

**Escuela Técnica Superior de Ingeniería**

**Universidad de Huelva**

Máster en Ingeniería Informática

**Trabajo Fin de Master**

Creación de un modelo de visión artificial ligero con Aprendizaje Federado

José María Vázquez Guerra

Jesús Campos Márquez

Julio 2022

Bajo la tutoría de Antonio Peregrín Rubio

Agradecimientos

**“*Dime y lo olvido, enséñame y lo recuerdo, involúcrame y lo aprendo*”**

Benjamín Franklin

Quiero dar las gracias a

Resumen

La inteligencia Artificial y, en concreto, el Deep Learning se ha vuelto una de las tecnologías con más auge en los últimos años. Esta subida es debida gracias a la gran experimentación real que se está llevando a cabo y a que, con nuevas ideas, proporcionamos no solo nuevas soluciones a problemas, sino que también mejoramos la resolución de estos para hacerlos de forma más segura y eficiente.

Sin embargo, los modelos tradicionales de aprendizaje automático necesitan de mucho tiempo de entrenamiento y de servidores potentes para conseguir buenos resultados, por lo que, para solucionar esto, surge el Aprendizaje Automático Federado o Aprendizaje Federado, otorgando la capacidad de crear modelos colaborativos sin la necesidad de estos grandes y potentes servidores, un entrenamiento muy exhaustivo inicial, ni de tener la capacidad de almacenar grandes cantidades de datos de entrenamiento.

Este Aprendizaje Federado, puede ser la clave para la creación de modelos que están constantemente aprendiendo y que satisfacen diferentes necesidades que se encuentran en la población gracias a la aplicación responsable de la IA, como la ayuda a las personas con discapacidad visual, a partir de un modelo capaz de adaptarse mejor a su entorno, mejorando la seguridad debido a la minimización de los sesgos.

Por lo tanto, se propone diseñar e implementar un modelo de visión artificial eficiente, optimizado para su implementación en dispositivos con recursos limitados, modelado por una arquitectura de aprendizaje federado. Incluirá un estudio experimental que documente su despliegue en el mundo real y cuantifique su utilidad.

Palabras clave: Aprendizaje Federado, FL, Raspberry, Deep Learning, IA, Aprendizaje profundo, Clasificación, Aprendizaje Automático, Aprendizaje de modelos, Modelos Keras, TensorFlow, Colaborativo, Discapacidad Visual, Media federada, Transfer Learning, MobileNetV2.

Abstract

Artificial intelligence and, specifically, Deep Learning has become one of the most popular technologies in recent years. This rise is due to the large amount of real experimentation that is being carried out and with new ideas, we provide not only new solutions to problems, but also improve the resolution of these problems to make them safer and more efficient.

However, traditional machine learning models need a lot of training time and powerful servers to achieve good results, so, to solve this, Federated Machine Learning or Federated Learning arises, giving the ability to create collaborative models without the need for these large servers, a very exhaustive initial training, or having the ability to store large amounts of training data.

This Federated Learning, can be the key to the creation of models that are constantly learning and that satisfy different needs found in the population with the responsible application of AI, such as helping visually impaired people, from a model capable of better adapting to their environment, improving safety due to the minimization of biases.

It is proposed to design and implement an efficient computer vision model, optimized for implementation on resource-constrained devices, modeled by a federated learning architecture. It will include an experimental study documenting its real-world deployment and quantifying its usefulness.

Key words: Federated Learning, FL, Raspberry, Deep Learning, AI, Deep Learning, Classification, Machine Learning, Model Learning, Keras Models, TensorFlow, Collaborative, Visual Disability, Federated Media, Transfer Learning, MobileNetV2.

Índice General

[Resumen 4](#_Toc106124957)

[Abstract 5](#_Toc106124958)

[Índice General 6](#_Toc106124959)

[Lista de Figuras 7](#_Toc106124960)

[Lista de Comandos 7](#_Toc106124961)

[Lista de Pseudocódigos 8](#_Toc106124962)

[Capítulo 1. Introducción 8](#_Toc106124963)

[1.1 Inteligencia Artificial 8](#_Toc106124964)

[1.2 Machine Learning 10](#_Toc106124965)

[1.3 Deep Learning 10](#_Toc106124966)

[1.4 Neuronal Networks 14](#_Toc106124967)

[1.5 Machine Learning vs Deep Learning vs Neuronal Networks 14](#_Toc106124968)

[1.6 Accesibilidad 16](#_Toc106124969)

[Capítulo 2: 18](#_Toc106124970)

[2.1 Visión Artificial 18](#_Toc106124971)

[2.2 IoT 19](#_Toc106124972)

[2.3 Edge Computing 20](#_Toc106124973)

[2.4 Redes Convolucionales 22](#_Toc106124974)

[2.5 Transfer Learning 25](#_Toc106124975)

[2.6 El Aprendizaje Federado como evolución 27](#_Toc106124976)

[2.7 Edge TPU. Coral USB Acelerator 29](#_Toc106124977)

[2.8 TensorFlow como herramienta para el Machine-Learning 31](#_Toc106124978)

[Capítulo 3 34](#_Toc106124979)

[3.1 Modelo Básico No Federado 34](#_Toc106124982)

[3.2 La Arquitectura Federada 35](#_Toc106124983)

[Capítulo 4 42](#_Toc106124984)

[4.1 Arquitectura Raspberry 42](#_Toc106124986)

[4.2 Puesta en marcha del dispositivo e instalación de librerías 42](#_Toc106124987)

[4.3 El Dataset 51](#_Toc106124988)

[4.4 Entrenamiento del modelo con TensorFlow 54](#_Toc106124989)

[Generación del modelo inicial 54](#_Toc106124990)

[Generación del sistema federado con estructura de ficheros y archivos 54](#_Toc106124991)

[Capítulo 5 54](#_Toc106124992)

[Referencias bibliograficas 56](#_Toc106124993)

Lista de Figuras

[Figura 1 - Diferenciación de conceptos en capas 13](#_Toc105610232)

[Figura 2 - Esquema de capas de la red neuronal 14](#_Toc105610233)

[Figura 3 - Etapas del aprendizaje por capas 15](#_Toc105610234)

[Figura 4 - Características de TensorFlow 17](#_Toc105610235)

[Figura 5 - Campos de uso del Transfer Learning actualmente 18](#_Toc105610236)

[Figura 6 - Funcionamiento esquemático del Transfer Learning 18](#_Toc105610237)

[Figura 7 - Tabla de principales modelos preentrenados 19](#_Toc105610238)

[Figura 8 - Arquitectura general de MobileNetV2 19](#_Toc105610239)

[Figura 9 - Sistema de Aprendizaje Federado de Google (Android GBoard) 21](#_Toc105610240)

[Figura 10 - Coral USB Acelerator 22](#_Toc105610241)

[Figura 11 - Proceso de creación de modelo compatible con TPU Edge 23](#_Toc105610242)

[Figura 12 - Arquitectura de Raspberry Pi 4 Model B 24](#_Toc105610243)

[Figura 13 - Raspberry Pi Imager 25](#_Toc105610244)

[Figura 14 - Instalación física de la Pi Camera 28](#_Toc105610245)

[Figura 15 - Configuración de la Pi Camera (I) 28](#_Toc105610246)

[Figura 16 - configuración de la Pi Camera (II) 29](#_Toc105610247)

[Figura 17 - Esquema de Arquitectura Federada básica 30](#_Toc105610248)

[Figura 18 - Arquitectura Federada con WorkFlow 35](#_Toc105610249)

[Figura 19 - Características de Keras 36](#_Toc105610250)

[Figura 20 - Ejemplo de 2D Average Pooling 37](#_Toc105610251)

[Figura 21 - Ejemplo de Flatten 37](#_Toc105610252)

[Figura 22 - Comparativa de funciones de activación 38](#_Toc105610253)

[Figura 23 - Penúltima capa unida con las últimas 3 capas propias agregadas 39](#_Toc105610254)

[Figura 24 - Ejemplo de Data Augmentation 40](#_Toc105610255)

[Figura 25 - Ejemplo de arreglo del desenfoque (UnsharpMask) 41](#_Toc105610256)

[Figura 26 - Primeras pruebas. 5 dispositivos durante 5 días 41](#_Toc105610257)

Lista de Comandos

[Comandos 1 - Instalación de TensorFlow 26](#_Toc105610264)

[Comandos 2 - Instalación de PyCoral 27](#_Toc105610265)

[Comandos 3 - Ejemplo de creación e instalación de fichero de requirements 27](#_Toc105610266)

[Comandos 4 - Entrar en configuración de Raspberry para habilitar la Pi Camera 28](#_Toc105610267)

[Comandos 5 - Resolución del problema: No module named cv2 29](#_Toc105610268)

[Comandos 6 - Resolución del problema: Matplot error 29](#_Toc105610269)

Lista de Pseudocódigos

[Pseudocódigo 1 - Máquina Virtual 31](#_Toc105606365)

[Pseudocódigo 2 - Devices 32](#_Toc105606366)

[Pseudocódigo 3 - Servidor 33](#_Toc105606367)

Capítulo 1. Introducción

En el primer capítulo, se describirán tanto la motivación que da lugar al trabajo como los objetivos del mismo, además de una explicación de conceptos que nos aportará una idea general de la propuesta.

* 1. Inteligencia Artificial

Con Inteligencia Artificial (IA), habitualmente nos referimos a la capacidad de crear máquinas “inteligentes” que son capaces de desarrollar algún tipo de tarea como si de un humano se tratase, con posibilidad de mejorar de forma iterativa a partir de la información que pueden ir recopilando. No se trata de crear un humano artificial, sino que se trata de crear un tipo de sistema capaz de resolver problemas de cierta complejidad de forma automática.

Concretamente, la IA es un campo que combina la ciencia de la computación y los conjuntos de datos robustos en busca de la resolución de problemas de la forma más eficiente posible.

Stuart Russell y Peter Norvig procedieron entonces a publicar, Artificial Intelligence: A Modern Approach, convirtiéndose en uno de los principales libros de texto en el estudio de la IA. Profundizaron en cuatro posibles objetivos o definiciones de la IA, que diferencian los sistemas informáticos en función de la racionalidad y el pensamiento frente a la actuación:

* Sistemas que piensan como humanos, que se encargan de automatizar actividades de toma de decisiones, resolución de problemas y aprendizaje del mismo, como es el reconocimiento facial o de objetos, siendo el tipo que abordaremos en este documento.
* Sistemas que actúan como humanos, que intentan imitar la forma en el que realizamos tareas los humanos, como por ejemplo hacen los robots, ayudando sobre todo a eliminar tareas peligrosas, pesadas y/o repetitivas.
* Sistemas que piensan racionalmente, donde se pretende emular el pensamiento lógico/racional de los humanos y, en el que se busca que una máquina pueda percibir y razonar en busca de la resolución a la tarea o problema.
* Sistemas que actúan racionalmente, que son los que pretenden imitar de manera racional el comportamiento que tenemos los humanos.

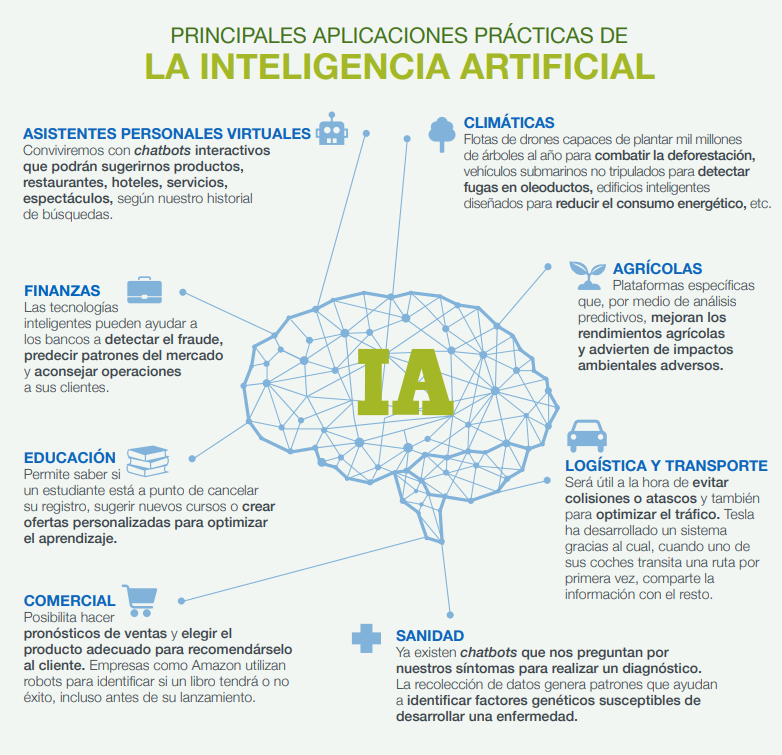
Hoy en día, la IA está presente en todos los ámbitos de nuestra vida como, por ejemplo, en aplicaciones como Siri o Alexa que realizan un procesamiento del lenguaje natural (NLP), convirtiendo nuestra voz en órdenes e incluso las acciones comerciales que nos encontramos en Amazon, recomendando productos en función a nuestras compras y/o búsquedas.

Figura 1 - Principales aplicaciones de la IA

* 1. Machine Learning

El aprendizaje automático (AA) o Machine Learning (ML) es una rama de la inteligencia artificial (IA) y de la informática que se centra en el uso de datos y algoritmos para imitar la forma en que los humanos aprenden, mejorando gradualmente su precisión sin ser expresamente programadas, si no que los sistemas son capaces de identificar patrones para predecir resultados y actuar en consecuencia a ellos.

El término se utilizó por primera vez en 1959 por Arthur Samuel. Sin embargo, en los últimos años ha ganado importante debido al aumento de la capacidad que los dispositivos de computación.

Podemos diferenciar tres categorías principales:

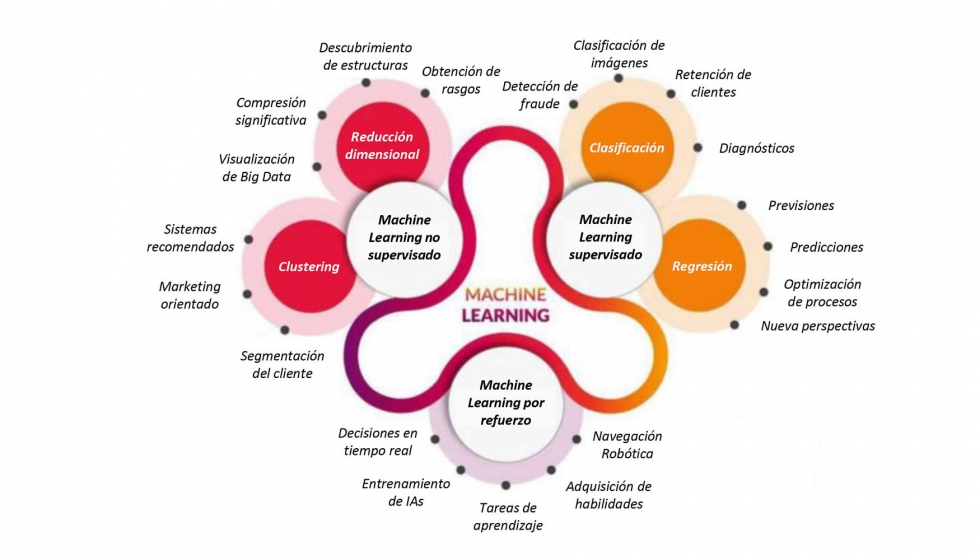
* + Aprendizaje Supervisado, donde los algoritmos necesitan de un aprendizaje previo basado en etiquetas asociadas a los datos, que les permitirá tomar decisiones o hacer predicciones.
  + Aprendizaje no supervisado, donde los algoritmos no necesitan de aprendizaje previo, si no que comienzan a aprender a medida que entrenan, dentro de los datos no organizados/etiquetados.
  + Aprendizaje por refuerzo, donde los algoritmos aprenden a partir de la propia experiencia basados en prueba y error.

Figura 2 - Diferenciación de tipos de Machine Learning

* 1. Deep Learning

El aprendizaje profundo o Deep Learning (DL) es un subconjunto del aprendizaje automático, que consiste esencialmente en una red neuronal con tres o más capas. Estas redes neuronales intentan simular el comportamiento del cerebro humano permitiéndole aprender de grandes cantidades de datos ciertos patrones que le ayudan a resolver el problema para el que se le ha entrenado. Aunque una red neuronal con una sola capa puede hacer predicciones aproximadas, las capas ocultas adicionales pueden ayudar a optimizar y refinar la precisión.

Actualmente, el DL está muy enfocado en el reconocimiento de vídeo, imagen, texto y audio, la conducción autónoma, la robótica y la sanidad entre otros, siendo un campo de estudio orientado a los resultados, por lo que está recibiendo mucha atención por parte de investigadores y académicos, donde los artículos científicos con nuevas formas de mejorar no paran de publicarse y su utilización forma parte de las necesidades del día a día para todos los sectores.

El aprendizaje profundo impulsa muchas aplicaciones y servicios de IA que mejoran la automatización, realizando tareas analíticas y físicas sin intervención humana. La tecnología de aprendizaje profundo está detrás de los productos y servicios cotidianos (como los asistentes digitales, los mandos de televisión con voz y la detección de fraudes con tarjetas de crédito), así como de las tecnologías emergentes (como los coches autónomos).

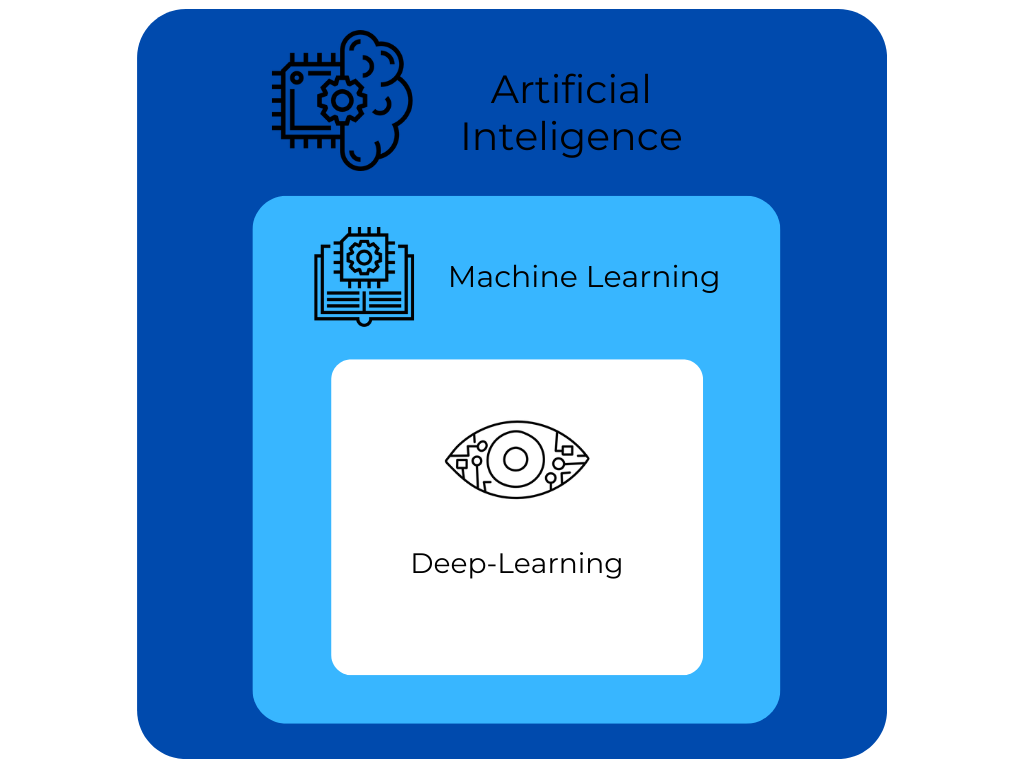
Sin embargo, no es lo mismo hablar de Inteligencia Artificial, Machine Learning y Deep Learning y, hay que hacer una distinción importante entre ellos.

Figura 3 - Diferenciación de conceptos en capas

El Deep Learning, forma parte del Machine Learning y este, a su vez, está englobado en el campo de la Inteligencia Artificial. En Deep Learning, a partir de una gran cantidad de datos y, tras un buen número de etapas de entrenamiento, conseguimos que el ordenador produzca un modelo que generaliza lo aprendido, o lo que es lo mismo, el modelo ha conseguido aprender por su propia cuenta, desde patrones en imágenes para su identificación y clasificación, como la capacidad de realizar predicciones, cuyo entrenamiento hace que cada vez los resultados sean más fiables.

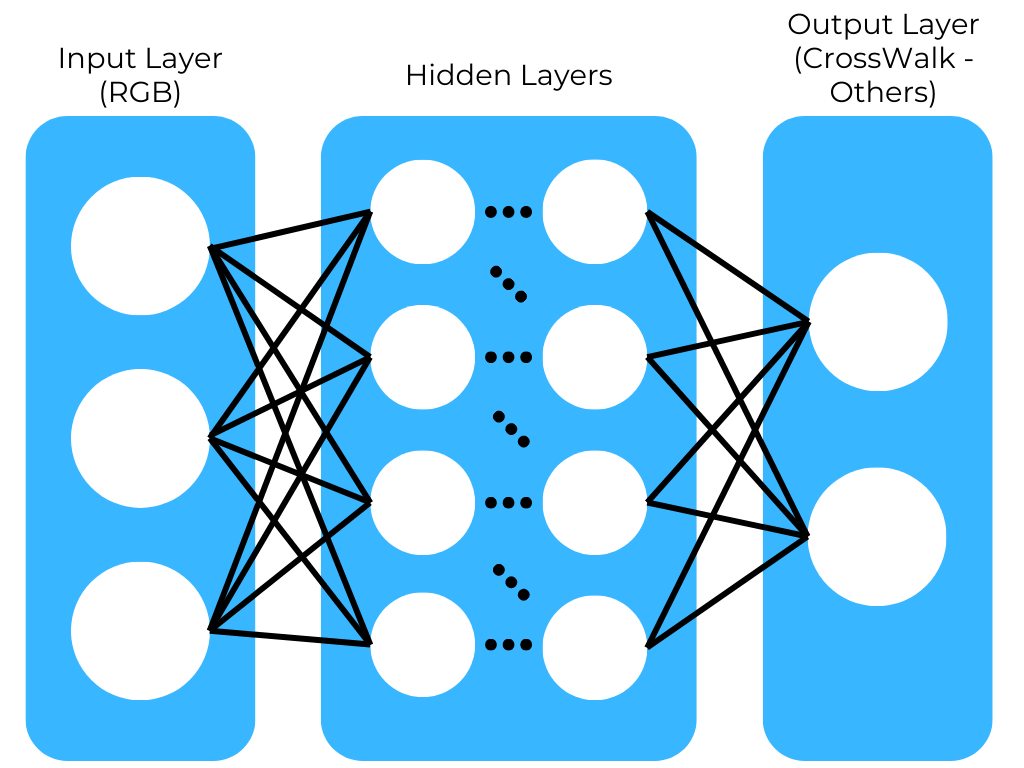
El Deep Learning funciona como una red neuronal, donde estas neuronas son artificiales, formadas por una estructura dividida en 3 capas principales (Figura): Input Layer (la entrada de los datos), Hidden Layers (las capas ocultas que pueden estar formadas por subcapas) y la Output Layer (la capa de salida, o capa de resultados).

Figura 4 - Esquema de capas de la red neuronal

Su flujo de trabajo es sencillo, en la que los datos entran por esta primera capa, como puede ser una imagen y, donde las neuronas se activan o no dependiendo de su función de activación, llegando a reconocer las combinaciones de líneas y puntos en las primeras capas. Posteriormente, se llega a las capas ocultas, donde se procesan y combinan estas líneas y puntos aprendidos, además, es el sitio en que se empiezan a aprender formas más complejas, y finalmente, llega la activación de la capa de salida que es la encargada de devolver el resultado de la inferencia. Todo este proceso proporciona un aprendizaje de menos a más, como si de una persona se tratara, empezando desde lo básico, por ejemplo, el entendimiento de los números, se avanzase por las operaciones básicas de sumas, restas, multiplicaciones y divisiones, y se llegase en una primera instancia a las ecuaciones.

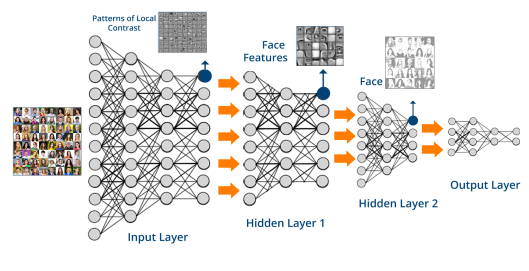
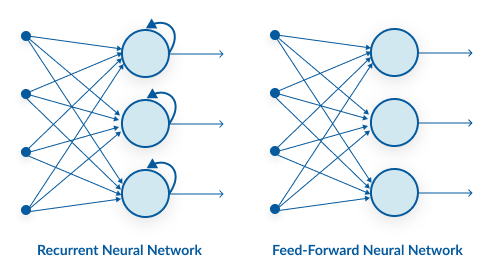


Figura 5 - Etapas del aprendizaje por capas

El DL es un proceso que lleva mucho tiempo y que, para obtener muy buenos resultados, requiere de mucha experiencia en ciencia de datos y, principalmente podemos clasificarlo en 2 de los tipos de redes más usadas:

* Convolutional Neural Networks (CNN) o Feed Forward Neural Networks, que son redes neuronales que han sido diseñadas para poder procesar matrices estructuradas, como podrían ser las imágenes, y que tienen como objetivo clasificar fotografías y videos entre otros, gracias al reconocimiento de patrones.
* Recurrent Neural Networks (RNN), que son redes neuronales que usan datos secuenciales o series de tiempo. Tienen como objetivo solucionar problemas como la traducción de idiomas, el reconocimiento por voz y, el procesamiento del lenguaje natural (NLP).

Para nuestro caso, utilizaremos la primera de ellas, es decir, la CNN, puesto que están orientadas a clasificación de imágenes mediante el reconocimiento y aprendizaje de patrones, siendo las RNN tecnología usada por las conocidas Siri o Alexa.

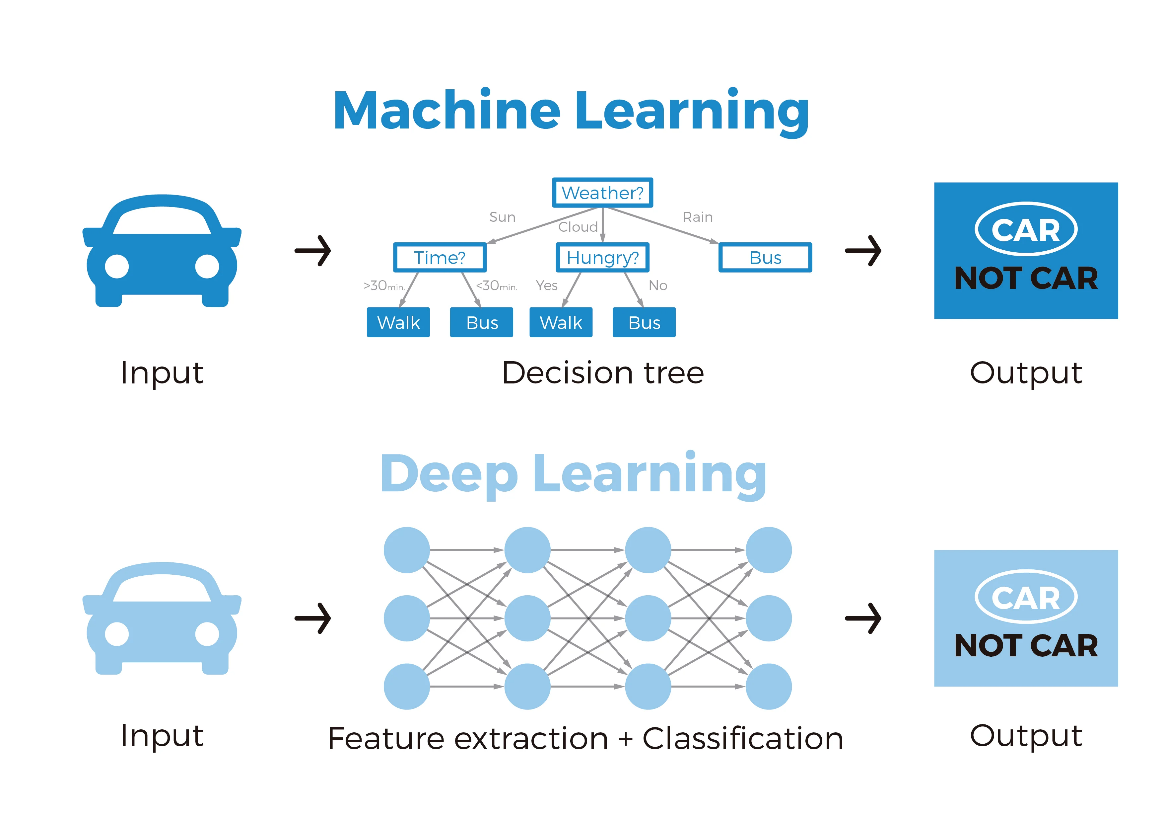
Figura 6 - RNN vs CNN

En cuanto a las plataformas de Deep Learning más comunes, hay dos que son las más destacadas: TensorFlow y PyTorch. Estas dos plataformas están en una constante competición por mejorar, aunque encontramos TensorFlow más fácil de usar, integrado con Keras, y será el marco que usaremos para la creación y entrenamiento de modelos.

* 1. Machine Learning vs Deep Learning

Dado que el aprendizaje profundo y el aprendizaje automático tienden a usarse indistintamente, vale la pena volver a recordar los matices entre ambos. El aprendizaje automático y el aprendizaje profundo son subcampos de la inteligencia artificial, y el aprendizaje profundo, es en realidad un subcampo del aprendizaje automático.

La diferencia entre el aprendizaje profundo y el aprendizaje automático radica en la forma en que cada algoritmo aprende. El aprendizaje profundo automatiza gran parte de la parte de extracción de características del proceso, eliminando parte de la intervención humana manual necesaria, permitiendo el uso de conjuntos de datos más grandes. Se puede considerar el aprendizaje profundo como un "aprendizaje automático escalable". El aprendizaje automático clásico, o "no profundo", depende más de la intervención humana para aprender. Los expertos humanos determinan el conjunto de características para entender las diferencias entre las entradas de datos, lo que suele requerir datos más estructurados para aprender pero que llegan a mejores resultados.

Figura 7 - Diferenciación entre Machine Learning y Deep Learning

El aprendizaje automático profundo puede aprovechar conjuntos de datos etiquetados, también conocidos como aprendizaje supervisado, para informar a su algoritmo, pero no requiere necesariamente un conjunto de datos etiquetados. Puede recibir datos no estructurados en su forma bruta (por ejemplo, texto o imágenes) y determinar automáticamente el conjunto de características que distinguen las diferentes categorías de datos entre sí.

Aunque, tanto al aprendizaje profundo como a las redes neuronales se les atribuye principalmente la aceleración del progreso en áreas como la visión por ordenador, el procesamiento del lenguaje natural y el reconocimiento del habla.

* 1. Motivación

Gracias a la globalización en la que vivimos, podemos crear modelos de forma colaborativa que mejoren los resultados y que sean independientes de un gran servidor, aportando buenos resultados desde el inicio de forma rápida. Esto, unido a la capacidad de crear modelos de Deep Learning (DL) de forma sencilla con las librerías de Keras, junto con las ganas de aportar aplicaciones que ayuden a la población, nos dan como resultado una aplicación cuyo modelo busca asistir a personas con discapacidad, con el objetivo de poder proporcionarles una vida un poco más fácil, la cual, gracias a la informática y, en concreto a la Inteligencia Artificial (IA), es posible.

* 1. Objetivos

El objetivo de este trabajo de fin de master es crear y entrenar un modelo de DL con la finalidad de apoyar a aquellas personas con discapacidad visual. Para que, al utilizar el modelo DL en un dispositivo, en nuestro caso una Raspberry, vaya avisando de forma sonora, en el caso de desviar su paso hacia una carretera, teniendo en cuenta los pasos de cebra como punto de referencia para poder cruzar. Además, trabajarán numerosos dispositivos de forma distribuida simulada, donde cada cual tendrá su propio modelo e imágenes con las que entrenar. Tales modelos serán parte de una agregación federada y redistribuidos para seguir aprendiendo, ayudando a los modelos a generalizar mejor evitando sesgos por las imágenes de entrenamiento propias.

Capítulo 2. Preliminares:

El campo del Aprendizaje Automático está sumergido en una constante evolución, alternando épocas en las que el avance es más rápido con otras en las que es más lento, muchas veces debido a la llegada de otras tecnologías que habilitan esos pasos adelante y, este fue el caso para la Visión artificial. En este capítulo, se describirán conceptos para poder entender el modelo propuesto de visión artificial, así como sus elementos agregados.

* 1. Visión artificial

La visión por ordenador es un campo de la inteligencia artificial que permite a los ordenadores y sistemas obtener información significativa a partir de imágenes digitales, vídeos y otras entradas visuales, con el objetivo de emprender acciones o hacer recomendaciones basadas en esa información. Si la IA permite a los ordenadores pensar, la visión por ordenador les permite ver, observar y comprender.

La visión por ordenador necesita muchos datos. Analiza los datos una y otra vez hasta que discierne las diferencias y, finalmente, reconoce las imágenes. Por ejemplo, para entrenar a un ordenador en el reconocimiento de neumáticos, es necesario alimentarlo con grandes cantidades de imágenes de neumáticos y elementos relacionados con ellos para que aprenda las diferencias y reconozca un neumático, especialmente uno sin defectos.

Para nuestro caso, tenemos que obtener grandes cantidades de fotos sobre pasos de cebras y carreteras, y que previamente al entrenamiento han sido etiquetadas para facilitar el aprendizaje.

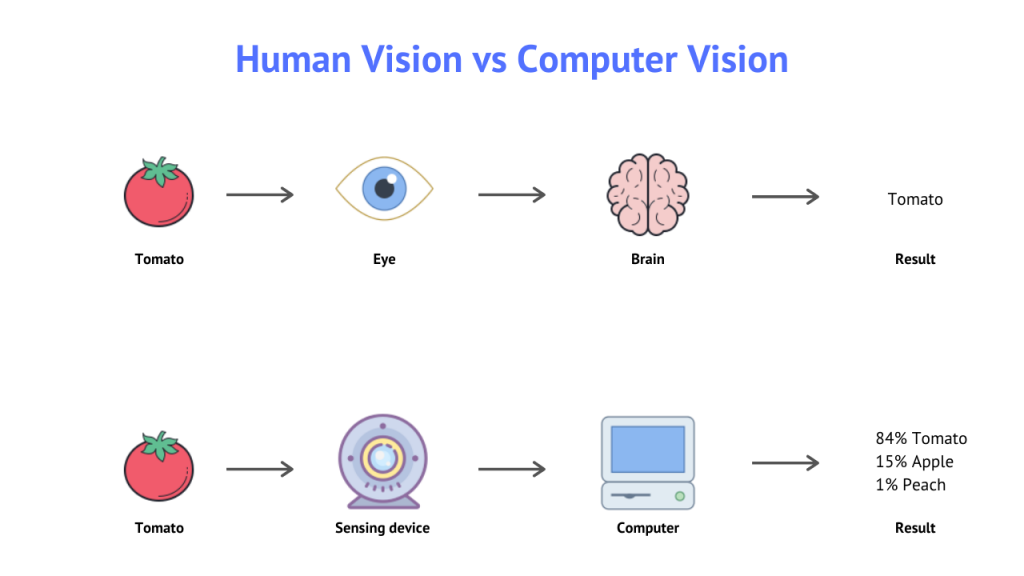
Todo esto es necesario para que tenga los suficientes ejemplos para generalizar y obtener un porcentaje de precisión más alto a la hora de realizar la inferencia.

Figura 8 - Ejemplo de visión artificial

* 1. IoT

El internet de las cosas, o IoT, es un sistema de dispositivos informáticos, máquinas mecánicas y digitales, objetos, animales o personas interrelacionados que están dotados de identificadores únicos (UID) y de la capacidad de transferir datos a través de una red sin que sea necesaria la interacción de persona a persona o de persona a ordenador.

El Internet de las cosas puede ser una persona con un implante de monitor cardíaco o cualquier otro objeto natural o artificial, que sea capaz de transferir datos a través de una red, como los relojes inteligentes o SmartWatchs.

El IoT también puede hacer uso de la inteligencia artificial (IA) y el aprendizaje automático para ayudar a que los procesos de recopilación de datos sean más fáciles y dinámicos.

Algunas de las ventajas del IoT son las siguientes:

* Capacidad de acceder a la información desde cualquier lugar y en cualquier momento en cualquier dispositivo
* Mejora de la comunicación entre los dispositivos electrónicos conectados
* Transferencia de paquetes de datos a través de una red conectada que ahorra tiempo y dinero
* Automatización de tareas que ayudan a mejorar la calidad de los servicios de una empresa y reducen la necesidad de intervención humana.

Algunas de las desventajas:

* A medida que aumenta el número de dispositivos conectados y se comparte más información entre ellos, también aumenta la posibilidad de que un hacker pueda robar información confidencial.
* Las empresas pueden llegar a tener que lidiar con un número masivo -quizás incluso millones- de dispositivos IoT, y recopilar y gestionar los datos de todos esos dispositivos no es una tarea sencilla.
* Si hay un error en el sistema, es probable que todos los dispositivos conectados se corrompan.
* Dado que no existe un estándar internacional de compatibilidad para el IoT, es difícil que los dispositivos de diferentes fabricantes se comuniquen entre sí.

En lo que respecta a nuestra propuesta, el IoT formaría parte de la implementación de nuestros dispositivos Raspberry con su Cámara incorporada, donde este procesaría el vídeo en tiempo real para resolver el problema y cuyo modelo sería compartido con los demás con el objetivo de seguir mejorando de forma común, gracias al envío de estos datos a un servidor encarga posteriormente de combinarlos según unas ciertas reglas.

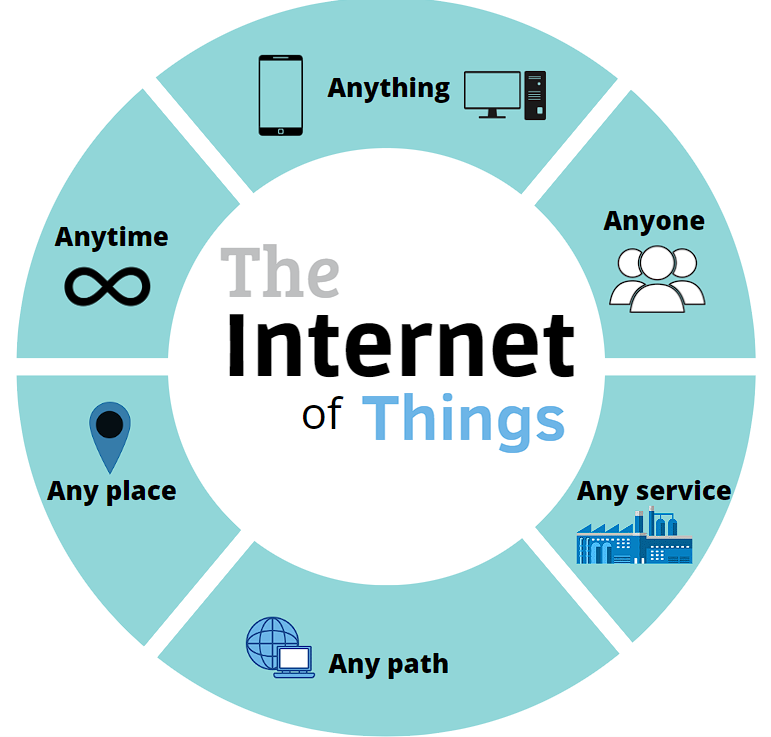


Figura 9 – Características del internet de las cosas (IoT)

* 1. Edge Computing

El Edge Computing es una arquitectura de tecnología de la información (TI) distribuida en la que los datos del cliente se procesan en el borde de la red, lo más cerca posible de la fuente de origen.

En términos sencillos, el Edge Computing traslada una parte de los recursos de almacenamiento y computación fuera del centro de datos central (servidor) y los acerca a la fuente de los datos (Para nuestro caso, el dispositivo Raspberry). En lugar de transmitir los datos en bruto a un centro de datos central para su procesamiento y análisis, ese trabajo se realiza en el lugar donde se generan los datos, evitando su paso a través de la red y otorgando más privacidad y seguridad a los mismos.

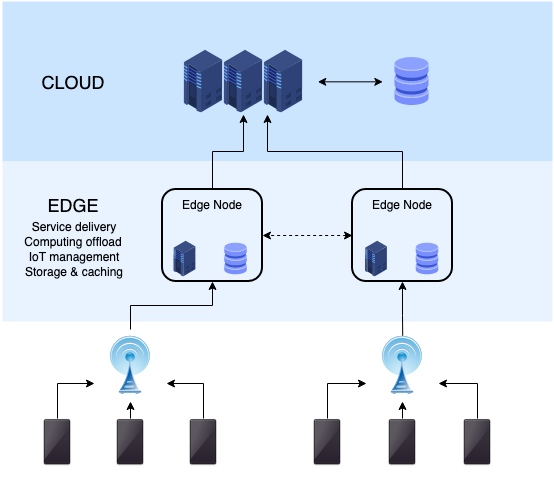
Todo esto surge debido a que el número de dispositivos que generan datos para ser tratados crece de forma exponencial, más rápido que la infraestructura tradicional de los centros de datos puedan acomodarse.

Figura 10 - Ejemplo de Edge Computing

* 1. Redes Convolucionales

Existen varios tipos de redes neuronales, las cuales se utilizan para diferentes casos de uso y tipos de datos. Por ejemplo, las redes neuronales recurrentes (RNN) se utilizan habitualmente para el procesamiento del lenguaje natural y el reconocimiento del habla, mientras que las redes neuronales convolucionales (ConvNets o CNN) se utilizan más a menudo para tareas de clasificación y visión por ordenador. Antes de las CNN, se utilizaban métodos de extracción de características manuales y lentos para identificar objetos en las imágenes. Sin embargo, las redes neuronales convolucionales proporcionan ahora un enfoque más escalable para las tareas de clasificación de imágenes y reconocimiento de objetos (que será la tarea principal para este proyecto de detección de pasos de peatones y carreteras), aprovechando los principios del álgebra lineal, concretamente la multiplicación de matrices, para identificar patrones dentro de una imagen. Sin embargo, pueden ser muy exigentes desde el punto de vista computacional y puede requerir de unidades de procesamiento gráfico (GPU) para entrenar los modelos de forma más eficiente.

Las redes neuronales convolucionales se distinguen de otras redes neuronales por su rendimiento superior con entradas de señales de imagen, habla o audio. Tienen tres tipos principales de capas, que son:

1. Capa convolucional: La capa convolucional es la primera capa de una red convolucional y pueden ir seguidas de otras capas convolucionales o capas de agrupación.

Además, es el bloque central de una CNN y es donde se produce la mayor parte del cálculo. Requiere unos pocos componentes, que son los datos de entrada, un filtro que es el encargado de detectar características y un mapa de características.

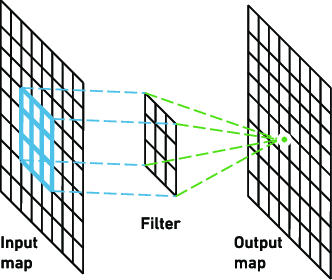
Hay que tener en cuenta que los pesos del detector de características permanecen fijos a medida que se desplaza por la imagen, lo que también se conoce como compartición de parámetros. Algunos parámetros, como los valores de los pesos, se ajustan durante el entrenamiento mediante el proceso de retro propagación o back propagation que es el algoritmo que se encarga de actualizar estos pesos de la red neuronal, en función a los resultados obtenidos y el descenso de gradiente.

Figura 11 - Capa convolucional

1. Capa de agrupación: Las capas pooling, también conocidas como downsampling, realizan una reducción de la dimensionalidad, reduciendo el número de parámetros en la entrada. De forma similar a la capa convolucional, la operación de pooling barre un filtro a través de toda la entrada, pero la diferencia es que este filtro no tiene pesos. En su lugar, el núcleo aplica una función de agregación a los valores dentro del campo receptivo, poblando la matriz de salida. Hay dos tipos principales de agrupación:

* Max pooling: A medida que el filtro se desplaza por la entrada, selecciona el píxel con el valor máximo para enviarlo a la matriz de salida. Este método se utiliza más a menudo que el pooling medio.
* Combinación de promedios: A medida que el filtro se desplaza por la entrada, calcula el valor medio dentro del campo receptivo para enviarlo a la matriz de salida.

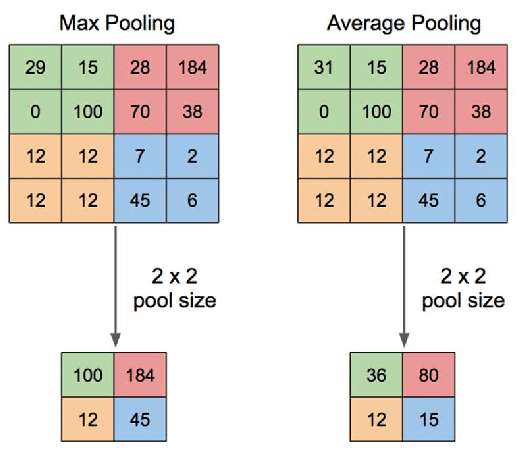
Aunque se pierde mucha información en la capa de pooling, ayuda a reducir la complejidad, mejora la eficiencia y limita el riesgo de sobreajuste.

Figura 12 - Capa pooling

1. Capa totalmente conectada (FC): Los valores de los píxeles de la imagen de entrada no se conectan directamente a la capa de salida en las capas parcialmente conectadas. Sin embargo, en la capa totalmente conectada, cada nodo de la capa de salida se conecta directamente a un nodo de la capa anterior.

Esta capa realiza la tarea de clasificación basándose en las características extraídas a través de las capas anteriores y sus diferentes filtros. Mientras que las capas convolucionales y de agrupación suelen utilizar funciones ReLu, las capas FC suelen aprovechar una función de activación SoftMax para clasificar las entradas de forma adecuada, produciendo una probabilidad de 0 a 1.

Con cada capa, la CNN aumenta su complejidad, identificando mayores partes de la imagen. Las primeras capas se centran en características simples, como los colores y los bordes. A medida que los datos de la imagen avanzan por las capas de la CNN, ésta empieza a reconocer elementos o formas más grandes del objeto hasta que finalmente identifica el objeto deseado.

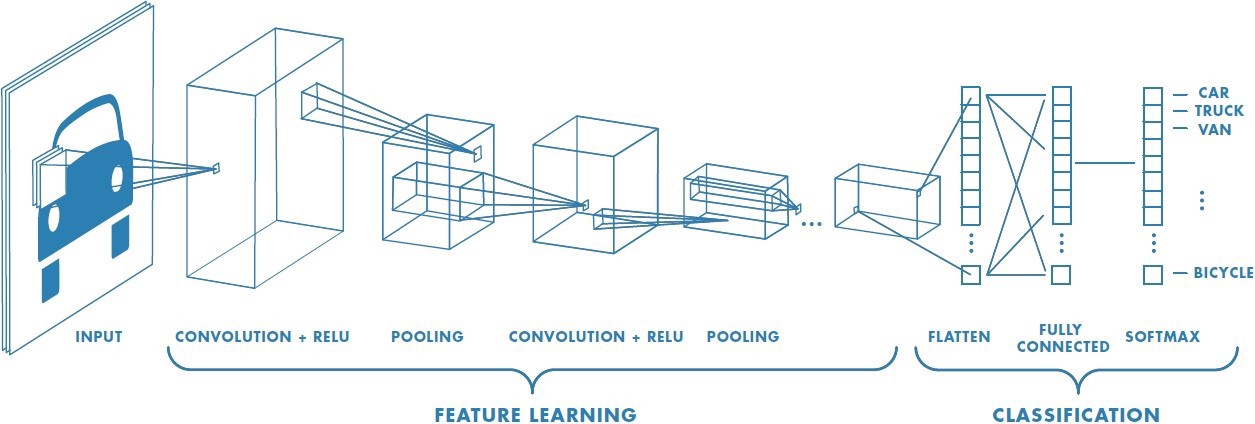


Figura 13 - WorkFlow del proceso de una red neuronal

* 1. Transfer Learning

El Transfer Learning o Aprendizaje por Transferencia es un método de aprendizaje automático en el que reutilizamos un modelo previamente entrenado como punto de partida para un modelo en una nueva tarea.

En pocas palabras, un modelo entrenado en una tarea se reutiliza en una segunda tarea relacionada como una optimización que permite un rápido progreso al modelar la segunda tarea.

Al aplicar el aprendizaje de transferencia a una nueva tarea, se puede lograr un rendimiento significativamente mayor que el entrenamiento con una pequeña cantidad de datos y en menos tiempo que entrenando un modelo desde cero, de hecho, el Transfer Learning es tan común, que es raro crear un nuevo modelo al que hay que entrenar desde el inicio para una tarea relacionada con el procesamiento de imágenes o del lenguaje natural.

En su lugar, los investigadores, desarrolladores y científicos de datos prefieren partir de un modelo preentrenado que ya sabe clasificar objetos, y ha aprendido características generales como los bordes o las formas en las imágenes.



Figura 14 - Campos de uso del Transfer Learning actualmente

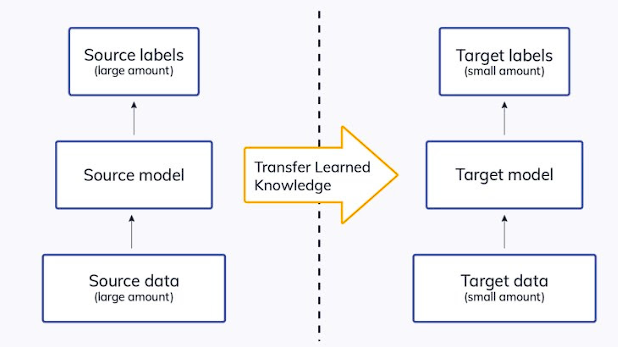
MobileNet, VGG16 e Inception, son ejemplos típicos de modelos que tienen como base el aprendizaje por transferencia, aunque a día de hoy existen muchísimos, los cuales podemos encontrar en la página oficial de Keras [12].

Figura 15 - Funcionamiento esquemático del Transfer Learning

Concretamente, MobileNetV2 es una arquitectura de red neuronal convolucional que busca un buen rendimiento en los dispositivos móviles, aunque es de los más utilizados en dispositivos con baja capacidad computacional, por lo tanto, es ideal para ayudarnos a resolver nuestro problema, proporcionando facilidad no solo a conseguir modelos buenos en menos tiempo y evitando necesitar muchísimas imágenes de entrenamiento, sino que además, nos ayuda a no tener que “reinventar la rueda” creando una arquitectura nueva para el modelo que necesitamos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modelo | Tamaño (MB) | Top1 Accuracy | # Parámetros | Tiempo (ms) de inferencia por paso (CPU) |
| VGG16 | 528 | 71.3% | 138.4M | 69.5 |
| ResNet50 | 98 | 74.9% | 25.6M | 58.2 |
| InceptionV3 | 92 | 77.9% | 23.9M | 42.2 |
| MobileNet | 16 | 70.4% | 4.3M | 22.6 |
| MobileNetV2 | 14 | 71.3% | 3.5M | 22.6 |

En cuanto a las características que tiene este modelo frente a su predecesor, como frente a otros modelos (*ver* Figura 8), es que cuenta con menos tamaño, por lo que la memoria del dispositivo no será un problema y, además cuenta con una baja cantidad de parámetros y mayor accuracy, lo que proporcionará también menos tiempo de entrenamiento en los steps y épocas.

Figura 16 - Tabla de principales modelos preentrenados

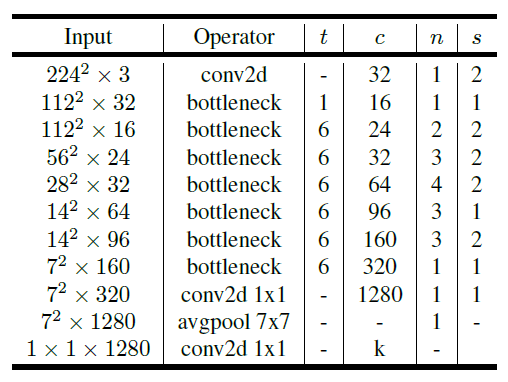
MobileNetV2 se basa en una estructura residual invertida en la que las conexiones residuales se encuentran entre las capas de cuello de botella. La capa intermedia de expansión utiliza convoluciones ligeras en profundidad para filtrar características. En conjunto, la arquitectura de MobileNetV2 contiene la capa inicial de convolución completa con 32 filtros, seguida de 17 capas residuales de cuello de botella [San18].

Figura 17 - Arquitectura general de MobileNetV2

* 1. El Aprendizaje Federado

Debido a la capacidad de cómputo que tienen los dispositivos más pequeños hoy en día, como las Raspberry-Pi o nuestros propios smartphones al cual damos uso diariamente, así como la privacidad de los datos, las arquitecturas federadas han ganado importancia.

El Aprendizaje Federado “Federated Learning: Collaborative Machine Learning without Centralized Training Data” [McM17] (en adelante FL de sus siglas en inglés), otorga la capacidad de aprendizaje de modelos de forma colaborativa entre diferentes dispositivos, sin tener que traspasar los datos de aprendizaje a un servidor.

Para nuestros modelos, usaremos el Accuracy como medida de comprobación para saber que tan bueno ha sido su aprendizaje, así como el paso a un servidor del propio modelo completo, sin imágenes para entrenar.

Este aprendizaje federado mejora la fiabilidad de los modelos, donde cada dispositivo tiene su propio modelo local que ha entrenado con sus propios datos que no serán comprometidos enviándolos a un servidor, si no que será el propio modelo el que sea enviado para que el servidor cree, entre todos los modelos enviados de los dispositivos asociados, uno finalmente globalizado y mejor (un modelo basado en la agregación de pesos de forma federada), que mejore su precisión y, por lo tanto, la fiabilidad del mismo. A este método de agregación se le conoce como Federated Averaging (FedAvg) o Media Federada, el cual se basa en una simple media aritmética.

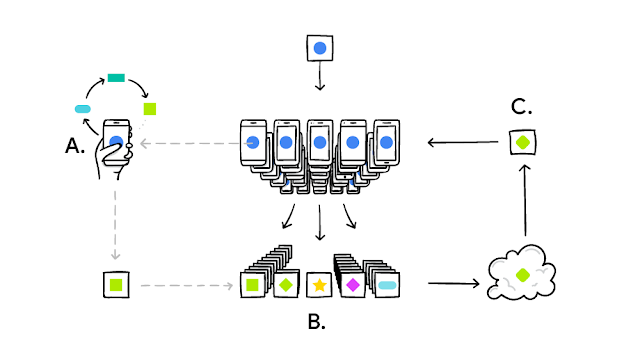
Con esta técnica, Google implementó su teclado predictivo, el GBoard el cual, como primera prueba real, fue un éxito.

Figura 18 - Sistema de Aprendizaje Federado de Google (Android GBoard)

Sin embargo, y aunque los datos no son enviados al servidor, estos modelos son enviados y, en el caso de modelos necesiten mantener su privacidad, se pueden utilizar diferentes técnicas [Ma20, Moh17, Gey18, Aca18], donde se usaría una comunicación encriptada [Bon17] y asíncrona dotando de mayor robustez al sistema.

O, inclusive, utilizar otras técnicas de agregación que mejoren los tiempos de convergencia y, su precisión, como pueden ser el método WAFFLE (Weighted Averaging for Personalized Federated Learning) [Bea21] o MHAT (Model-Heterogenous Aggregation Training) [Hu21] entre otros, frente al Federated Averaging que propone Google.

En cuanto a sus usos, como se ha comentado anteriormente, son para aquellos donde la privacidad y la seguridad es esencial [Yan19], como puede ser en la predicción de enfermedades a través de datos de las redes hospitalarias, donde se beneficiarían sobre todo aquellos hospitales con menos capacidades, con acceso a información de ayuda más precisa y de nivel experto.

Finalmente, FL abre un nuevo abanico de posibilidades, donde todos los dispositivos puedan compartir sus modelos y, donde los métodos de agregación de modelos están en continuo desarrollo, el cual juega un papel sumamente importante en la mejora de la fiabilidad de los modelos posteriormente distribuidos y en la reducción de computación en el lado del servidor junto con los costes que ello conlleva.

* 1. Edge TPU. Coral USB Acelerator

El Acelerador USB de Coral, es un tipo de acelerador de hardware, orientado específicamente a optimizar y mejorar las cargas de trabajo de machine Learning que han sido generadas con TensorFlow, es decir, tiene como objetivo principal mejorar la inferencia que se lleva acabo de los modelos de IA.

Figura 19 - Coral USB Acelerator

Entonces Coral, nos ayuda a llevar a nuestras aplicaciones de IA hacia una aplicación final, ofreciendo modelos precompilados o librerías como la de PyCoral, que proporciona más facilidades y, menores tiempos, a la hora de hacer las inferencias. Sin embargo, solo admite por el momento, modelos de TensorFlow Lite (tflite), es decir, modelos de TensorFlow reducidos.

Por lo tanto, quiere decir que no se pueden entrenar modelos directamente con Coral AI, sino que trabajará a partir de un modelo de tflite, el cual ya estará entrenado y convertido a tflite previamente.

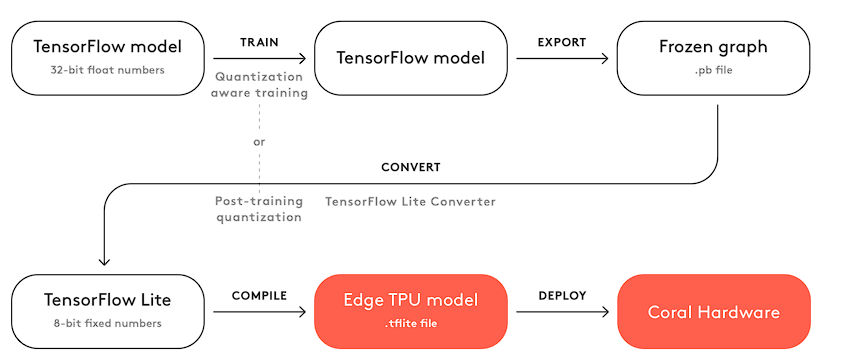


Figura 20 - Proceso de creación de modelo compatible con TPU Edge

Algunos de los usos más populares que nos encontramos a día de hoy, suelen basarse en la visión por ordenador y el Edge Deep Learning como pueden ser:

* Detección de objetos y personas mediante vídeo en tiempo real.
* Estimación de poses de personas u objetos a partir del seguimiento de puntos clave.
* Segmentación de imágenes para identificar objetos y ubicaciones en tiempo real.

Particularmente para nuestro problema nos encontramos en Edge Deep Learning, donde buscaremos procesar las imágenes de un video para ver si nos adentramos a un paso de cebra o vamos directos a una carretera, lo cual es un peligro que sería necesario detectar lo más rápido posible.

1. Funcion Cargar\_Imagen(Ruta):
3. CargarImagen()
4. ImagenTensor = ConvertirImagenATensor()
6. Devolver ImagenTensor
8. InicializarInterpreteEDGETPU(Modelo)
9. RedimensionarImagen()

12. Loop Imagenes
13. InvocarInterprete(Interprete,Imagen)
14. Clase = ObtenerClase(Interpreter)
15. MostrarResultadoPorPantalla()

Pseudocódigo 1 - Inferencia con Coral

* 1. TensorFlow como herramienta para el Machine-Learning

TensorFlow, desarrollado por Google y open-source, se ha consolidado como una de las librerías más importantes de Machine-Learning (ML), con una grandísima comunidad con multitud de recursos y tutoriales de acceso gratuito del propio Google, que permite a desarrolladores e investigadores a crear e implementar aplicaciones de Aprendizaje Automático (AA) de manera más rápida y sencilla con su implementación con Keras, el cual es una API de alto nivel de fácil uso.

Actualmente, se centra en la creación y entrenamiento de redes neuronales, las cuales pueden detectar y descifrar correlaciones entre variables, como pasaría en el razonamiento y aprendizaje humano.

Igualmente, con su facilidad de uso, da la capacidad de implementaciones del cálculo con más de una CPU o GPU, e incluso, se puede usar TPU (Tensor Processing Unit), que es un acelerador de Inteligencia Artificial (IA) el cual mejora el rendimiento al usar los modelos en los dispositivos para que la salida y, evaluación de la red neuronal, sea aún más rápida.

Para nuestro caso, usaremos en el dispositivo Raspberry el Acelerador USB TPU de Coral, para que la inferencia de nuestras imágenes sea lo más rápida posible y dar más seguridad y eficiencia a nuestra aplicación final que hace uso del modelo entrenado, a partir de TensorFlow Lite (tflite) que es una versión ligera de TensorFlow, que proporciona un modelo reducido con el que trabajaremos en la aplicación final para la inferencia, debido a que se consigue una inferencia de baja latencia.

Por lo tanto, a día de hoy, muchas de las aplicaciones que utilizamos como herramientas en nuestro uso diario y, sin percatarnos de ello, utilizan modelos creados y entrenados con TensorFlow, como podrían ser la mejora en las fotos que tomamos con nuestros smartphones, el teclado predictivo de nuestros móviles o, incluso el marketing personalizado, donde los anuncios que vemos dependen del uso y búsquedas que realizamos.

Figura 21 - Características de TensorFlow

* 1. GitHub para desarrollo

Una de las principales necesidades que teníamos al inicio del desarrollo de todo este proyecto de fin de máster ha sido el cómo podíamos organizar el desarrollo de código en paralelo, sin perder actualizaciones y siendo una forma ágil para probarlo dentro de nuestra Raspberry Pi.

Para dar solución a esta necesidad, decidimos hacer uso de Git para tener un control de versiones y poder hacer uniones del código desarrollado en paralelo.

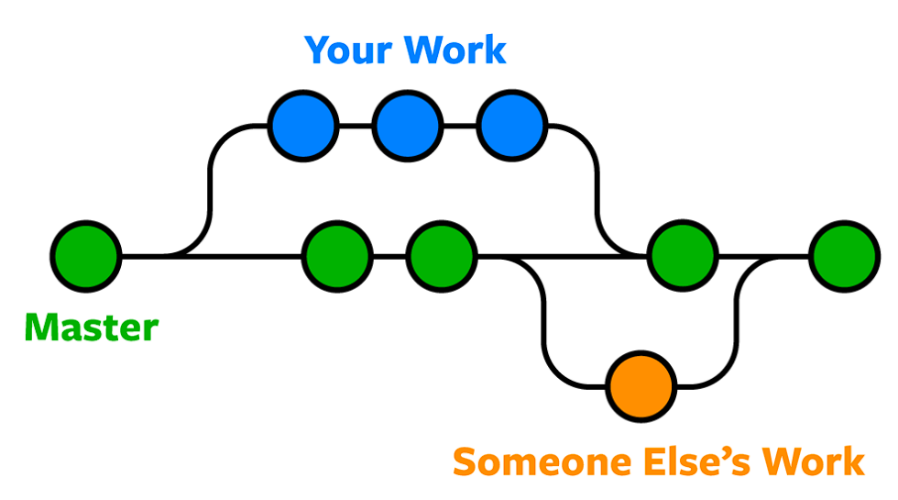
Git es el sistema de control de versiones más utilizado, el cual te permite hacer un seguimiento de los cambios que se realizan en los archivos, de modo que se tiene un registro de lo que se ha hecho y, se puede volver a versiones específicas si se necesita. Git también facilita la colaboración, ya que permite que los cambios realizados por varias personas se fusionen en una sola fuente.

Figura 22 - Ejemplo de GitHub

De esta forma inicializamos un repositorio para el proyecto, donde poder subir las diferentes versiones y poder hacer un pull dentro de la Raspberry Pi para poder ejecutar el código.

Los principales comandos que hemos utilizado han sido:

* 1. **GIT INIT**: Se utiliza para inicializar un nuevo repositorio Git vacío
  2. **GIT CLONE**: A menudo, si se tiene un repositorio Git existente (a veces alojado en un sitio como GitHub o Bitbucket) y quieres copiarlo a tu máquina local. Se usa el comando git clone.
  3. **GIT STATUS**: Git siempre está pendiente de los cambios en el directorio de trabajo del proyecto. Esto incluye cambios como la creación de un nuevo archivo, la adición de un archivo para su seguimiento, la eliminación de un archivo, el cambio de permisos de un archivo, la modificación del nombre o el contenido de un archivo, etc.
  4. **GIT ADD**: Cuando se realiza alguno cambio en un archivo del directorio de trabajo y se confirme que son correctos con el comando git status. Esto añade el cambio al área de staging de Git.
  5. **GIT COMMIT**: Cuando se tienen los cambios preparados, se usa el comando git commit para guardar esos cambios en el repositorio Git. Un commit de Git es un conjunto de cambios de archivos que se almacenan en Git como una unidad.
  6. **GIT BRANCH**: Puedes pensar en una rama de Git como una cadena de commits o una línea de desarrollo. El nombre de una rama es sólo una etiqueta que apunta a un ID de confirmación específico. Cada ID de confirmación está vinculado a su ID de confirmación padre, lo que forma una cadena de historia de desarrollo.
  7. **GIT CHEKOUT**: El comando git checkout te permite saltar (cambiar) entre diferentes ramas actualizando tu directorio de trabajo para reflejar la punta de la rama chequeada.
  8. **GIT MERGE**: permite tomar las líneas independientes de desarrollo creadas por git branch e integrarlas en una sola rama
  9. **GIT PUSH**: Hasta ahora, todos los comandos que hemos ejecutado sólo han afectado al entorno local. Este comando permite confirmar el código con los demás desarrolladores, empujándolos al repositorio remoto (a menudo alojado en sitios como GitHub y Bitbucket).
  10. **GIT PULL**: El comando git pull es justo lo contrario de git push. Puedes utilizarlo para descargar los cambios realizados por otros desarrolladores en tu repositorio local.

Capítulo 3. Modelo propuesto

En este capítulo, se explicará cómo se ha puesto en marcha el modelo de Deep Learning y la Arquitectura federada, con el objetivo de explicar el funcionamiento del sistema federado desarrollado y las partes que lo componen.

2. 1. Modelo básico no federado

Para entender como se ha creado el modelo, primero vamos a conocer y entender un poco que es Keras, el cuál es un Framework de alto nivel y open source para AA, con el objetivo de facilitar y otorgar más velocidad a todo el proceso de creación, entrenamiento y experimentación.

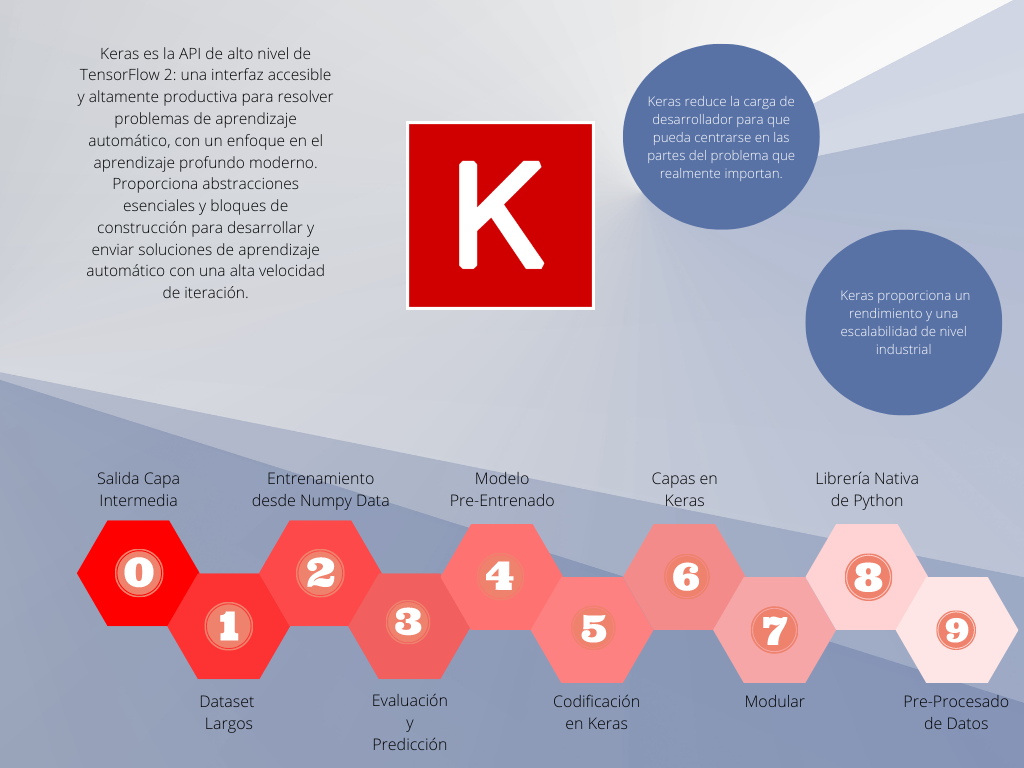
Teniendo en cuenta todo lo que nos ofrece Keras, lo que se pretende es que, a través de un modelo ya creado y preentrenado como MobileNetV2, del que hemos hablado, nos proporcione la capacidad de entrenar con incluso mayor rapidez y partiendo de una buena base con una arquitectura creada, tanto para tiempos de inferencia, como para entrenamientos, evitando los más que posibles estancamientos de una red propia.

Figura 23 - Características de Keras

Nuestro modelo, estará formado por todas las capas del modelo MobileNetV2, con la diferencia de que eliminaremos la última capa de salida y seremos nosotros quienes introduciremos 3 capas unidas a la penúltima capa del modelo preentrenado.

Además, también se establecerá la primera capa de entrada, donde encontramos el primer punto de flexibilidad, ya que tenemos que decidir según lo que nos convenga. Esto es debido a que, si las entradas de la red son pequeñas, implica que de forma genérica va a ir más rápido, pero perdemos calidad de imagen. Entonces hay que encontrar un balance que se nos ajuste al dispositivo que va a ejecutar este modelo para realizar la inferencia para que funcione de una forma eficiente.

* 1. La Arquitectura Federada

La creación de la arquitectura federada, así como de sus componentes será clave para realizar un proceso de entrenamiento lo más optimo posible de forma simulada.

La simulación es debida a que los sistemas federados necesitan de diferentes nodos de entrenamiento. Tales nodos son nuestros dispositivos que en el caso que nos ocupa son las Raspberry, por lo tanto, no disponer de numerosos dispositivos dificulta un proceso real con comunicaciones reales a un servidor de los modelos.

Es por ello que se ha realizado una simulación en Python de toda la etapa de entrenamiento federada, con diferente número de dispositivos y que debido a nuestro reducido número de Cores en la Raspberry (4, de los cuales solo tendríamos 3 disponibles ya que uno se lo llevaría el OS), se ha realizado de forma secuencial, sin entrar en la creación de hilos pues el tiempo de desarrollo del sistema federado se dispararía y, no genera ninguna ventaja más que una reducción del tiempo de entrenamiento para el conjunto total de dispositivos, ya que se trata de una simulación y no un proceso real.

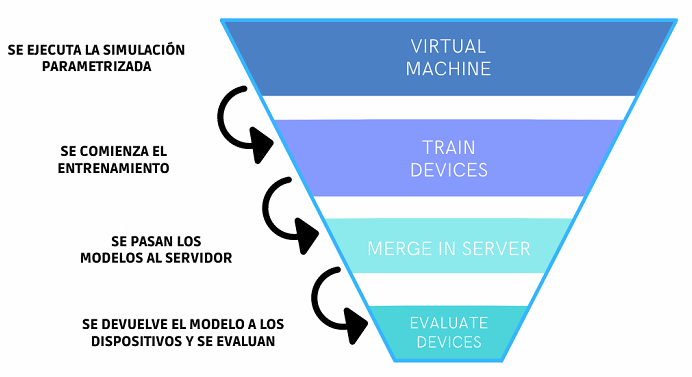
Cuando hablamos de simulación secuencial, quiere decir que, nuestros dispositivos se ejecutan todos en una misma Raspberry, y que cuando un dispositivo termina su entrenamiento, comenzaría el siguiente hasta acabar con los dispositivos planeados en el entrenamiento.

Figura 24 - Esquema de Arquitectura Federada básica

Como Arquitectura federada, podemos pensar básicamente en 4 etapas:

1. La puesta en marcha de todos los dispositivos implicados, así como su conexión al servidor.
2. La ejecución de todos los dispositivos de su etapa de entrenamiento.
3. El paso de los modelos entrenados de los dispositivos al Servidor, que será el encargado de agregar/fusionar los modelos.
4. La evaluación de los dispositivos del modelo generado por el servidor, así como su posterior guardado o desecho dependiendo de la calidad del nuevo modelo.

Sin embargo, cada una de estas etapas tiene de forma interna un flujo más definido, en el cuál, en una primera instancia, en la etapa de inicialización, la simulación tiene que contener el número de dispositivos que estarán presentes, así como la elección del modelo inicial y, el tipo de agregación federada, que para nuestro caso podría ser la Media Federada (FedAvg) o la Media Ponderada Federada.

Posteriormente, la Máquina Virtual o Virtual Machine (VM), pasará los parámetros necesarios a los dispositivos, tales como el ancho y alto de la imagen que tendrá que procesar, el dispositivo con el que tendrá que empezar, o su porcentaje de datos para entrenamiento.

1. Funcion ProcesadoImagenes(Ruta\_Dataset):
2. Loop Ruta\_Dataset:
3. AplicarSharp(Imagen\_Ruta)
5. Funcion Main():
6. Si(Primera\_Vez):
7. print("Creando carpetas de dispositivo")
8. CrearCarpetaImagenes()
9. Loop Imagenes\_Carreteras:
10. Renombrar("road\_"+i.jpg)
11. CopiarEnRutaDestino(imagen)
12. Loop Imagenes\_Pasos\_Cebra:
13. Renombrar("crosswalk\_"+i.jpg)
14. CopiarEnRutaDestino(imagen)
16. ProcesadoImagenes(Ruta\_Dataset)

19. Loop Dias\_Ejecucion:
20. Loop Lista\_Dispositivos:
21. CrearDispositivo(Epocas, Tipo\_Modelo, Batch\_Size ...)
22. Accuracy, Val\_Accuracy, Loss, Val\_Loss = EjecutarDispositivo()
23. GuardarValoresDevueltos(Accuracy, Val\_Accuracy, Loss, Val\_Loss)
25. Si Día es 0:
26. GuardarDiccionarioCSV()
27. Sino:
28. AgregarAlDiccionarionActual()
30. CrearServer(Tipo\_Server)
31. EjecutarMergeServer()
33. Loop Lista\_Dispositivos:
34. CrearDispositivo(Epocas, Tipo\_Modelo, Batch\_Size ...)
35. Cambia\_Modelo?, Accuracy = Dispositivo.EvaluarNuevo()
36. GuardarValoresDevueltos(Cambia\_Modelo?, Accuracy)
38. ResultadoCSV = LeerResultadosCSV()
39. ActualizarDatos(ResultadoCSV)

Pseudocódigo 2 - Máquina Virtual

Además, una vez que los dispositivos empiezan a ejecutar su etapa de entrenamiento, previamente han tenido que seleccionar las imágenes de entrenamiento con las que va a entrenar que, al ser una simulación, estas están almacenadas en la propia Raspberry y de las cuales se hará una distinción basada en lotes, de la que hablaremos próximamente.

1. Clase Device:
3. Funcion Inicializacion(number, path, path\_dataset, data\_percentage, train\_percentage, model\_type, epochs, steps\_per\_epoch, image\_height, image\_width, batch\_size, day):
4. AsignacionDeParametros()
6. Funcion Ejecutar():
7. CargarImagenes()
9. SegmentarTrainSet(TrainData, 20%)
10. CargarDatasetsDeValidacion()
11. Modelo=CargarModeloTipo()
12. CompilarModelo(Optimizador,Perdida,Metricas)
14. Modelo.Entrenar(TrainGenerator,ValidationGenerator,Epocas)
15. GuardarModelo()
16. GuardarConfiguracion(Historial)
18. Devolver ObtenerCabeceraHistorial()
20. Funcion CargarDatosEnRuta():
21. CrearDirectorios()
22. MoverDatosRoad()
23. MoverDatosPasoCebra()
25. Devolver Entrenamiento, Prueba
27. Funcion CargarDatosImagenes(self):
28. Loop DataInPath:
29. AñadirLabel()
31. CogerPorcionTrainDataSet()
32. CogerPorcionTestDataSet()
34. Loop TrainDataSet:
35. Si 'crosswalk'
36. AñadirLabel('crosswalk')
37. sino
38. AñadirLabel('road')
40. Loop TestDataSet:
41. Si 'crosswalk'
42. AñadirLabel('crosswalk')
43. sino
44. AñadirLabel('road')
46. Devolver TrainData, TestData

49. Funcion CargarDataSetDeValidacion(TrainData,EvaluateSet):
50. ReescaladoTrain(escalado = 1./255)
51. ReescaladoEvaluate(escalado = 1./255)
53. Devolver TrainData,EvaluateSet
55. Funcion CargarModeloTipo():
56. Si PrimerDia
57. CargarMobileNetV2(ancho,alto,3)
58. MejorModelo = CompararModelos()
59. Devolver MejorModelo
60. sino
61. ModeloActual = CargarElMismoModelo()
62. Devolver ModeloActual
64. Funcion PintarHistorial(historico):
65. PintarEnPantallaHistorico(historico)
67. Funcion GuardarConfiguracion(historico):
68. AbrirJson(Path)
69. EscribirJson(historico)
71. Funcion Evaluar(path):
73. CargarDatosImagenes\_Nuevo()
75. DividirTrainTest(DataTrain)
77. CargarDatasetsDeValidacion\_new()
79. CargarModelo(path+'/model.h5')
80. CompilarModelo()
81. ObtenerResultados()
83. ObtenerUltimoAccYUltimoValAcc()
85. Si MejorResultado
86. CogerNuevoModelo()
87. GuardarModeloAnterior()
88. Devolver 1
89. sino:
90. Devolver 0

Pseudocódigo 3 – Devices

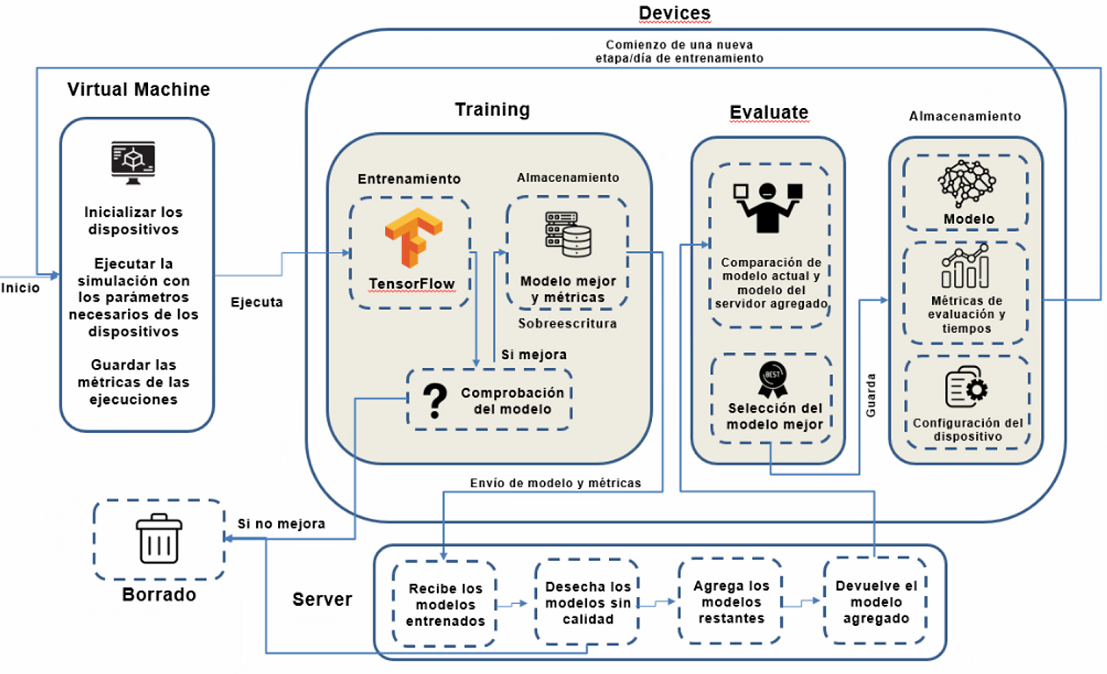
Una vez que el dispositivo ha completado su etapa de entrenamiento y ha guardado sus métricas, tales como el accuracy y el loss (que es el Error cuadrático Medio o MSE de sus siglas en inglés) para su etapa de entrenamiento, su accuracy y loss para su etapa de validación, los tiempos de entrenamiento o las veces que ha cambiado el dispositivo de modelo por el que le ha pasado el servidor, entre otros, el modelo generado es enviado al Servidor, que es el encargado de procesar estos modelos.

Este procesamiento del servidor, en primer lugar desecha aquellos modelos que no han tenido un entrenamiento lo suficientemente satisfactorio como para generalizar su modelo, y que su precisión en los datos de validación sean peores al inicial y, posteriormente, trata de que dependiendo del tipo de agregación, ya sea una media federada o una media ponderada federada, se generen los pesos de cada capa de la red neuronal que conforma nuestro modelo de Deep-Learning, creando un nuevo modelo con la misma arquitectura interna, pero modificando los pesos, solo en aquellas capas entrenables.

1. Clase Servidor:
3. Funcion Inicializacion(Tipo\_Merge):
4. Merge = Tipo\_Merge
6. Funcion Creacion\_Modelo\_Federado(Miembros, Pesos):
7. CalcularNumeroDeLayers()
8. Inicializar\_Array\_Media\_Pesos\_Ponderados()
9. Loop Layers:
10. CogerLayerParaCadaMiembro()
11. Layer.PonderarMediaDePesos()
12. Layer.GuardarPesosPonderados()
13. Modelo = clone\_model(members[0])
14. ObtenerModeloMiembro().SetPesos(MediaPesosPonderados)
15. Modelo.Compilar()
16. Devolver Modelo
18. Funcion Merge(Ruta):
19. tf.keras.backend.clear\_session() # importante para evitar que los archivos temporales se pisen y coja datos de otra ejecución
20. Loop Dispositivos:
21. Loop Archivos in Ruta:
22. Si Historial.json exists:
23. GuardarHistorial()
24. Si Último\_Accuracy\_Evaluacion > Accuracy\_Validacion\_Modelo\_Inicial:
25. AñadirModeloALista(Model)
27. Si Tipo\_Merge==1:
28. CalcularMediaPonderada(ListaDevices)
29. new\_model = CrearNuevoModelo(ListaDevices, Pesos)
31. Sino:
32. CalcularMediaAritmetica(ListaDevices)
33. new\_model = CrearNuevoModelo(ListaDevices, Pesos)
35. GuardarNuevoModelo()

Pseudocódigo 4 – Servidor

Una vez que el Servidor ha terminado de agregar los modelos, este es enviado a todos los dispositivos, por lo que estos ahora necesitarán empezar a ejecutar su etapa de evaluación, donde con sus imágenes de validación, nos permitirán saber si el modelo pasado por el servidor, mejora la evaluación de su modelo actual, almacenando el nuevo modelo en caso de que este tenga un accuracy mayor y que, por lo tanto, sea un modelo más genérico con mayor precisión y fiabilidad, donde una vez terminan los dispositivos, comienzan una nueva etapa/día de entrenamiento.

Figura 25 - Arquitectura Federada con WorkFlow

Capítulo 4. Estudio experimental

La experimentación está basada en ejecuciones secuenciales en un dispositivo Raspberry-Pi, donde se simula el funcionamiento de numerosos dispositivos en una arquitectura federada, con su propio aprendizaje en función a sus imágenes asignadas, tanto para entrenamiento como para test.

1. 1. Arquitectura Raspberry

Para todo el entrenamiento y posterior inferencia, se utilizará una Raspberry-Pi 4 Model B de 4GB de RAM y, 4 Cores a 1.5GHz con Procesador ARM Cortex-A72 que ejecutará un modelo basado en Keras y entrenado con TensorFlow. Esto nos acercará a lo que se conoce como Edge Computing.

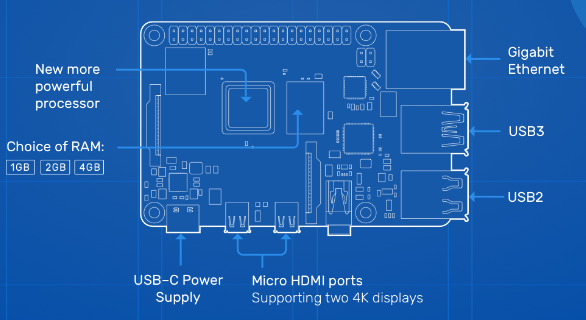
Para ello, se ha tenido que hacer una serie de instalaciones y resolución de problemas típicos que nos encontramos con una arquitectura ARM.

Figura 26 - Arquitectura de Raspberry Pi 4 Model B

* 1. Puesta en marcha del dispositivo e instalación de librerías
* El Sistema Operativo

Para poder hacer uso del dispositivo, comenzamos con la instalación típica del Sistema Operativo (OS de sus siglas en inglés) el cuál podemos realizar desde el instalador Raspberry Pi Imager [27] que nos da a elegir entre diferentes OS, y que, dependiendo de nuestras necesidades seleccionaremos uno u otro [28]. Para el caso que nos ocupa, utilizaremos una versión completa, con aras de facilitar el movimiento por carpetas, las conexiones y las descargas de ficheros, sin embargo, para una versión más rápida y con vistas a una aplicación cerrada final, lo más interesante es una instalación de solo consola, que consuma menos recursos que una instalación completa con escritorio.

Figura 27 - Raspberry Pi Imager

* Las librerías

En cuanto al dispositivo, se ha tenido que realizar una serie de instalaciones de librerías, siendo una de las más importantes TensorFlow. Esta librería es la que nos permitirá la capacidad de entrenar de forma sencilla nuestro modelo de Deep-Learning y que, tal y como se indica en la documentación oficial, requiere de las versiones de Python 3.6 a 3.9. En nuestro caso, tenemos instalada tanto la versión de Python 3.9 como la versión de Python 3.7, aunque, por ser más antigua y, más estable las versiones para Python 3.7, es la que utilizaremos y utilizaremos la versión de TensorFlow 2.1.0.

Además, otros requisitos necesarios son:

* + Pip 19.0 o versiones posteriores
  + Ubuntu 16.04 o versiones posteriores (64 bits)

En lo referente a su instalación, es bastante larga llegando a tardar unas 14 horas aproximadamente y los comandos utilizados han sido:

$ sudo apt-get update

$ sudo apt-get upgrade

# borramos versions anteriores

$ sudo pip uninstall tensorflow

$ sudo pip3 uninstall tensorflow

# instalar las dependencias

$ sudo apt-get install gfortran

$ sudo apt-get install libhdf5-dev libc-ares-dev libeigen3-dev

$ sudo apt-get install libatlas-base-dev libopenblas-dev libblas-dev

$ sudo apt-get install openmpi-bin libopenmpi-dev

$ sudo apt-get install liblapack-dev cython

$ sudo pip3 install keras\_applications==1.0.8 --no-deps

$ sudo pip3 install keras\_preprocessing==1.1.2 --no-deps

$ sudo pip3 install -U --user six wheel mock

$ sudo -H pip3 install pybind11

$ sudo -H pip3 install h5py==2.10.0

$ sudo -H pip3 install --upgrade setuptools

# descargamos el wheel

$ wget https://github.com/Qengineering/Tensorflow-Raspberry-Pi/raw/master/tensorflow-2.1.0-cp37-cp37m-linux\_armv7l.whl

# instalamos TensorFlow

$ sudo -H pip3 install tensorflow-2.1.0-cp37-cp37m-linux\_armv7l.whl

# reiniciamos

$ sudo reboot

Comandos 1 - Instalación de TensorFlow

Igualmente, otra de las librerías que se ha tenido que instalar y, que nos ayudará a nuestra parte de inferencia, ha sido la de PyCoral, la cual ha sido instalada debido al uso del Acelerador USB Coral, que nos proporciona una TPU Edge como coprocesador para nuestra Raspberry. Esto quiero decir que, nos ayuda a acelerar la inferencia del modelo de AA cuando se conecta.

Para poder utilizar el Acelerador USB, es necesario cumplir que:

* + OS Linux Debian 10 o un derivado del mismo (Ubuntu 18.04 es una alternativa)
  + Una Arquitectura ARMv7 (32 bits) o ARMv8 (64 bits)
  + Puerto USB disponible (en nuestro caso utilizamos el puerto 3.0 para mayor velocidad)
  + Versión de Python 3.6 a 3.9

En cuanto a su instalación, es bastante más rápida que la de TensorFlow y los comandos utilizados son los siguientes:

# agregamos el package

$ echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" | sudo tee /etc/apt/sources.list.d/coral-edgetpu.list

$ curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

$ sudo apt-get update

$ sudo apt-get install libedgetpu1-std

# conectamos el Acelerador USB

# para más velocidad instalamos la capacidad de operación de máxima frecuencia

$ sudo apt-get install libedgetpu1-max

# instalamos la Liberia de Python

$ sudo apt-get install python3-pycoral

# se comprueba que la instalación está correcta

# descargamos el codigo de jeemplo de GitHub

$ mkdir coral && cd coral

$ git clone https://github.com/google-coral/pycoral.git

$ cd pycoral

# descargamos el modelo, las labels y la foto

$ bash examples/install\_requirements.sh classify\_image.py

# ejecutamos el clasificador

$ python3 examples/classify\_image.py \

--model test\_data/mobilenet\_v2\_1.0\_224\_inat\_bird\_quant\_edgetpu.tflite \

--labels test\_data/inat\_bird\_labels.txt \

--input test\_data/parrot.jpg

Comandos 2 - Instalación de PyCoral

Hay que volver a destacar que la librería de PyCoral, está pensada para mejorar la inferencia final, es decir, que los tiempos de respuesta de la solución de un problema sea lo más rápido posible, y no está pensada para que la etapa propia de entrenamiento utilice este método de Edge TPU. Básicamente, está pensado para hacer uso de un modelo tflite, que es el resultante de una transformación a un modelo ya previamente entrenado.

Teniendo en cuenta estas librerías, junto con sus dependencias, además de otras como son Numpy, Sklearn, Pandas o Matplotlib, lo más sencillo para volver a replicar las instalaciones es utilizar un fichero típico de instalaciones como un requirements.txt, que englobe todas las librerías junto con sus versiones.

# para crear un fichero requirements

$ sudo python3.7 -m pip freeze > requirements.txt

# para instalar los requirements

$ sudo python3.7 -pip install requirements.txt

Comandos 3 - Ejemplo de creación e instalación de fichero de requirements

* Pi Camera

Además de instalación de librerías y del acelerador USB, se ha tenido que instalar una cámara, que es la que nos ayudará a obtener las imágenes en la aplicación final y que serán las que se mandaran a la parte de inferencia del modelo, para avisar si estamos en el camino correcto, es decir, vamos a un paso de cebra de forma correcta, o vamos hacia un peligro inminente como es una carretera.

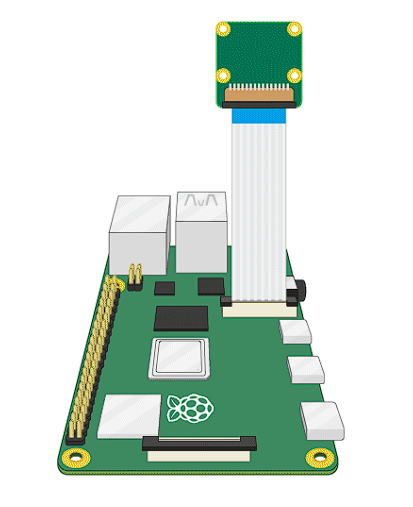
Esta cámara que instalamos se trata de una Pi Camera, que hay que conectarla tanto de forma física, como indicarle que utilizaremos una cámara en la parte de configuración.

Figura 28 - Instalación física de la Pi Camera

Para habilitar la cámara tendremos que entrar en la Configuración de Raspberry y hacer un set Enable de la Camera:

$ sudo raspi-config

Comandos 4 - Entrar en configuración de Raspberry para habilitar la Pi Camera

Seguidamente entramos en Interface Options:

Figura 29 - Configuración de la Pi Camera (I)

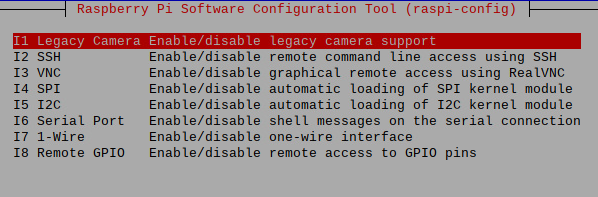
Posteriormente, le damos a Legacy Camera y lo ponemos Enable:

Figura 30 - configuración de la Pi Camera (II)

* Errores comunes de instalación

Durante algunas las fases de las instalaciones de librerías, han surgido problemas difíciles de resolver. Algunos de ellos han sido:

* + ImportError: No module named cv2:

A simple vista parece que la librería no está instalada, sin embargo, tras varios intentos desinstalando, haciendo rebuild e instalando distintas versiones, dimos con la solución. Al parecer la instalación típica que se hace de ‘*pip3 install opencv*’ no funciona cuando estamos ejecutando en un environment, para ello hay que hacer lo siguiente:

$ sudo python3.7 -m pip install opencv-python

# para ejecutar dentro de nuestro env la libreria opencv instalamos  
$ sudo apt-get install python-numpy

$ sudo apt update && sudo apt-get install python-opencv python-scipy ipython

Comandos 5 - Resolución del problema: No module named cv2

* + Matplotlib error: command ‘/usr/bin/x86\_64-linux-gnu-gcc’ failed with exit code 1’:

Este error se debe a una falta de dependencias que no se instalan de forma automática si Matplot es instalado como dependencia de la librería de Keras. La solución finalmente fue la siguiente:

$ sudo apt-get install libxml2-dev libxslt1-dev

$ sudo apt-get install gcc python3.7-dev

Comandos 6 - Resolución del problema: Matplot error

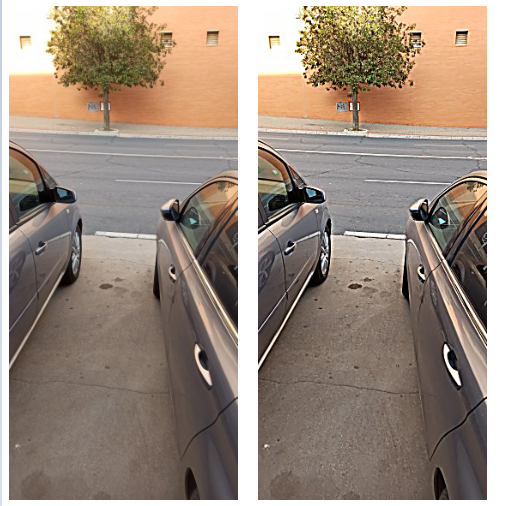
* 1. El Dataset

Para todo el proceso de entrenamiento, se ha necesitado de un dataset, el cuál hemos empezado nosotros a realizarlo desde 0 y llegando a sacar unas 1487 imágenes propias, unas 709 de pasos de cebra y unas 778 de carreteras.

Sin embargo, al tratarse de un dataset pequeño, se pensó utilizar alguna técnica que nos ayudase a incrementar el número de imágenes y, para ello, se ha utilizado lo que se conoce como Data augmentation, que proporciona diversos cambios en las imágenes y que, ayuda a incrementar la precisión del modelo mediante rotaciones, variaciones del color, recortes, y/o volteos entre otros, evitando así el sobreajuste que ocurre cuando una red no llega a generalizar.

Figura 31 - Ejemplo de Data Augmentation

Además de eso, se implementó para las imágenes un método para el enfoque de las imágenes, conocido como UnsharpMask [Par20], con el objetivo de arreglar aquellas imágenes que se viesen borrosas y que generases algún tipo de ruido en el entrenamiento dificultando el aprendizaje del modelo.

Figura 32 - Ejemplo de arreglo del desenfoque (UnsharpMask)

Pese a estas técnicas las primeras pruebas no arrojaron buenos resultados, tras ejecutar 5 dispositivos durante 5 días, con un accuracy medio de los dispositivos que en entrenamiento puede verse una mejora, pero que, para su validación, el dataset no era capaz de llegar a tener una buena precisión, es decir, mostraba un modelo que no era capaz de generalizar, por lo tanto, se propuso aumentar este dataset con más imágenes para poder realizar una mejor etapa de entrenamiento.

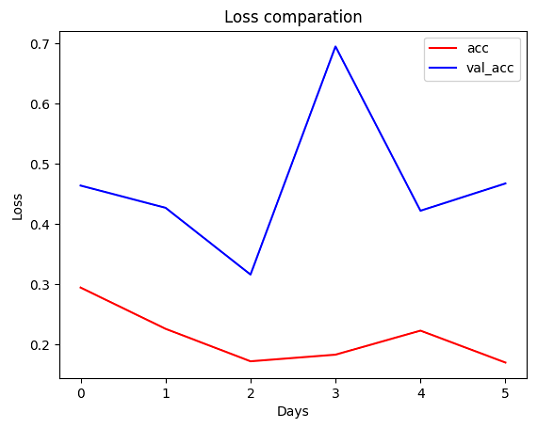
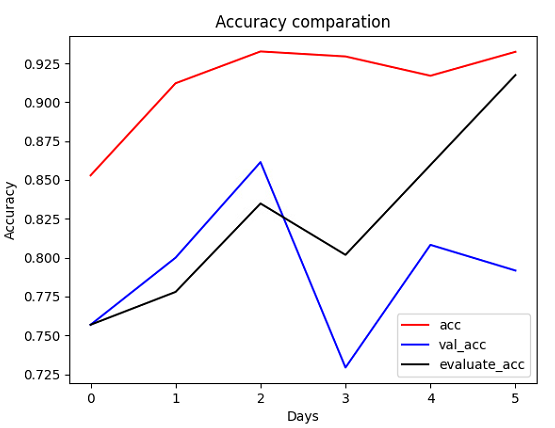


Figura 33 - Primeras pruebas. 5 dispositivos durante 5 días

Estas nuevas imágenes las obtuvimos de dos fuentes distintas, la primera de ellas [22] que corresponden a imágenes de carreteras, tanto asfaltadas como sin asfaltar, de las cuales solo se han utilizado las imágenes del archivo 18102016\_Part01, 18102016\_paved\_Part01 y 18102016\_unpaved\_Part01, consiguiendo que nuestro dataset para la clasificación de carreteras llegue a las 5.605 imágenes.

Igualmente, para la clasificación de pasos de cebra, se ha utilizado otro dataset que proviene de [23] y que, al añadirlos a nuestro dataset, compone un total de 5.802 imágenes, dándonos un dataset con un total de 11.407 imágenes, repartidas en 50.86% para imágenes de pasos de cebra y 49.14% para imágenes de carreteras, lo que supone un dataset muy balanceado en clases.

Sin embargo, para la implementación federada, se ha diferenciado entre las imágenes de Huelva, que son las imágenes que hemos realizado nosotros, y las imágenes pertenecientes a otras ciudades que no son Huelva. El objetivo principal es entrenar con las imágenes que no pertenecen a Huelva, para que el sistema federado use las imágenes de Huelva y que se logre adaptar al entorno en el que se desenvuelve normalmente.

El modelo inicial, entrenará con 2800 imágenes no pertenecientes a Huelva de esas 11.407, las cuales estarán divididas en 2000 imágenes para entrenamiento y 800 imágenes para validación para que los tiempos de entrenamiento no se disparen y podamos obtener buenos resultados, lo que resultaría en un 70% para entrenamiento y un 30% para validación de forma aproximada (concretando, 71.42% para entrenamiento y 28.58% para validación).

Para los modelos entrenados con el sistema federado, también se usará una partición de 70-30 con imágenes de Huelva.

Estas imágenes, en cada dispositivo estarán compuestas por lotes de imágenes pseudoaleatorias por seed (por el propio número del dispositivo), esto es debido a que no tenemos una gran cantidad de imágenes y en el caso de probar con 20 dispositivos tendríamos que crear lotes ya no de forma secuencial, si no de forma también aleatoria, debido a que no habría 2800 imágenes distintas para todos en el caso del entrenamiento del modelo inicial. Por esta misma razón, y ya que hay que hacer una asignación de forma pseudoaleatoria, las imágenes son asignadas en lotes pseudoaleatorios desde un inicio.

Finalmente, tanto en el modelo inicial como en el federado, se seguirá haciendo uso del Data augmentation pues supone un incremento de la precisión para los datos de validación [Wan17] y, debido a que, para nuestro problema, este es un punto crítico pues se trata de ayudar a personas con discapacidad, donde la seguridad es primordial.

* 1. Entrenamiento del modelo con TensorFlow

Cuando nos dedicamos a la parte de entrenamiento de nuestro modelo de Keras, tuvimos que empezar a definir una serie de parámetros. Estos son los conocidos como Hiperparámetros, los cuales forman un punto importante en cualquier etapa de entrenamiento y esencial para conseguir unos resultados óptimos de forma eficiente ya que controlan el proceso de aprendizaje.

Algunos de los hiperparámetros pueden ser:

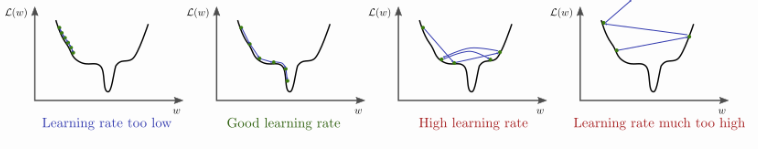
* Número de nodos de cada capa de la red neuronal, que depende del tipo de red neuronal, de la arquitectura diseñada y del objetivo del aprendizaje, siendo una variante el número final de nodos de salida. Para nosotros, este hiperparámetro podría verse como preconfigurado y posteriormente modificado, ya que hacemos uso del Transfer Learning, donde la red neuronal viene previamente diseñada, aunque posteriormente le hacemos unos cambios para adaptarla a nuestro problema, es decir, añadiéndole capas para alcanza nuestros objetivos.
* La tasa de aprendizaje o Learning rate, es un valor comprendido entre [0,1] y, es el encargado de la cantidad en la que los pesos son actualizados mediante el back propagation (o propagación hacia atrás de errores). Este hiperparámetro, consigue los mejores resultados en un Learning rate por defecto del 0.001, sin embargo, dependerá del problema, por lo que nosotros hemos optado por hacer uso de un Learning rate adaptativo, concretamente utilizaremos el optimizador Adam, que es un método de descenso por gradiente estocástico que comienza en este valor por defecto.

Figura 34 - Diferencia de Learning rates

* La división del dataset en datos de entrenamiento y pruebas, donde normalmente se usan 70-80% para entrenamiento y 30-20% para test. En nuestro caso usaremos la división del 70-30% para entrenamiento y test respectivamente, siendo un 80-20% utilizado normalmente cuando hay un conjunto de datos de entrenamiento más amplio.
* El batch size o tamaño del lote, es el número de ejemplos de entrenamiento utilizados en una iteración antes de actualizar los pesos de la red, esto quiere decir que, si utilizamos un lote muy pequeño, el algoritmo tardará mucho en entrenar y le costará generalizar, y un tamaño del lote muy grande tiene posibilidades de dar problemas de memoria en el dispositivo, además de que el tiempo de entrenamiento se dispararía, por lo tanto, hay que encontrar un punto intermedio. Lo normal es utilizar de 32 a 64 como batch size, sin embargo, cuando estamos en Edge Computing, podemos tener problemas de memoria, por lo tanto, nosotros utilizaremos un batch size de 8, el cual nos ha arrojado mejores resultados que con 16, llegando a una mejor generalización del modelo.

[Img comparativa nuetsra o internet del batch size]

* Número de épocas de entrenamiento, sería nuestra convergencia, es decir, durante cuánto tiempo tengo que seguir entrenando. Normalmente se suele poner el número más alto posible y se establece una condición de parada en función a la pérdida o loss, que suele ser el Error cuadrático medio (ECM o MSE) para evitar el sobreajuste. Para nuestro problema, este hiperparámetro no es clave, ya que lo que buscamos es que los dispositivos aprendan de forma rápida y que, envíen al servidor este modelo entrenado para su agregación federada, por lo que el valor lo pondremos a 1, donde nuestra convergencia serán el número de “días” que el federado tendrá que estar trabajando.
  1. Generación del modelo de Keras

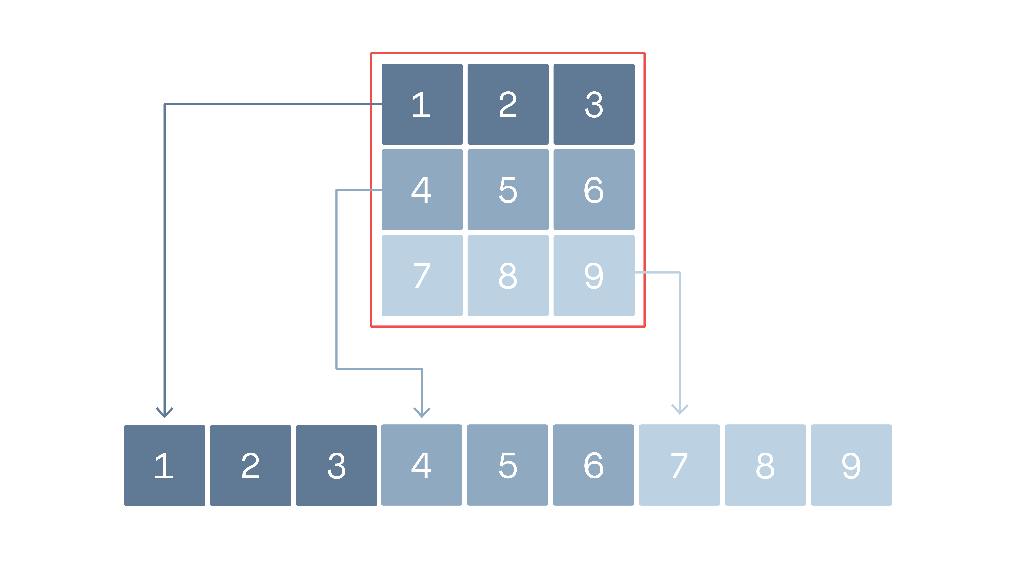
Para la creación de nuestro modelo, se ha tenido en cuenta una dimensión de las imágenes de 256 píxeles, necesitando las imágenes de entrada de un redimensionado previo. Po lo tanto, la capa de entrada está compuesta por una terna (256, 256, 3) la cual nos indica la altura y ancho de nuestra imagen, así como el número de matrices (channels), en nuestro caso al ser imágenes en color, tenemos 3 matrices que corresponden al RGB.

En cuanto a las ultimas capas, está tendrá añadido una capa 2D Average Pooling con el objetivo de no aumentar drásticamente el tamaño de parámetros de entrenamiento, que es lo que ocurriría si hiciésemos una capa Flatten (que se encarga de aplanar las dimensiones pero que multiplica los parámetros).

La capa 2D Average Pooling, trabaja como si se tratase de una reducción de tamaño a una imagen, donde se hace una agrupación promedio y de un bloque de tensores de entrada, genera un bloque más pequeño, por lo que, en sus siguientes conexiones, necesitará de menos parámetros y menos cantidad de cómputo.

Figura 35 - Ejemplo de 2D Average Pooling

Por lo tanto, y debido a que se busca no generar un modelo demasiado grande que necesite de mucho computo para entrenar, sino todo lo contrario, es decir, generar un modelo liviano que con poco computo pueda mejorar de forma rápida, se utilizará la capa *2D Average Pooling* frente a la *Flatten*.

Figura 36 - Ejemplo de Flatten

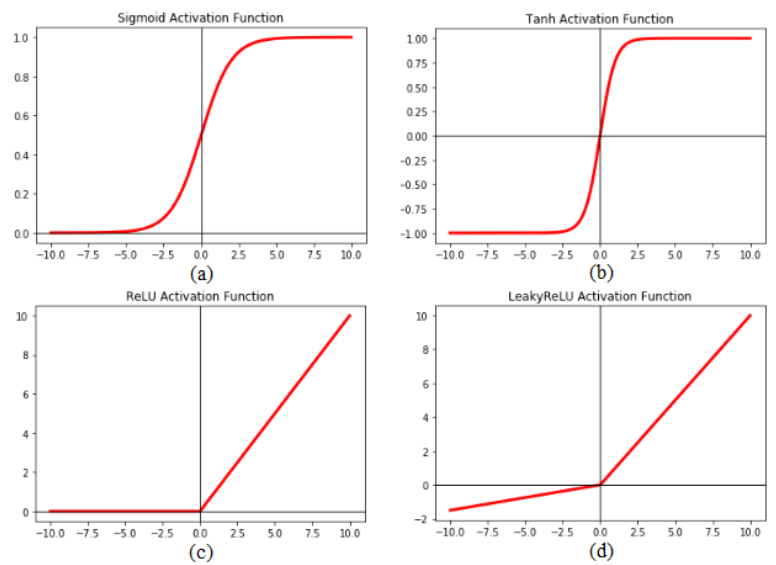
Finalmente, dos capas Dense, la primera de ellas, compuesta por 512 neuronas con una función de activación ReLU (Rectified Lineal Unit), ya que ayuda a que el modelo aprenda más rápido, frente a la tanH (Tangente Hiperbólica) y la sigmoide, evitando las zonas de saturación, esto es, que para tanH en los valores [-1, 1] y que para la sigmoide en los valores [0, 1], los valores de las neuronas no cambien significativamente, lo que hace que el gradiente no cambie y que por lo tanto los pesos de la red no generen cambios que se lleguen a apreciar, provocando mayor estancamiento en el proceso de entrenamiento, frente a la función ReLU, que al tener un comportamiento lineal para los positivos, no posee estas regiones de saturación.

Figura 37 - Comparativa de funciones de activación

Y para la segunda y última, la cuál es nuestra capa de salida, estará formada por dos neuronas con una función de activación SoftMax, la cual es una generalización de la regresión lógica y que soporta los sistemas de clasificación multiclases, por lo que es una de las funciones principales para los clasificadores. Nuestra principal ventaja al utilizar esta función, es que el rango de salida será de 0 a 1, y la suma de probabilidades será igual a 1, por lo que para nuestro problema biclase, es sumamente sencillo de calcular y nos ayuda a resolverlo.

Con estos cambios realizados a la red preentrenada de MobileNetV2, nos quedamos con:

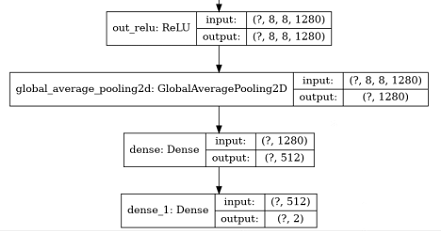
* + Número de parámetros totales: 2.914.882
  + Número de parámetros entrenables: 656.898
  + Número de parámetros no entrenables: 2.257.984

Figura 38 - Penúltima capa unida con las últimas 3 capas propias agregadas

Por lo tanto, con pocos cambios y sin generar muchísimos parámetros que entrenar, nos quedamos con una red que será capaz de aprender en menos tiempo y que está optimizada para trabajar en dispositivos de bajos recursos tanto para entrenamiento, como para inferencia.

* 1. El modelo propuesto

Al entrenar un modelo de Keras de forma convencional con un número de épocas y con un dataset de imágenes compuesta por fotografías de diversos lugares no pertenecientes a Huelva y que ha sido entrenado durante 10 épocas.

Obtenemos unos resultados que a simple vista son buenos, donde parece que el modelo es capaz de generalizar lo suficiente como para tener una buena precisión con sus datos de validación, llegando a dar unos resultados del **91.59%** de accuracy, frente al **96.21%** de accuracy en sus datos de entrenamiento, siendo ambos unos resultados bastante altos.

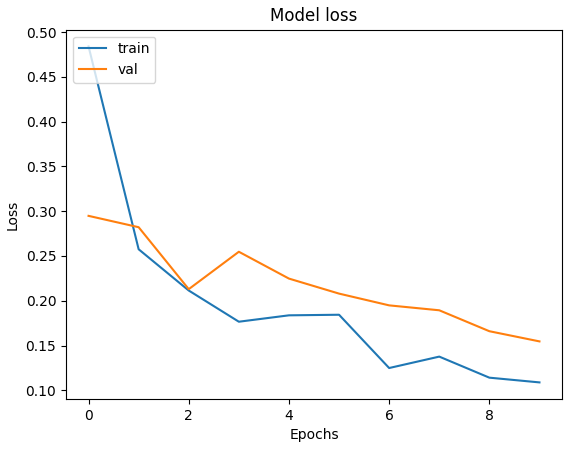
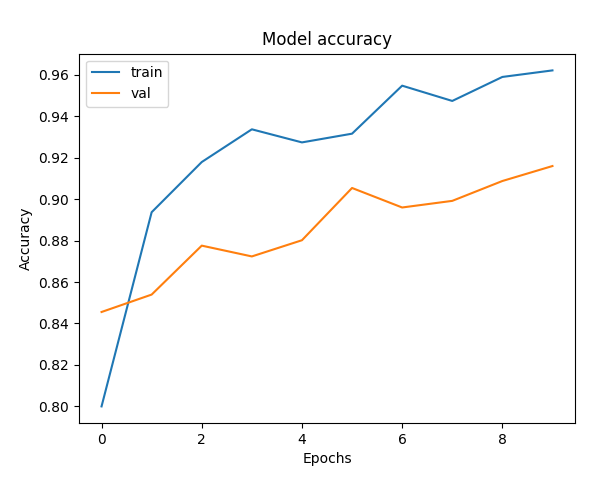


Figura 39 - Resultados del modelo inicial

Sin embargo, al pasar un dataset de validación con imágenes de una ciudad en concreto, en nuestro caso Huelva, el modelo no ha sido capaz de llegar a la precisión que tenía anteriormente, consiguiendo apenas un **77.53%** de accuracy para los datos de validación.

Esto puede ser debido a:

* Una mala generalización del modelo, necesitando este de más entrenamiento con un mayor número de imágenes, donde se cuente con un mayor número de ciudades, lo que supondría más tiempo y la necesidad de un servidor más potente.
* El tamaño de las fotografías es diferente, por lo que estas al redimensionarse para la entrada de la red neuronal pierde información.

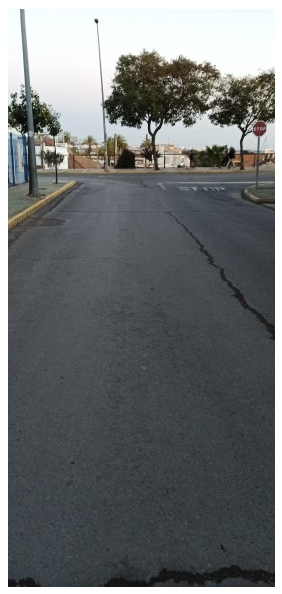


Figura 40 - Ejemplo de imágenes obtenidas en Huelva



Figura 41 -Ejemplo de imágenes no pertenecientes a Huelva

Como solución a este problema se propone un sistema federado, que busca resolver el problema de la necesidad de grandes cantidades de imágenes, y de todo el trabajo que conllevaría obtener tal dataset, además de una necesidad grande, tanto de cómputo, como de tiempo, para resolver problemas que podemos solucionar con la filosofía de “Divide y vencerás” donde los dispositivos, son capaces de adaptarse mejor a su entorno tras un periodo de reentrenamiento, con su computación y datasets propios distribuidos mejorando la precisión final del modelo.

Una vez, nuestro modelo inicial está listo, se procederá a la ejecución del sistema federado, donde todos los dispositivos recibirán este modelo como el inicial por defecto y, posteriormente, se comenzará su etapa de adaptación al entorno, reentrenando la red neuronal con las imágenes de Huelva.

El entorno federado ha sido probado con diverso número de dispositivos, aunque se destaca el que cuenta con 20 dispositivos en 15 etapas. Estas etapas, son el número de veces que se ha comunicado el modelo perteneciente al dispositivo al servidor y este ha agregado los modelos para volver a enviarlos a los dispositivos, además del tiempo de evaluación de los modelos nuevos enviados por cada dispositivo. Cabe destacar que el accuracy que vemos en el modelo federado es un accuracy medio de todos los dispositivos que han formado parte del sistema, ya que cada uno al contener sus propias imágenes tendrán un entrenamiento, validación y evaluación distintas.

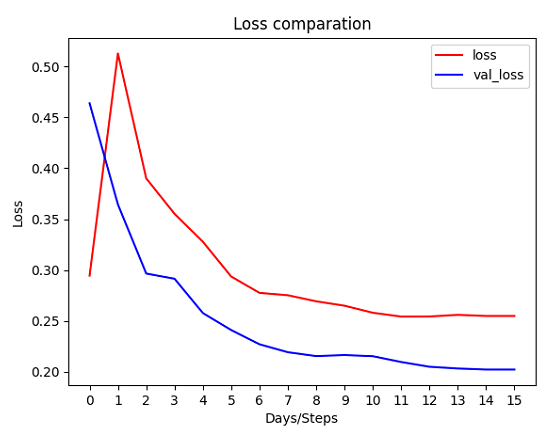
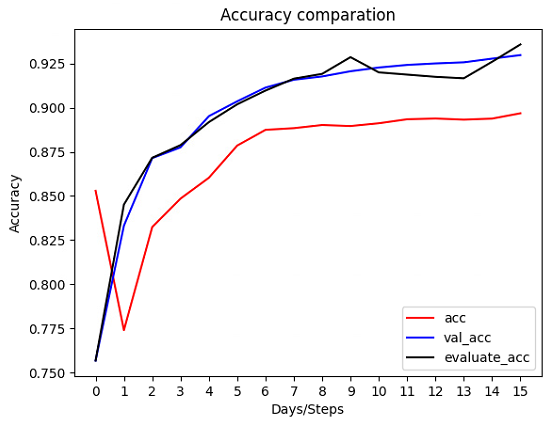


Figura 42 - Resultados del mejor modelo federado

Teniendo en cuenta que inclusive con un dataset formado con las imágenes de Huelva con un total de **1487** fotografías frente a las **2800** fotografías que usa un modelo sin federar, el sistema federado arroja mejores resultados que el entrenamiento del modelo sin federar.

El modelo federado llega al **93.57%** de accuracy para su evaluación del modelo final pasado en el servidor en su última etapa, frente al **91.59%** de validación del modelo inicial sin federar y, cambia el modelo el **40.19%** de las veces que el servidor les pasa un nuevo modelo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tipo | Accuracy (%) | Validation Accuracy (%) | Evaluate Accuracy (%) | Tiempo total de ejecución por dispositivo (h) | Número de imágenes |
| No federado | 96.21 | 91.59 | - | 1.25 (10 épocas) | 2800 |
| Federado | 89.67 | 92.97 | 93.57 | 2.37 (15 etapas) | 1487 |

Tabla 1 - Resumen comparativo del sistema federado y no federado

El tiempo de computación total en un dispositivo ronda las 2.5 horas totales, considerando las 15 etapas propuestas, pudiéndose completar la adaptación al entorno en una sola noche, aunque esto podría afectar a la precisión pues no se estarían obteniendo más imágenes del entorno para que este entrenamiento sea más preciso.

Finalmente, podemos decir que los resultados nos ofrecen una visión clara de que utilizar un sistema federado para adaptarse al entorno, resuelve problemas como la falta de datos que podría conllevar a una poca generalización del modelo, además de la eliminar la necesidad de una máquina potente para la generación de un modelo global que contenga todas las imágenes posibles, así como sus tiempos de cómputo, gracias a la utilización de una arquitectura que escala horizontalmente y no, verticalmente.

* 1. Los dispositivos federados

El modelo federado está compuesto por 2 o más dispositivos, los cuales simulan el paso de modelos y de precisiones a un servidor, que hace el trabajo de agregación de modelos.

Cada dispositivo está compuesto por un modelo en formato h5, debido a que el entrenamiento no puede realizarse con ficheros de   
TensorFlow Lite por ahora, un fichero de configuración y un fichero histórico de resultados (accuracy, validation accuracy, loss y validation loss) en formato json.

Igualmente, y con el objetivo de llevar un histórico de modelos, los modelos no son sobrescritos directamente, sino que se renombraran con su accuracy.

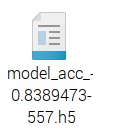
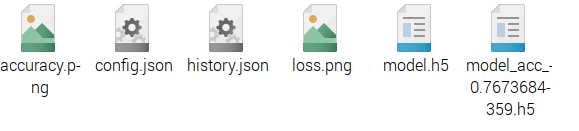


Figura 43 - Ejemplo de fichero de un dispositivo federado

Además, cuenta con un lote de imágenes para el entrenamiento propio basado en el dataset de imágenes pertenecientes a Huelva, cuya partición en entrenamiento y validación son del 70-30.

* 1. Experimentación en la arquitectura federada

Como hemos comentado anteriormente, cada experimentación realizada en la Raspberry, está formada por un conjunto de dispositivos virtuales que son ejecutados con sus imágenes propias.

Para poder guardar los resultados y poder posteriormente mostrar gráficas de resultados, se insertan en un fichero .csv los resultados de las experimentaciones, donde almacenamos datos de cada entrenamiento como:

* Dispositivo
* Accuracy
* Validation accuracy
* Evaluate accuracy
* Loss
* Validation los
* Day/step
* Tiempo de ejecución del entrenamiento en segundos (execute\_time\_seconds)
* Tiempo de evaluación del modelo pasado por el servidor en segundos (evaluate\_time\_seconds)
* Si el modelo ha cambiado por el que le pasa el servidor (is\_model\_changed)

En cuanto al número de dispositivos usados en la experimentación, se han realizado comparaciones entre 5, 10 y 20 dispositivos.

1. Experimentación con 5 dispositivos

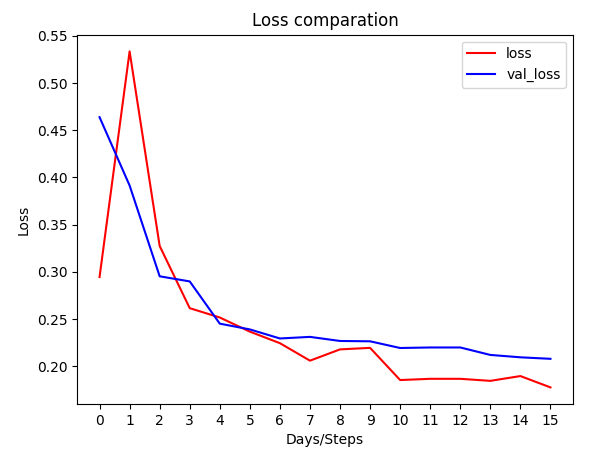
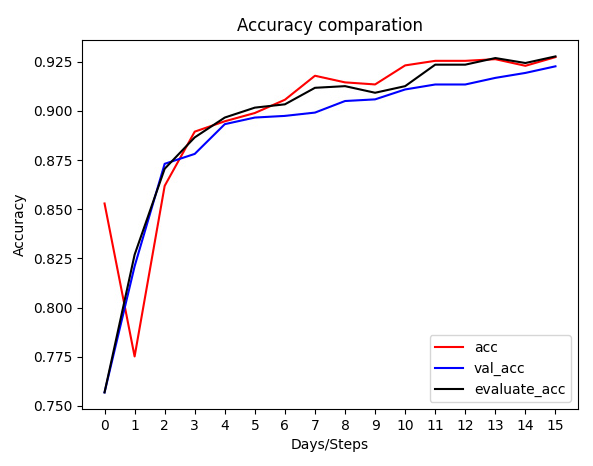


Figura 44 - Resultados con 5 dispositivos federados

Los resultados de la experimentación con 5 dispositivos ha sido buena, donde la media federada sigue trabajando bien incluso con pocos dispositivos, llegando a dar unos resultados medios del **92.73%** de accuracy, el **92.26%** de accuracy para la validación y un **92.77%** de accuracy para la evaluación, donde tenemos que hacer hincapié en este último resultado, debido a que es el correspondiente a la etapa de evaluación del modelo pasado por el servidor ya ensamblado entre todos los que se mandaron previamente. Igualmente, se logra cambiar el 67.10% de las veces por el modelo que le envía el servidor.

Observamos también una caída en la primera etapa, la cual es debida a la adaptación del modelo no federado a los modelos federados, con imágenes nuevas, en este caso, las pertenecientes a Huelva. Sin embargo, también podemos ver que el modelo federado se adapta mejor a las imágenes de forma rápida.

1. Experimentación con 10 dispositivos

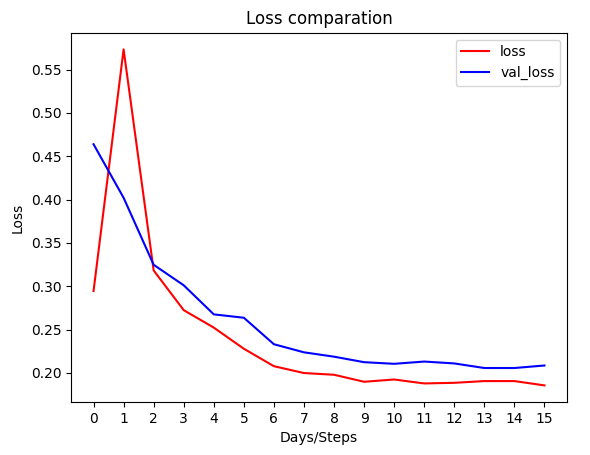
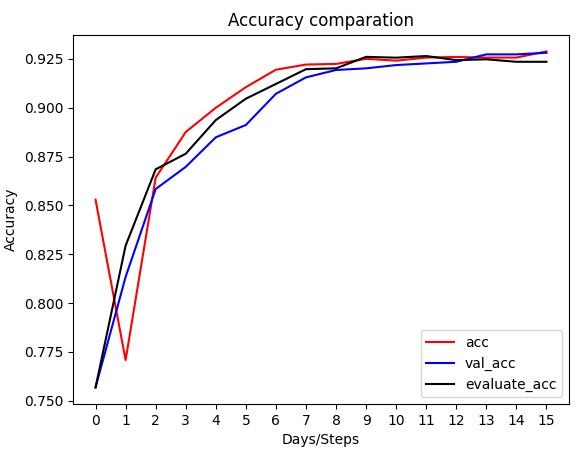


Figura 45 - Resultados con 10 dispositivos federados

Al igual que en los resultados de la federación de 5 dispositivos, observamos la caída inicial del accuracy en la primera etapa de adaptación al modelo federado. Además, este modelo consigue unos resultados un poco mayores tanto para el accuracy con un **92.88%** como para el accuracy de validación con un **92.81%**, sin embargo, vemos que empeora un poco el accuracy de evaluación del modelo agregado con un **92.35%** de media

Este pequeño empeoramiento del modelo en la etapa de evaluación, puede deberse a que se están utilizado modelos en la etapa de agregación que no son demasiado buenos, lo cual perjudica un poco al entrenamiento de grupo, donde el número de veces que se cambia el modelo es bastante menor, con un 53.64% de las veces.

1. Experimentación con 20 dispositivos

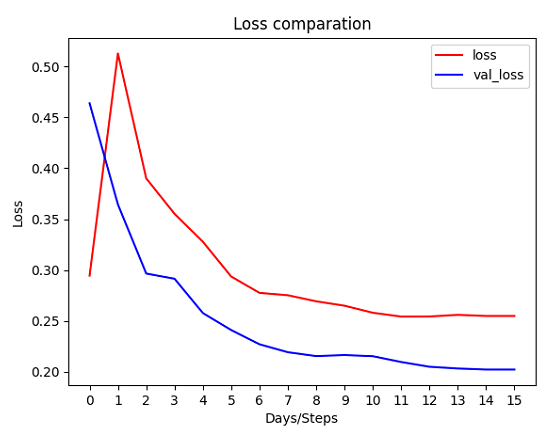
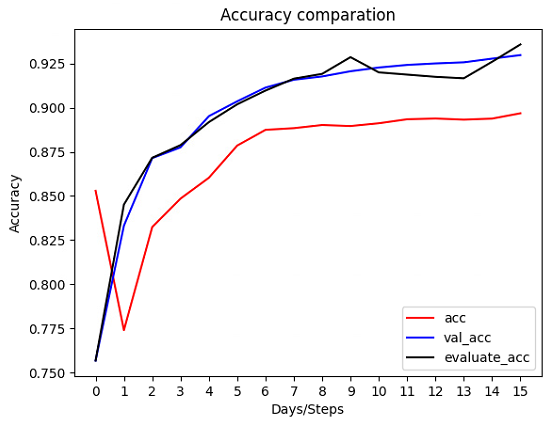


Figura 46 - Resultados con 20 dispositivos federados

Los resultados nos permiten observar que, al igual que las experimentaciones anteriores, tenemos una caída del accuracy en la primera etapa, lo cual nos habla de esa readaptación del modelo federado y, además, podemos ver claramente una gran diferencia entre el accuracy de entrenamiento, con un **89.67%,** y los accuracy obtenidos tanto en validación como en evaluación, con un **92.97%** y **93.57%** respectivamente, donde además cambia solamente el 40.19% de las veces.

Este último resultado nos deja entrever que un mayor número de dispositivos podría reducir el accuracy en la etapa de entrenamiento, lo que significaría un avance más lento de este aprendizaje. Puede deberse a un sobreajuste en el entrenamiento, debido a que se están utilizando muchas imágenes iguales en los dispositivos (debido a un dataset pequeño), aunque este sobreajuste en el entrenamiento no ha perjudicado a la generalización del modelo, sino que más bien, el tener un mayor número de dispositivos, ha influido positivamente en la obtención de una mejor precisión en el modelo, mejorando tanto al modelo no federado, como a las experimentaciones anteriores.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | No federado | 5 dispositivos | 10 dispositivos | 20 dispositivos |
| Accuracy (%) | 96.91 | 92.73 | 92.88 | 89.67 |
| Validation accuracy (%) | 91.59 | 92.26 | 92.81 | 92.97 |
| Evaluate accuracy (%) | - | 92.77 | 92.35 | 93.57 |
| Cambios (%) | - | 67.10 | 53.64 | 40.19 |
| Tiempo de entrenamiento por dispositivo (h) | 1.25 | 1.37 | 1.39 | 2.39 |

Tabla 2 - Comparativa de resultados

Los tiempos que se representan en la *Tabla 2* están formados por el tiempo que ha tardado en entrenar un dispositivo completo, donde en la ejecución del sistema federado se tiene en cuenta, tanto el tiempo de entrenamiento, como el tiempo de agregación del modelo en el Servidor, como el tiempo de evaluación posterior, donde la agregación del modelo de 20 dispositivos

Para probar que un mayor numeor d emodelos, no tiene por que perjudicar a la precisión de la etapa de agregación, se probó con 20 dispositiv

nos dice que un número pequeño de dispositivos obtendrá peores resultados que un número mayor, lo más probable debido a que se usan menos imágenes finalmente para entrenar

[Comparativa entre resultados de modelo federado de x dias y modelo con imágenes de solo Huelva generado con x epocas]

Ademas, de utilizar para nuestra experimentación una media federada o Federated Average, se ha utilizado una media ponderada, con el objetivo de comparar los datos y ver qué resultados son mejores en una experimentación real

* Se esta ejecutando en este momento. Se añadirá a la memoria

Igualmente se ha probado que un número mayor de dispositivos influye positivamente en la generación d ellos modelos, llegando a generalizar mejor obteniendo mejores resultados

* 1. Puesta en marcha de la aplicación real
* Explicar el vidoe streaming con su código
* Imágenes de como reconoce

Capítulo 5. Conclusiones

En este último capítulo se abordará una conclusión fina sobre el sistema federado implementado y su experimentación, además, se comentarán posibles trabajos futuros con el que seguir probando el sistema, así como posibles mejoras al propio sistema implementado, con intención de que cualquiera que lea el Trabajo de Fin de Master, puedo interesarle avanzar por algunas de las ramas comentadas.

* 1. Conclusiones del estudio
  2. Posibles mejoras y trabajos futuros

Durante la realización del Trabajo de fin de Master, han surgido ciertas ideas de mejoras a implementar o vías de desarrollo futuras para el sistema federado.

En cuanto a las mejoras, consideramos que utilizar datasets con un mayor número de imágenes, proporcionaría una visión más real y amplia de la capacidad y comportamiento de este sistema. Además de ello, podríamos trabajar con un entorno en el que la etapa de adaptación utilizase imágenes propias en cada dispositivo nuevas, sin ser compartidas por todos y por lo tanto repetidas (debida al poco número de imágenes), lo que daría probablemente a mejores resultados en precisión.

Por otra parte, una implementación con dispositivos reales nos daría un punto de vista distinto sobre el paso de modelos a un servidor junto con su implementación, además de poder conocer los tiempos de este paso de mensajes y como afectaría al sistema global.

En lo referido a las líneas futuras de investigaciones y desarrollos, podríamos seguir algunas de las siguientes:

* El estudio de diferentes tipos de agregaciones en el servidor, donde se tenga en cuenta el accuracy del propio modelo o técnicas más avanzadas, donde un modelo mal generalizado no perjudique al modelo agregado.
* Uso de diferentes métricas, sustituyendo al accuracy, como podrían ser el Recall, Precisión o el TNR (Tasa Negativa Verdadera).
* Implementación en otro tipo de problema, que contenga multiclases en la clasificación.
* Estudio sobre la mejore técnica tanto para agregación como para selección del mejor modelo.
* Comparativa de resultados entre un Clustering binario y el problema propuesto mediante el sistema federado junto con la comparativa de un Clustering multiclase y un sistema federado multiclase.
* Mejora del sistema para que reconozca semáforos, señales y objetos en el camino.

Referencias bibliograficas

[1] ODSC - Open Data Science, (2020). What is Federated Learning? Url: <https://medium.com/@ODSC/what-is-federated-learning-99c7fc9bc4f5>

[2] McM17. Federated Learning: Collaborative Machine Learning without Centralized Training Data. Url: <https://ai.googleblog.com/2017/04/federated-learning-collaborative.html>

[3] Kon16. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. arXiv:1610.02527v1 [cs.LG] 8 Oct 2016.

[4] Bon17. Practical Secure Aggregation for Privacy-Preserving Machine Learning. Url: <https://eprint.iacr.org/2017/281.pdf>

[5] Ram17. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629v3 [cs.LG] 28 Feb 2017.

[6] Ric17. Federated Learning: Strategies for Improving Communication Efficiency. arXiv:1610.05492v2 [cs.LG] 30 Oct 2017

[7] Bon19. Towards Federated Learning at Scale: System Design. arXiv:1902.01046v2 [cs.LG] 22 Mar 2019

[8] What is TensorFlow? The machine learning library explained. Url: <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html>

[9] Tensorflow Basic Concepts. Url: <https://www.tensorflow.org/guide/basics?hl=es_419>

[10] Transfer Learning Guide. Url: <https://www.v7labs.com/blog/transfer-learning-guide>

[11] Figura 47 - Etapas del aprendizaje por capas. Url: <https://davidnez.wordpress.com/2017/10/30/el-deep-learning-una-nueva-carrera/>

[12] Tablas de Modelos preentrenados de Keras. Url: <https://keras.io/api/applications/>

[13] Figura 48 - Arquitectura general de MobileNetV2. Url: <https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c>

[14] San18. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381v4 [cs.CV] 21 Mar 2019

[15] Bea21. WAFFLE: Weighted Averaging for Personalized Federated Learning. arXiv:2110.06978v2 [cs.LG] 13 Dec 2021

[16] Hu21. MHAT: An efficient model-heterogenous aggregation training scheme for federated learning. Url: <https://doi.org/10.1016/j.ins.2021.01.046>

[17] Yan19. Federated Machine Learning: Concept and Applications. arXiv:1902.04885v1 [cs.AI] 13 Feb 2019

[18] Figura 49 - Ejemplo de 2D Average Pooling. <Url:> <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-average-pooling>

[19] Figura 50 - Ejemplo de Flatten. Url: <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/flatten>

[20] Figura 51 - Comparativa de funciones de activación. Url: <https://www.researchgate.net/figure/Plot-of-different-activation-functions-a-Sigmoid-activation-function-b-Tanh_fig4_339991922>

[21] Figura 52 - Ejemplo de Data Augmentation. Url: <https://medium.com/analytics-vidhya/data-augmentation-in-deep-learning-3d7a539f7a28>

[22] Dataset utilizado para la agregación de más carreteras. Url: <https://lapix.ufsc.br/pesquisas/projeto-veiculo-autonomo/datasets/?lang=en>

[23] Dataset utilizado para agregar más pasos de cebra. Url: <https://github.com/samuelyu2002/ImVisible>

[24] Wan17. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. Url: <http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf>

[25] Par20. Blur More To Deblur Better: Multi-Blur2Deblur For Efficient Video Deblurring. arXiv:2012.12507v1 [cs.CV] 23 Dec 2020.

[26] Instalación de TensorFlow 2.1.0 en Raspberry Pi 4. Url: <https://qengineering.eu/install-tensorflow-2.1.0-on-raspberry-pi-4.html>

[27] Instalador del OS de Raspberry. Url: <https://www.raspberrypi.com/software/>

[28] Distintos Sistemas Operativos para Raspberry Pi. Url: <https://www.raspberrypi.com/software/operating-systems/>

[29] Información Inteligencia Artificial. Url: <https://www.ibm.com/cloud/learn/what-is-artificial-intelligence>

[30] Figura 1 – Principales aplicaciones de la IA. Url: <https://www.iberdrola.com/documents/20125/40288/Infografia_inteligencia_artificial.pdf/9140d25f-e071-345b-52d0-afc81297d9dd?t=1627271721296>

[31] Figura xx -> RNN vs CNN. Url: <https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/>

[32] Optimizador Adam. Url: https://keras.io/api/optimizers/adam/

[33] Artificial Visión Url: <https://www.n-ix.com/computer-vision/>

[34] Iot Url: <https://matob.web.id/news/most-popular-iot-technologies/>

[35] Edge Computing Url: <https://es.wikipedia.org/wiki/Edge_computing>

[36] CNN Url: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

[37] Figura 53 - Capa convolucional. Url: <https://www.researchgate.net/figure/Outline-of-the-convolutional-layer_fig1_323792694>

[38] Figura 54 - Capa pooling. Url: <https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451>

[39] Git: Url: <https://www.atlassian.com/es/git/tutorials/what-is-git>

[40] Git Info. Url: <https://www.nobledesktop.com/learn/git/what-is-git>