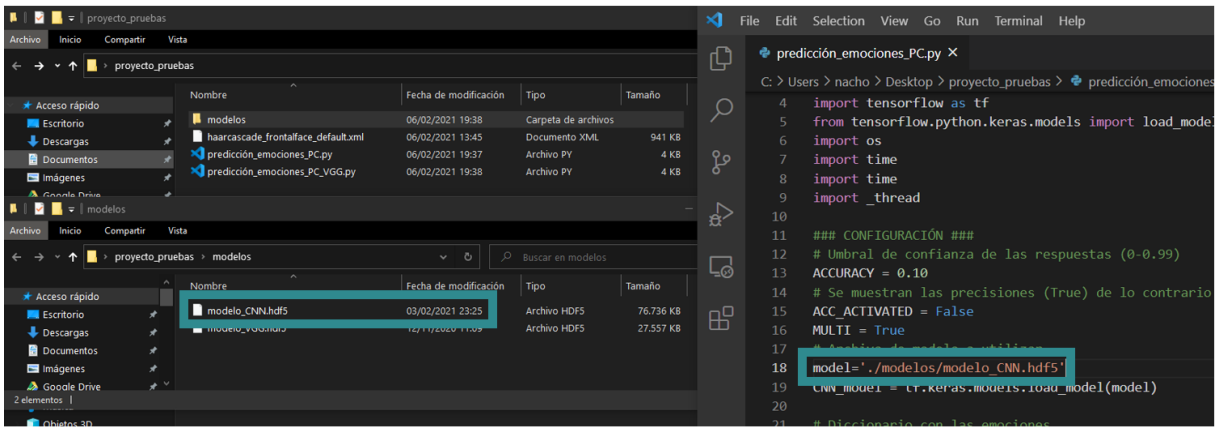


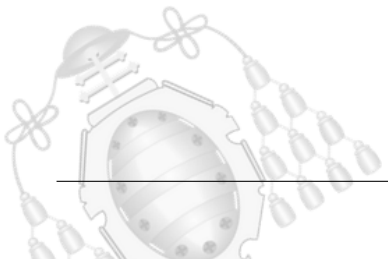
Figura 8.19.- Cargar el modelo creado en el programa de predicción

9. Presupuesto

En este capítulo se detallan los costes del proyecto vinculados a su categoría correspondiente. Los elementos se desarrollan en tres categorías según pertenezcan a hardware, software o recursos humanos. En la primera sección 9.1, se detallan los dispositivos hardware y otros elementos relacionados que han sido utilizados a lo largo de todo el proyecto. En la sección 9.2, se especifican los sistemas operativos empleados. Además, se enumeran todos los componentes de software utilizados, ya sean de uso libre o de pago. En la tercera sección 9.3, se detallan los recursos humanos necesarios para desarrollar, probar y ejecutar el proyecto.

Los equipos y recursos que no fueron empleados en su totalidad para este proyecto se reflejarán en la sección 9.4. En ella, se calcula la amortización de cada elemento, proporcionando su coste real para este trabajo.

Finalmente serán agrupados todos los presupuestos descritos anteriormente. Aplicando la amortización a aquellos elementos especificados, se proporciona el presupuesto total, mostrando el importe final del proyecto, sección 9.5.



9.1.- Equipamiento hardware

Equipo	Concepto	Cantidad	Coste(€)
Raspberry Pi 4	Placa programable	1	59,90
Cable alimentación	Alimentación de la placa	1	6,10
Caja Bruphny	Carcasa protectora y refrigeración	1	20,99
Samsung EVO Plus 64 Gb	MicroSD	1	13,00
Longruner WS2812B LED	Panel LED	1	32,99
Pack Elegoo cables DuPont	Cables extensores de pines	1	6,99
Joy Camera PC	Webcam	1	23,73
Madera	Ensamblaje del panel LED	1	9,99
Plástico	Protector LED	1	5,00
Láminas adhesivas oscuras	Tinte del plástico protector	1	2,25
Cola especial	Pegamento para madera	1	3,00
Pantalla 27’	Monitor	2	500,00
Periféricos	Elementos ofimática	2	100,00
PC	Ordenador potencia gráfica	1	2250
PC 2	Ordenador potencia gráfica	1	2000
SUBTOTAL HARDWARE			**5033,94**

Tabla 9.1.- Costes asociados al hardware

9.2.- Equipamiento software

Elemento	Concepto	Cantidad	Coste(€)
Windows 10 Pro	Sistema operativo	1	259,00
Rasbian OS	Sistema operativo	1	0,00
Visual Studio Code	Editor y depurador de código	1	0,00
Virtual Networking Computing	Control remoto de dispositivos	1	0,00
No-IP	Servicios DNS	1	0,00
Anaconda Individual Edition	Entornos de trabajo	1	0,00
LucidChart	Herramienta de diagramación	1	9,99
Overleaf	Editor de LaTeX en la nube	1	0,00
SUBTOTAL SOFTWARE			**268,99**

Tabla 9.2.- Costes asociados al software

9.3.- Recursos humanos

Tipo de RRHH	Cantidad (horas)	Coste(€/h)	Coste total(€)
Investigador MachineLearning	378	20,00	7560,00
Programador Junior	126	15,00	1890,00
SUBTOTAL RRHH			**9450,00**

Tabla 9.3.- Costes asociados a los contratos

9.4.- Amortización

Algunos elementos que conforman el presupuesto de este proyecto no han sido específicamente adquiridos para este. Para aportar su coste real dedicado al proyecto debe calcularse su amortización. En la tabla 9.4, se especifican los elementos hardware y software, a los que se aplica esta reducción de coste en base a su tiempo de uso. Las amortizaciones se aplican según los valores especificados por la Agencia Tributaria [69],

donde el coeficiente lineal máximo es del 20 % para equipos hardware y del 33 % para software.

Elemento	Precio total(€)	Amortización(%)	Coste total(€)
PC	2250,00	20	450,00
PC 2	2000,00	20	400,00
Pantalla 27’	500,00	20	100,00
Periféricos	100,00	20	20,00
Windows 10 Pro	259,00	33	85,47

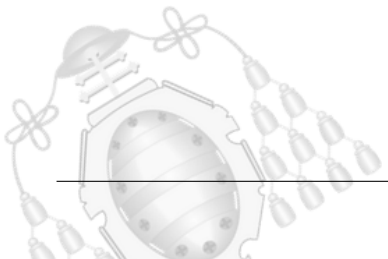
Tabla 9.4.- Costes de los elementos con amortización

9.5.- Presupuesto total

Tipo de coste	Subtotal(€)	Total con amortización(€)
Recursos humanos	9450,00	9450,00
Equipamiento hardware	5033,94	1153,94
Equipamiento software	268,00	98,46
Costes indirectos (agua y luz)	300,00	300,00
Subtotal		**11001,46**
IVA (21 %)		**2310,30**
PRESUPUESTO FINAL DEL PROYECTO		**13311,80**

Tabla 9.5.- Desglose final del proyecto

El presupuesto final del proyecto es de TRECE MIL TRESCIENTOS ONCE EUROS CON OCHENTA CÉNTIMOS



Bibliografía

[1] A. Mishra, D. Patil, N. Karkhanis, V. Gaikar and K. Wani, "Real time emotion detection from speech using Raspberry Pi 3," 2017 *International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, 2017, pp. 2300-2303, doi: 10.1109/WiSPNET.2017.8300170.*

[2] Suchitra, Suja P. and S. Tripathi, "Real-time emotion recognition from facial images using Raspberry Pi II," 2016 *3rd International Conference on Signal Processing and Integrated Networks (SPIN), Noida, 2016, pp. 666-670, doi: 10.1109/SPIN.2016.7566780.*

[3] C.-CJay Kuo, "Understanding convolutional neural networks with mathematical model", *Scienciences Direct, 2016*

[4] V. Paul and J. Michael, "Robust Real-time Face Detection", *International Journal of computer vision, vol. 57, no. 2, pp. 137-154, May 2004.*

[5] H. Li et al., "A Multi-feature Fusion and SSAE-Based Deep Network for Image Semantic Recognition," 2019 *IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService), Newark, CA, USA, 2019, pp. 322-327, doi: 10.1109/BigDataService.2019.00057.*

[6] Aurélien Géron, (2019), *Hands-on Machine Learning with Scikit-Learn, KEras & Tensorflow, página 31*, Editorial Oreilly Media

[7] Salvador Torra PorrasEnric Monte Moreno, *Modelos Neuronales Aplicados en Economía: Casos Prácticos mediante Mathematica / Neural Networks, página 17*

[8] Citación creadores del dataset: *I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor,*

M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, and Y. Bengio. Challenges in representation learning: A report on three machine learning contests. Neural Networks, 64:59–63, 2015. Special Issue on "Deep Learning of Representations"

[9] Gráfica operaciones coma flotante, https://ourworldindata.org/uploads/2020/11/Transistor-Count-over-time.png 15 de julio 2020

[10] El auge de la IA, https://www.wipo.int/wipo_magazine/es/2019/01/article_0001.html#:~:text=El%20auge%20de%20la%20AI&text=Desde%20la%20aparici%C3%B3n%20de%20la,de%20publicaciones%20cient%C3%ADficas%20al%20respecto. 10 de septiembre 2020

[11] Internet of Things, https://www.obrasurbanas.es/internet-cosas-iot/ 15 de julio 2020

[12] Librería OpenCV, https://opencv.org/opencv-4-5-0/ 15 de julio 2020

[13] Librería Dlib, http://dlib.net/ 15 de julio 2020

[14] Emotion Recognition Method Based on Multimodal Sensor Fusion Algorithm, https://www.koreascience.or.kr/article/JAKO200824556528619.page 18 de julio 2020

[15] Estudio de la conducción, https://www.fundacionmapfre.org/fundacion/es_es/programas/seguridad-vial/movilidad-segura-salud/sabias-que/control-emociones-conduccion-segura.jsp 4 de enero 2021

[16] Google Coral, https://www.theverge.com/2020/1/14/21065141/google-coral-ai-edge-computing-products-applications-cloud 15 de julio 2020

[17] Raspberry-Pi, https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/?resellerType=home 15 de julio 2020

[18] Sistema operativo Rasbian, https://www.raspberrypi.org/downloads/raspberry-pi-os/ 16 de julio 2020

[19] Software para preparar la microSD, https://www.balena.io/etcher/ 16 de julio 2020

[20] Lenguaje Python, https://es.wikipedia.org/wiki/Python 16 de julio 2020

[21] Extensión Remote para VSC, https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack 16 de julio 2020

[22] Protocolo SSH, https://www.ssh.com/ssh/protocol/ 16 de julio 2020

[23] Proveedor de dominios de Internet, https://www.no-ip.com 17 de julio 2020

[24] Test de Turing, https://searchenterpriseai.techtarget.com/definition/Turing-test 17 de julio 2020

[25] Minimización del riesgo empírico, https://es.wikipedia.org/wiki/Aprendizaje_supervisado 18 de julio 2020

[26] Aprendizaje no supervisado, https://www.datarobot.com/wiki/unsupervised-machine-learning 18 de julio 2020

[27] Aprendizaje multitarea, https://repositorio.upct.es/bitstream/handle/10317/706/amp.pdf?sequence=1&isAllowed=y 18 de julio

[28] Navie Bayes, https://www.jacobsoft.com.mx/es_mx/clasificador-naive-bayes/ 18 de julio 2020

[29] Fórmulas algoritmo K-Means, https://medium.com/datos-y-ciencia/aprendizaje-no-supervisado-en-machine-learning-agrupacion-bb8f25813edc 18 de julio 2020

[30] Modelar comportamiento XOR, https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b 12 de septiembre 2020

[31] Funciones de activación, https://www.analyticssteps.com/blogs/7-types-activation-functions-neural-network 19 de diciembre 2020

[32] Red neuronal profunda, https://insidebigdata.com/2018/10/07/introduction-deep-learning-neural-networks/neural-network-diagram/ 19 de diciembre 2020

[33] Función de error, https://stats385.github.io/assets/img/grad_descent.png 19 de diciembre 2020

[34] Estructura de redes convolucionales , http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf/ 23 de diciembre 2020

[35] Simulador filtros kernel, https://setosa.io/ev/image-kernels/ 28 de diciembre 2020

[36] Filtro convolucional, https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/ 26 de diciembre 2020

[37] Simulador de una CNN en el espacio, https://www.cybercontrols.org/ 2 de enero 2021

[38] Definición y capacidad del BigData, https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/ia-inteligencia-artificial-y-machine-learning-se-vinculan-al-big-data 10 de septiembre 2020

[39] Definición csv, https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/ia-inteligencia-artificial-y-machine-learning-se-vinculan-al-big-data 10 de septiembre 2020

[40] Influencia del dataset, https://sitiobigdata.com/2018/08/27/interpretar-dataset-en-machine-learning/ 11 de septiembre 2020

[41] Definición de emoción, https://www.significados.com/emocion/ 29 de septiembre 2020

[42] Definición de NumPy, https://numpy.org/ 19 de octubre 2020

[43] Estructura de clases Keras, https://keras.io/api/preprocessing/image/ 20 de octubre 2020

[44] Definición de tensorflow, https://www.tensorflow.org/?hl=es-419 10 de octubre 2020

[45] Definición de Keras, https://keras.io/ 12 de septiembre 2020

[46] Inicios de Tensorflow, https://modeliza.me/blog/ejecucion-eager-con-keras-y-tensorflow/ 10 de octubre 2020

[47] Neuronas de tipo dense, https://keras.io/api/layers/core_layers/dense/ 25 de octubre 2020

[48] Proyecto sobre la función Dropout, https://jmlr.org/papers/v15/srivastava14a.html 25 de octubre 2020

[49] Métricas de Keras, https://keras.io/api/metrics/accuracy_metrics/#accuracy-class 25 de octubre 2020

[50] Definición One-hot Vector, https://en.wikipedia.org/wiki/One-hot 3 de noviembre 2020

[51] Función categorical crossentropy, https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy 14 de diciembre 2020

[52] Clase Adam Keras, https://keras.io/api/optimizers/adam/ 14 de diciembre 2020

[53] Método fit Keras, https://keras.io/api/models/model_training_apis/ 14 de diciembre 2020

[54] CVSLogger Keras, https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/CSVLogger 15 de diciembre 2020

[55] Documentación Tensorboard, https://www.tensorflow.org/tensorboard/get_started?hl=es-419 15 de diciembre 2020

[56] Eficiencia GPU, http://www.redgamingtech.com/why-modern-gpus-perform-faster-than-cpus-good-at-parallel-computing-part-1/ 16 de diciembre 2020

[57] Nvidia GPU comparado con CPU, https://www.nvidia.com/es-la/drivers/what-is-gpu-computing/ 15 de diciembre 2020

[58] Cuda developer, https://developer.nvidia.com/cuda-zone 16 de diciembre 2020

[59] Fenómeno de underfitting, https://www.datarobot.com/wiki/underfitting/ 17 de diciembre 2020

[60] Imagen de efectos en el entrenamiento, https://www.educative.io/edpresso/overfitting-and-underfitting 16 de diciembre 2020

[61] Raspberry Pi 4 PINES, https://www.raspberrypi.org/documentation/usage/gpio/ 4 de enero 2021

[62] Modelo VGG16, https://neurohive.io/en/popular-networks/vgg16/ 17 de diciembre 2020

[63] Toward data science, https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c 17 de diciembre 2020

[64] Comparativa de procesadores, https://www.cpu-monkey.com/ 4 de enero 2021

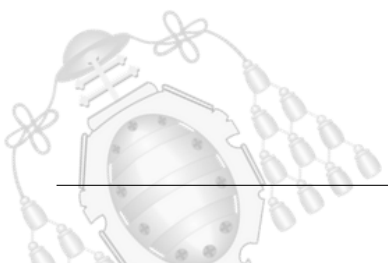
[65] Información GPU TensorFlow, https://www.tensorflow.org/install/gpu?hl=es-419 2 de enero 2021

[66] Compatibilidad TensorFlow, https://www.tensorflow.org/install/source_windows#tested_build_configurations 2 de enero 2021

[67] Web de descarga CUDA , https://developer.nvidia.com/cuda-toolkit-archive 3 de enero 2021

[68] Web de descarga cuDNN, https://developer.nvidia.com/rdp/cudnn-archive 3 de enero 2021

[69] Tabla de amortización Agencia Tributaria, https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml 1 de enero 2021



A. Siglas y acrónimos

IA: Inteligencia Artificial

HDMI: High-Definition Multimedia Interface

SSH: Secure Shell

VNC: Virtual Network Computing

IoT: Internet Of Things

WOL: Wake on LAN

DNS: Domain Name System

BIOS: Basic Input Output System

VPN: Virtual Private Network

MAC: Media Access Control Address

SPAM: mensaje electrónico no deseado

CPU: unidad central de procesamiento

GPU: unidad de procesamiento gráfico

CNN: convolutional neural network

RNA: artificial neural network

TF: TensorFlow

ML: machine learning

RAM: random access memory