

**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

Resumen

La Inteligencia Artificial (IA) y, en concreto, el Deep Learning (DL) se ha vuelto una de las tecnologías con más auge en los últimos años. Este aumento es debido a la gran aplicabilidad práctica que tiene y a que, con nuevas ideas, se proporcionan no solo nuevas soluciones a problemas, sino que también se mejora la resolución de estos para hacerlos de forma más segura y eficiente.

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

Se propone diseñar e implementar un modelo de Visión Artificial (VA) eficiente, optimizado para su implementación en dispositivos con recursos limitados, modelado por una arquitectura de Aprendizaje Federado. Se desarrollará un modelo que ayude a las personas invidentes, se incluirá un estudio experimental que documente su despliegue en el mundo real y se cuantifique su viabilidad y 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 (AI) and, specifically, Deep Learning (DL) has become one of the most booming technologies in recent years. This increase is due to its great practical applicability and the fact that, with new ideas, it not only provides new solutions to problems, but also improves their resolution 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 problem, Federated Learning arises, providing the ability to create collaborative models without the need for large and powerful servers, a very exhaustive initial training, or having the ability to store large amounts of training data.

It is proposed to design and implement an efficient computer vision model, optimized for deployment on resource-constrained devices, modeled by a Federated Learning architecture. A model will be developed to assist blind people, an experimental study will be included to document its deployment in the real world, and its feasibility and usefulness will be quantified.

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 3](#_Toc108113310)

[Abstract 4](#_Toc108113311)

[Índice General 5](#_Toc108113312)

[Lista de Figuras 6](#_Toc108113313)

[Lista de Comandos 8](#_Toc108113314)

[Lista de Pseudocódigos 8](#_Toc108113315)

[Capítulo 1. Introducción 10](#_Toc108113316)

[1.1 Objetivos 10](#_Toc108113317)

[1.2 Inteligencia Artificial 10](#_Toc108113318)

[Capítulo 2. Preliminares: 18](#_Toc108113319)

[2.1 Visión Artificial 18](#_Toc108113320)

[2.2 IoT 19](#_Toc108113321)

[2.3 Edge Computing 21](#_Toc108113322)

[2.4 Redes Convolucionales 22](#_Toc108113323)

[2.5 Transfer Learning 25](#_Toc108113324)

[2.6 Keras 27](#_Toc108113325)

[2.7 El Aprendizaje Federado 28](#_Toc108113326)

[2.8 Edge TPU. Coral USB Acelerator 30](#_Toc108113327)

[2.9 TensorFlow como herramienta para el Machine-Learning 32](#_Toc108113328)

[2.10 GitHub para desarrollo 33](#_Toc108113329)

[Capítulo 3. Modelo propuesto 36](#_Toc108113330)

[3.1 Modelo original 36](#_Toc108113333)

[3.2 La Arquitectura Federada 38](#_Toc108113334)

[Capítulo 4. Estudio experimental 48](#_Toc108113335)

[4.1 Arquitectura Raspberry Pi 48](#_Toc108113337)

[4.2 Puesta en marcha del dispositivo e instalación de librerías 49](#_Toc108113338)

[4.3 El Dataset 54](#_Toc108113339)

[4.4 Entrenamiento del modelo con TensorFlow 57](#_Toc108113340)

[4.5 Generación del modelo de Keras 58](#_Toc108113341)

[4.6 El modelo propuesto 61](#_Toc108113342)

[4.7 Experimentación en la arquitectura federada 65](#_Toc108113343)

[4.8 Puesta en marcha de la aplicación real 69](#_Toc108113344)

[Capítulo 5. Conclusiones 72](#_Toc108113345)

[5.1 Conclusiones del estudio 72](#_Toc108113347)

[5.2 Posibles mejoras y trabajos futuros 73](#_Toc108113348)

[Referencias bibliográficas 74](#_Toc108113349)

Lista de Figuras

[Figura 1 - Principales aplicaciones de la IA [31] 10](#_Toc108113427)

[Figura 2 - Diferenciación de tipos de ML [32] 11](#_Toc108113428)

[Figura 3 - Diferenciación de conceptos en capas 12](#_Toc108113429)

[Figura 4 - Esquema de capas de la red neuronal 13](#_Toc108113430)

[Figura 5 - Etapas del aprendizaje por capas [33] 14](#_Toc108113431)

[Figura 6 - RNN vs CNN [34] 15](#_Toc108113432)

[Figura 7 - Ejemplo de VA [35] 18](#_Toc108113433)

[Figura 8 – Características del internet de las cosas (IoT) 19](#_Toc108113434)

[Figura 9 - Ejemplo de Edge Computing [36] 20](#_Toc108113435)

[Figura 10 - Capa convolucional [37] 21](#_Toc108113436)

[Figura 11 - Capa pooling [38] 22](#_Toc108113437)

[Figura 12 - WorkFlow del proceso de una red neuronal [39] 23](#_Toc108113438)

[Figura 13 - Funcionamiento esquemático del Transfer Learning [40] 24](#_Toc108113439)

[Figura 14 - Arquitectura de MobileNetV2 [41] 25](#_Toc108113440)

[Figura 15 - Tabla de principales modelos preentrenados 25](#_Toc108113441)

[Figura 16 - Características de Keras 26](#_Toc108113442)

[Figura 17 - Sistema de Aprendizaje Federado de Google (Android GBoard) [42] 27](#_Toc108113443)

[Figura 18 - Coral USB Acelerator [43] 29](#_Toc108113444)

[Figura 19 - Proceso de creación de modelo compatible con TPU Edge [44] 30](#_Toc108113445)

[Figura 20 - Características de TensorFlow 32](#_Toc108113446)

[Figura 21 - Ejemplo de GitHub [45] 33](#_Toc108113447)

[Figura 22 - Ejemplo del problema 35](#_Toc108113448)

[Figura 23 - Esquema del modelo propuesto 36](#_Toc108113449)

[Figura 24 - Ejemplo de modelo federado 37](#_Toc108113450)

[Figura 25 - Ejemplo de modelo con retroalimentación 38](#_Toc108113451)

[Figura 26 - Esquema de Arquitectura Federada básica 39](#_Toc108113452)

[Figura 27 - Ejemplo de fichero de un dispositivo federado 42](#_Toc108113453)

[Figura 28 - Arquitectura Federada con WorkFlow 45](#_Toc108113454)

[Figura 29 - Arquitectura de Raspberry Pi 4 Model B 47](#_Toc108113455)

[Figura 30 - Raspberry Pi Imager [46] 48](#_Toc108113456)

[Figura 31 - Instalación física de la Pi Camera [47] 51](#_Toc108113457)

[Figura 32 - Configuración de la Pi Camera (I) 51](#_Toc108113458)

[Figura 33 - configuración de la Pi Camera (II) 52](#_Toc108113459)

[Figura 34 - Ejemplo de Data Augmentation [48] 53](#_Toc108113460)

[Figura 35 - Ejemplo de arreglo del desenfoque (UnsharpMask) 54](#_Toc108113461)

[Figura 36 - Primeras pruebas. 5 dispositivos durante 5 días 54](#_Toc108113462)

[Figura 37 - Diferencia de learning rates [49] 56](#_Toc108113463)

[Figura 38 - Ejemplo de 2D Average Pooling [50] 58](#_Toc108113464)

[Figura 39 - Ejemplo de Flatten [51] 58](#_Toc108113465)

[Figura 40 - Comparativa de funciones de activación [52] 59](#_Toc108113466)

[Figura 41 - Penúltima capa unida con las últimas 3 capas propias agregadas 60](#_Toc108113467)

[Figura 42 - Resultados del modelo inicial 61](#_Toc108113468)

[Figura 43 - Ejemplo de imágenes obtenidas en Huelva 61](#_Toc108113469)

[Figura 44 -Ejemplo de imágenes no pertenecientes a Huelva 62](#_Toc108113470)

[Figura 45 - Resultados del mejor modelo federado 63](#_Toc108113471)

[Figura 46 - Resultados con 5 dispositivos federados 64](#_Toc108113472)

[Figura 47 - Resultados con 10 dispositivos federados 65](#_Toc108113473)

[Figura 48 - Resultados con 20 dispositivos federados 66](#_Toc108113474)

[Figura 49 - Resultados con 20 dispositivos federados con media ponderada 67](#_Toc108113475)

[Figura 50 - Inferencia de una carretera en la aplicación real 69](#_Toc108113476)

[Figura 51 - Inferencia de un paso de peatones en la aplicación real 69](#_Toc108113477)

[Figura 52 - Inferencia de una carretera en la aplicación real (II) 70](#_Toc108113478)

Lista de Comandos

[Comandos 1 - Instalación de TensorFlow 50](#_Toc108093125)

[Comandos 2 - Instalación de PyCoral 51](#_Toc108093126)

[Comandos 4 - Entrar en configuración de Raspberry Pi para habilitar la Pi Camera 52](#_Toc108093127)

[Comandos 5 - Resolución del problema: No module named cv2 53](#_Toc108093128)

[Comandos 6 - Resolución del problema: Matplot error 53](#_Toc108093129)

Lista de Pseudocódigos

[Pseudocódigo 1 - Inferencia con Coral 31](#_Toc107156145)

[Pseudocódigo 2 - Máquina Virtual 41](#_Toc107156146)

[Pseudocódigo 3 - Devices 43](#_Toc107156147)

[Pseudocódigo 4 - Servidor 44](#_Toc107156148)

Capítulo 1. Introducción

En el primer capítulo, se describirán los objetivos que dan lugar al trabajo y se desarrollan los conceptos de IA y DL junto a los subtipos que lo forman.

* 1. Objetivos

El primer objetivo de este trabajo de fin de master es crear y entrenar un modelo de DL con objeto de apoyar a personas con discapacidad visual, haciéndose uso de un dispositivo con capacidad de avisar de forma sonora o vibratoria, en caso de que la persona se esté dirigiendo hacia la carretera en vez de a un paso de peatones.

Como segundo, se implementará el modelo de DL en un sistema que sigue los principios de la Filosofía Federada, para trabajar de forma distribuida y obtener mejores resultados sin necesidad de un entrenamiento exhaustivo en un único servidor.

* 1. Inteligencia Artificial

Cuando se trata de resolver problemas o incluso de proponer mejoras a estos, una de las formas más comunes de solucionarlos es hacer uso de la IA.

Cuando se habla de IA, habitualmente se refiere de creación máquinas “inteligentes” que son capaces de desarrollar algún tipo de tarea de forma autónoma. Con posibilidad de mejorar de forma iterativa a partir de los datos que pueden ir recopilando, donde se busca resolver problemas de cierta complejidad de la forma más eficiente posible.

En lo referente a la IA existen cuatro objetivos o definiciones, 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 se abordará en este documento.
* Sistemas que actúan como humanos, que intentan imitar la forma en el que realizan tareas los humanos como, por ejemplo, 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 tienen los humanos.

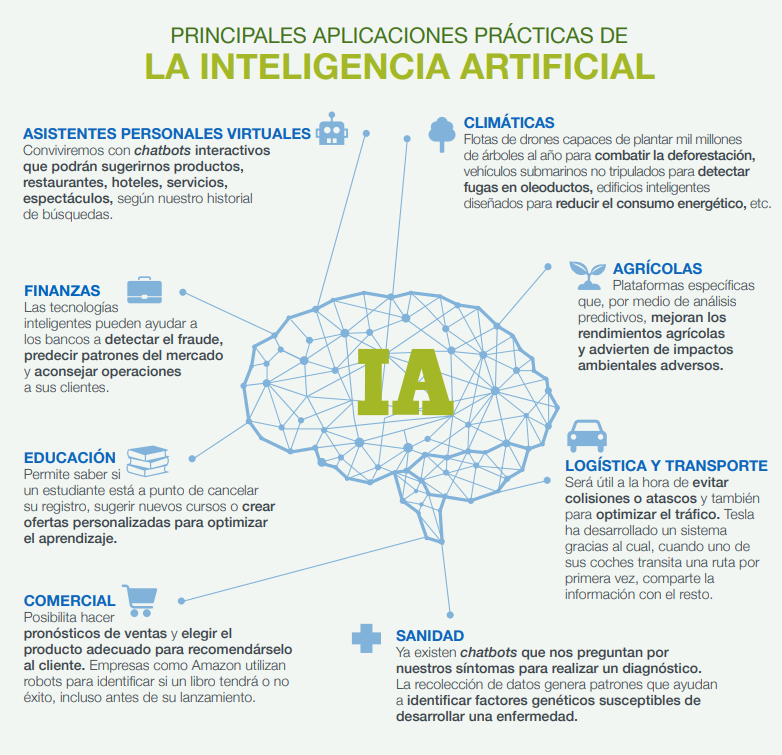
Hoy en día, la IA está presente en todos los ámbitos de nuestra vida (*ver Figura 1*) como, por ejemplo, en aplicaciones como Siri o Alexa que realizan un procesamiento del lenguaje natural (NLP), convirtiendo la voz en órdenes e incluso las acciones comerciales que se encuentran en Amazon, recomendando productos en función a nuestras compras y/o búsquedas.

Figura - Principales aplicaciones de la IA [31]

Igualmente, cuando se profundiza en las ramas de la IA, se empieza a ver diferentes tipos de paradigmas encargados de dar solución a distintos problemas y, es aquí donde se encuentra con el AA o Machine Learning (ML), 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 1956 por John McCarthy. Sin embargo, en los últimos años ha ganado importancia debido al aumento de la capacidad que los dispositivos de computación.

Se deferencian tres categorías principales (*ver Figura 2*):

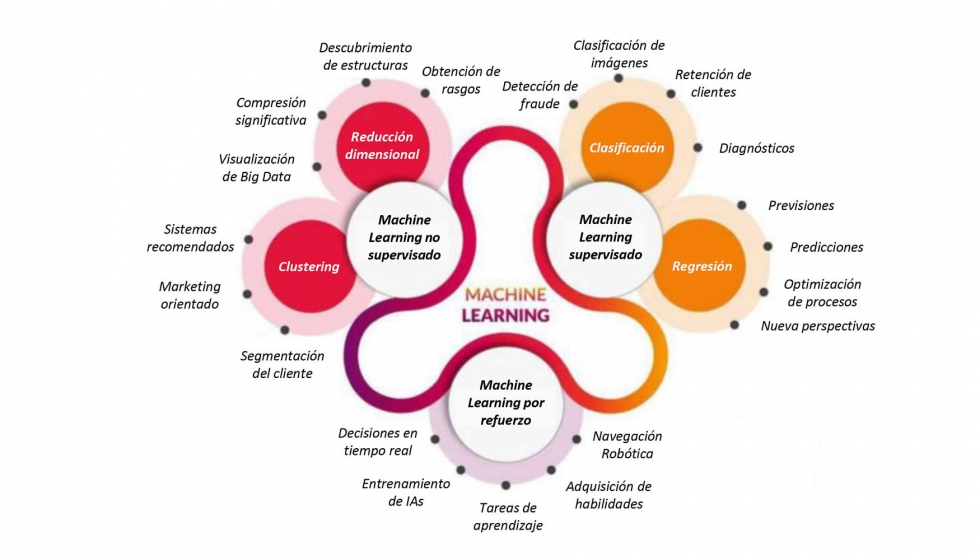
* + Aprendizaje supervisado, donde los algoritmos necesitan de un aprendizaje previo basado en etiquetas asociadas a los datos, que les permitirá tomar decisiones o hacer predicciones.
  + Aprendizaje no supervisado, donde los algoritmos no necesitan de aprendizaje previo, si no que comienzan a aprender 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 - Diferenciación de tipos de ML [32]

Sin embargo, dentro del ML, se encuentra una rama más compleja, que busca construir y entrenar redes neuronales, con el objetivo de poder ordenar, clasificar o encontrar anomalías en sus datos.

Concretamente, se habla de la neuro computación que forma parte del DL y que es una red neuronal que presenta una arquitectura profunda, donde se busca un aprendizaje desde lo más básico a lo más complejo basado en niveles de abstracción por 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.

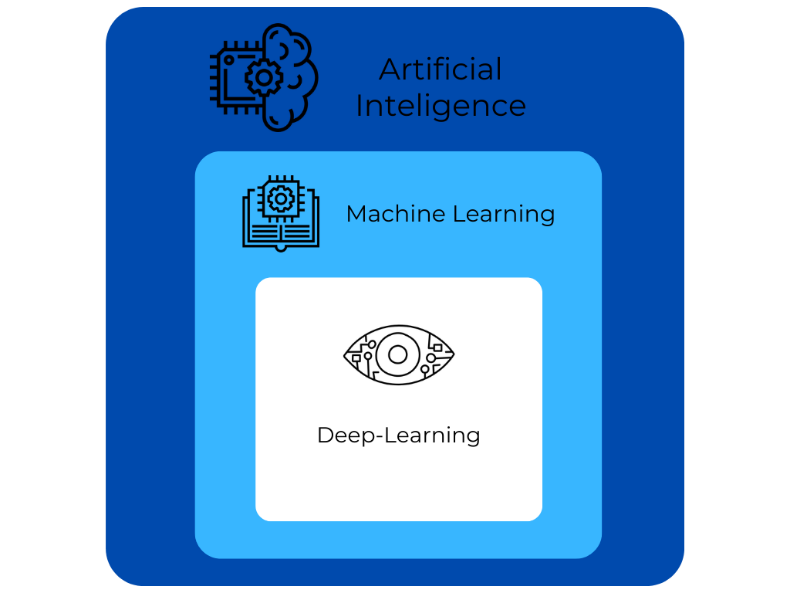
Actualmente, el DL está muy enfocado en el reconocimiento de vídeo, imagen, texto y audio, la conducción autónoma, la robótica y la sanidad entre otros, siendo un campo de estudio orientado a los resultados, por lo que está recibiendo mucha atención por parte de investigadores y académicos, donde los 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.

Figura - Diferenciación de conceptos en capas

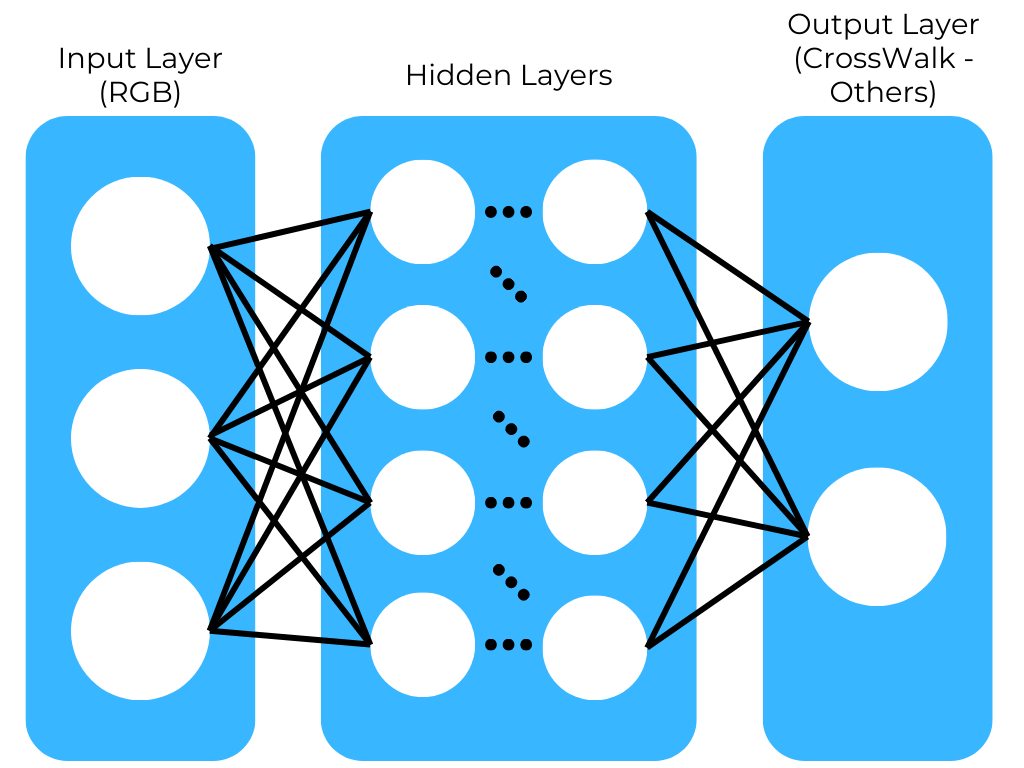
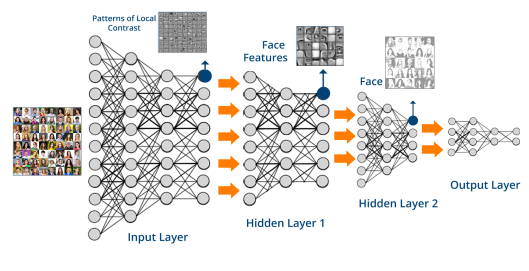
Sin embargo, no es lo mismo hablar de IA, ML y DL. (*ver Figura 3*). En el DL se parte de una gran cantidad de datos y, tras un buen número de etapas de entrenamiento, se consigue que el ordenador produzca un modelo que generaliza lo aprendido, o lo que es lo mismo, el modelo consigue aprender de forma autónoma, 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.

Figura - Esquema de capas de la red neuronal

Una red neuronal está formada por neuronas artificiales estructuradas en 3 niveles principales (*ver Figura 4*):

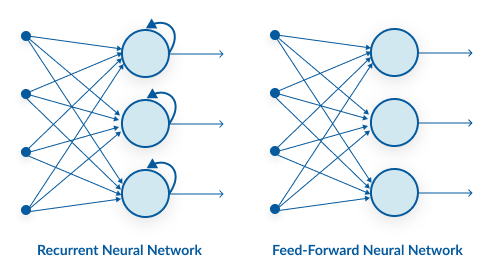
1. Input Layer: en este primer nivel se reciben los datos a inferir y, donde las neuronas se activan o no dependiendo de su función de activación.
2. Hidden Layers: este segundo nivel está formado por las capas ocultas, donde se comienza con un reconocimiento a nivel sencillo (líneas y puntos) y que posteriormente se aumenta este nivel de reconocimiento a figuras geométricas más complejas.
3. Output Layer: este tercer nivel presenta la predicción de los datos de entrada.

Figura - Etapas del aprendizaje por capas [33]

Se pueden clasificar en 2 de los tipos de redes (*ver Figura 6*):

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

Se utilizará la CNN, puesto que están orientadas a clasificación de imágenes mediante el reconocimiento y aprendizaje de patrones, siendo las RNN la tecnología usada por los asistentes personales.

Figura - RNN vs CNN [34]

En cuanto a las plataformas de DL más comunes, dos son las más destacadas: TensorFlow y PyTorch. Estas dos plataformas están en constante mejora, aunque TensorFlow tiene una codificación más sencilla para usuarios iniciados. Además, cuenta con integración con Keras, y será por ello el que se use para la creación y entrenamiento de modelos.

Capítulo 2. Preliminares:

El campo del AA 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 VA.

En este capítulo, se hablará de los dispositivos y conceptos para poder entender el modelo propuesto de VA, la estructura del Internet de las Cosas (IoT), el Edge Computing, se profundizará en las redes neuronales especialmente en las redes convolucionales, el Transfer Learning y el Aprendizaje Federado, además de sus elementos agregados.

* 1. Visión Artificial

La visión por ordenador es un campo de la IA 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 requiere de una cantidad de datos lo suficientemente grande, los cuales son analizados de forma repetitiva hasta que discierne las diferencias y, finalmente, es capaz de inferir las imágenes y obtener un resultado esperado. Por ejemplo, para entrenar a un ordenador en el reconocimiento de neumáticos, es necesario proporcionarle grandes cantidades de imágenes de neumáticos y elementos relacionados con ellos para que aprenda las diferencias y reconozca un neumático.

En este proyecto, es necesario contar con imágenes sobre pasos de peatones y carreteras, y que previamente al entrenamiento han sido etiquetadas para facilitar el aprendizaje.

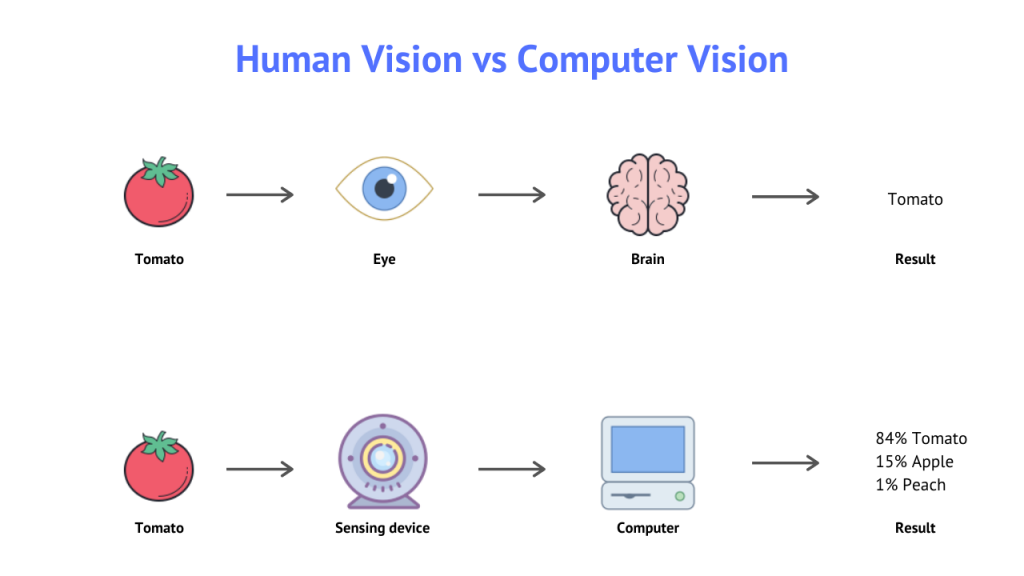
Sera necesario que contenga suficientes ejemplos para generalizar y obtener un porcentaje de aciertos al realizar la inferencia de las imágenes.

Figura - Ejemplo de VA [35]

* 1. IoT

El 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.

Dentro de un sistema de IoT puede formar parte 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 SmartWatches.

El IoT también puede hacer uso de la IA y el AA para ayudar a que los procesos de recopilación de datos sean más fáciles y dinámicos.

Algunas de las ventajas del IoT son las siguientes:

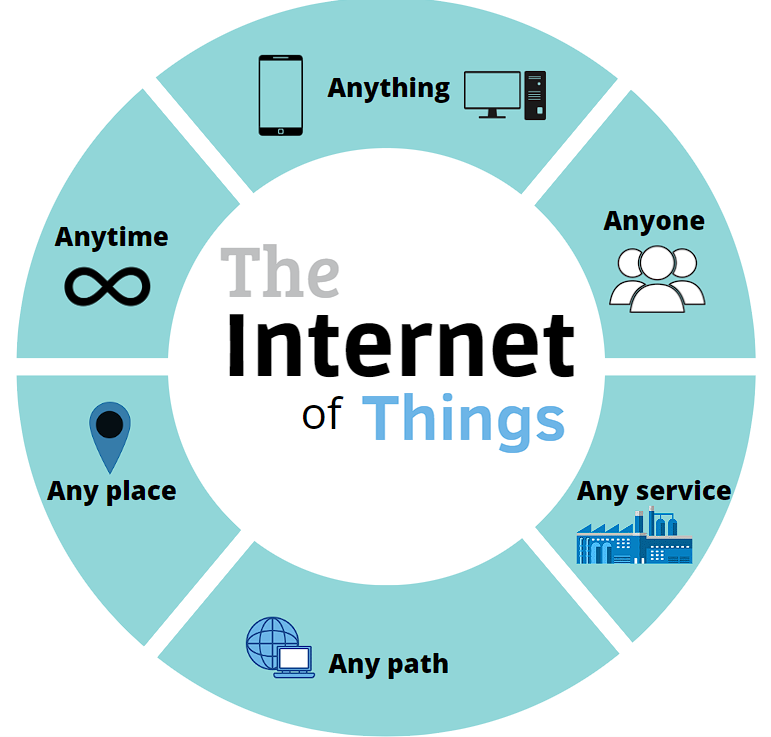
* Capacidad de acceder a la información desde cualquier lugar y en cualquier momento en cualquier dispositivo.
* Mejora de la comunicación entre los dispositivos electrónicos conectados.
* Transferencia de paquetes de datos a través de una red conectada en el que se ahorra tiempo y dinero.
* Automatización de tareas donde se ayuda a mejorar la calidad de los servicios de una empresa y se reducen la necesidad de intervención humana (*ver Figura 8*).

Figura – Características del internet de las cosas (IoT)

Algunas de las desventajas:

* A medida que aumenta el número de dispositivos conectados y se comparte más información entre ellos, se incrementa la posibilidad de que un hacker pueda robar información confidencial.
* Las empresas pueden llegar a tener que lidiar con un número masivo de dispositivos IoT, además de la recopilación y gestión de los datos de todos los dispositivos.
* Si ocurre un fallo en el sistema, es probable que todos los dispositivos conectados se corrompan.
* Debido a 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 este proyecto, el IoT formaría parte de la implementación de los dispositivos Raspberry Pi, donde se procesaría el vídeo en tiempo real para resolver el problema y cuyo modelo será compartido con el resto de dispositivos, con el objetivo de seguir mejorando, mediante el envío de los modelos a un servidor que se encarga de combinarlos según unas ciertas reglas.

* 1. Edge Computing

El Edge Computing es una arquitectura de tecnología de la información 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. Esta arquitectura de procesamiento surge como consecuencia del incremento exponencial del número de dispositivos que generan datos para ser tratados.

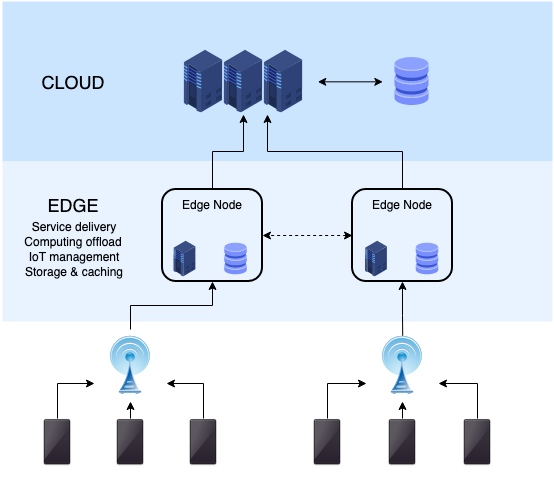
En él se traslada una parte de los recursos de almacenamiento y computación fuera del centro de datos (servidor central) y los acerca a la fuente de los datos (el dispositivo Raspberry Pi). En lugar de transmitir los datos en bruto a un centro de datos para su procesamiento y análisis, el trabajo se 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 (*ver Figura 9*).

Figura - Ejemplo de Edge Computing [36]

* 1. Redes Convolucionales

Antes de las CNN, se utilizaban métodos de extracción de características manuales y lentos para identificar objetos en las imágenes. Sin embargo, las CNN actuales proporcionan un enfoque más escalable para las tareas de clasificación de imágenes y reconocimiento de objetos, aprovechando los principios del álgebra lineal, concretamente la multiplicación de matrices, para identificar patrones en la imagen. Pueden ser muy exigentes desde el punto de vista computacional y se requiere de unidades de procesamiento gráfico para entrenar los modelos de forma más eficiente.

Presentan tres tipos de capas:

1. Capa convolucional: Es la capa más importante de una CNN y forma la arquitectura que permite que la red se concentre en características de bajo nivel en la primera capa oculta, posteriormente se ensamblan las entidades del nivel superior de la siguiente capa oculta.

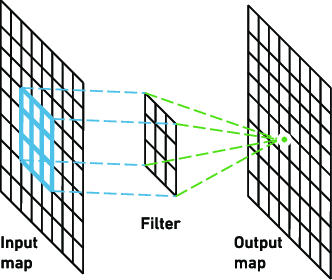
Requiere de un filtro o Kernel encargado de extraer características importantes o patrones de la imagen, siendo uno de los procesos más distintivos de las CNN, donde se recorren todas las neuronas de entrada y se obtendrá una nueva matriz que será una de las capas pertenecientes a las Capas Ocultas (*ver Figura 10*).

Figura - Capa convolucional [37]

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.

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.

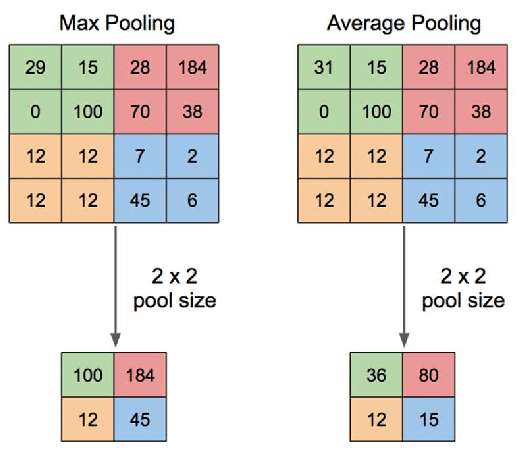
Aunque se pierde mucha información en la capa de pooling, ayuda a reducir la complejidad, mejora la eficiencia y limita el riesgo de sobreajuste (*ver Figura 11*).

Figura - Capa pooling [38]

1. Capa totalmente conectada: 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 totalmente conectadas suelen aprovechar una función de activación SoftMax para clasificar las entradas de forma adecuada, generando una salida con una probabilidad de 0 a 1.

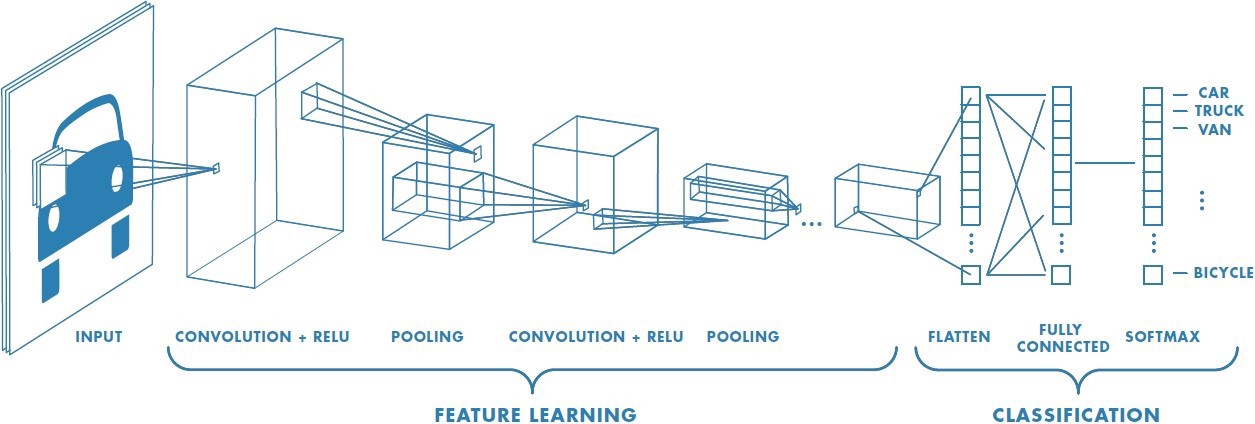
Con cada capa, la CNN aumenta su complejidad, identificando mayores partes de la imagen (*ver Figura 12*). Las primeras capas se centran en características simples, como los colores y los bordes. A medida que los datos de la imagen avanzan por las capas de la CNN, ésta empieza a reconocer elementos o formas más grandes del objeto hasta que finalmente identifica el objeto deseado.

Figura - WorkFlow del proceso de una red neuronal [39]

* 1. Transfer Learning

El Transfer Learning o Aprendizaje por Transferencia es un método de AA que se basa en reutilizar un modelo previamente entrenado como punto de partida para el entrenamiento de un nuevo modelo en una tarea diferente, permitiendo un proceso de optimización más rápido.

Al aplicar el Aprendizaje por Transferencia a una nueva tarea, se logra un rendimiento significativamente mayor que el entrenamiento de un modelo inicial con una pequeña cantidad de datos y en un menor tiempo de entrenamiento.

Esto es debido a que estos modelos preentrenados ya son capaces de reconocer bordes o formas en las imágenes, por lo que ya no es común entrenar un nuevo modelo desde el inicio para tareas relacionadas con el procesamiento de imágenes o del lenguaje natural.

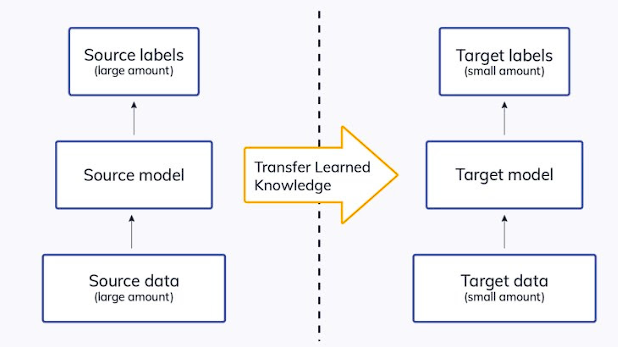
MobileNet, VGG16 e Inception, son ejemplos típicos de modelos que tienen como base el aprendizaje por transferencia (*ver Figura 13*), aunque a día de hoy existen muchísimos, los cuales se pueden encontrar en la página oficial de Keras [11].

Figura - Funcionamiento esquemático del Transfer Learning [40]

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 resolver problemas de Edge Computing, facilitando la obtención de modelos óptimos en menor tiempo. Igualmente, proporciona una buena arquitectura para mejorar la velocidad de inferencia del modelo.

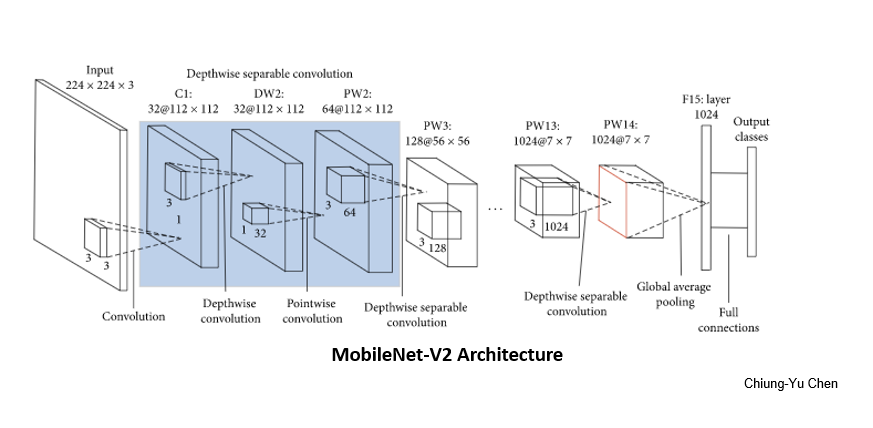
MobileNetV2 se basa en una estructura residual invertida en la que las conexiones residuales se encuentran entre las capas de cuello de botella. La capa intermedia de expansión utiliza convoluciones ligeras en profundidad para filtrar características. En conjunto, la arquitectura de MobileNetV2 contiene la capa inicial de convolución completa con 32 filtros, seguida de 19 capas residuales de cuello de botella (*ver Figura 14*) [San18].

Figura - Arquitectura de MobileNetV2 [41]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 el modelo MobileNetV2 frente a su predecesor MobileNet (*ver Figura 15*), 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 (precisión), lo que proporcionará también menos tiempo de entrenamiento en los steps y épocas.

Figura - Tabla de principales modelos preentrenados

* 1. Keras

En el DL, se utilizan redes neuronales que emplean múltiples operadores colocados en nodos para ayudar a descomponer el problema en partes más pequeñas, que se resuelven individualmente. Pero las redes neuronales pueden ser realmente difíciles de implementar, y de este problema se encarga Keras.

Keras 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.

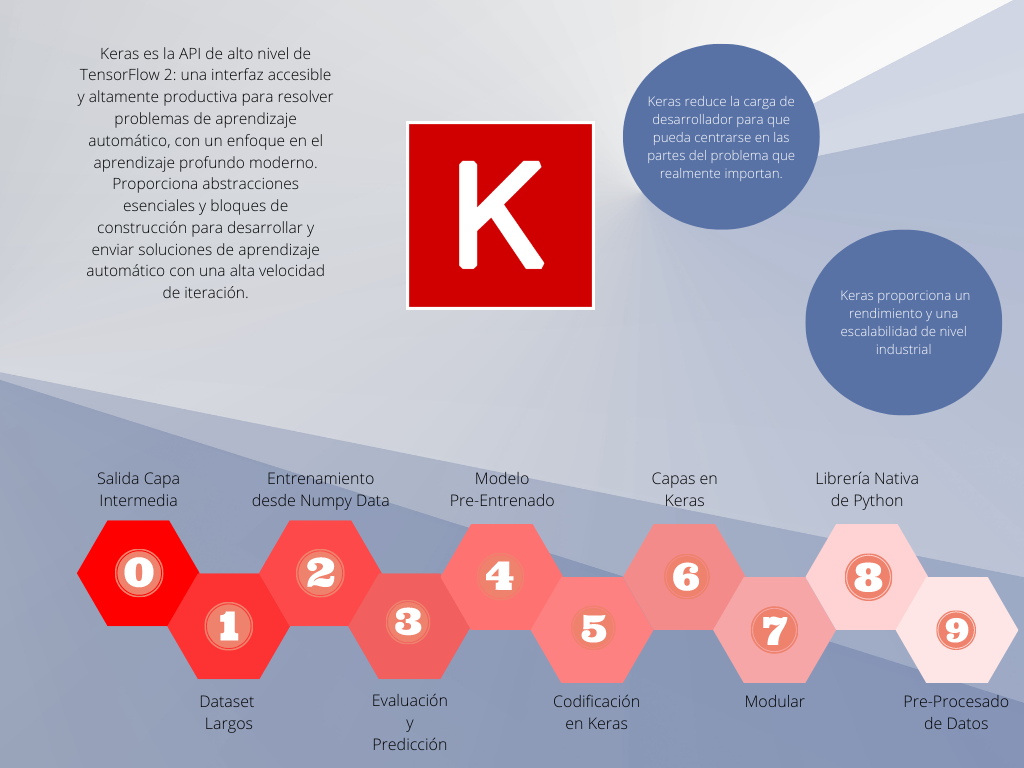
Es relativamente fácil de aprender y trabajar. Proporciona un front-end de Python con un alto nivel de abstracción mientras que tiene la opción de múltiples back-ends para fines de computación. Esto hace que Keras sea más lento que otros marcos de aprendizaje profundo, pero extremadamente amigable para los principiantes.

Figura - Características de Keras

* 1. El Aprendizaje Federado

Debido a la capacidad de cómputo que tienen los dispositivos más pequeños en la actualidad, como las Raspberry Pi o los smartphones al cual se da uso diariamente, las arquitecturas federadas han ganado importancia.

El Aprendizaje Federado “Federated Learning: Collaborative ML without Centralized Training Data” [McM17] otorga la capacidad de aprendizaje de modelos de forma colaborativa entre diferentes dispositivos, sin tener que traspasar los datos de aprendizaje a un servidor central o a la nube.

Este Aprendizaje Federado mejora la fiabilidad de los modelos, donde cada dispositivo cuenta con su modelo local que ha entrenado con sus datos almacenados, los cuales no serán comprometidos enviándolos a un servidor.

Los dispositivos envían sus respectivos modelos al servidor central que se encargará de crear un nuevo modelo ensamblado a partir de la agregación de todos los modelos recibidos. A este método de agregación se le conoce como Federated Averaging (FedAvg) o Media Federada, el cual se basa en una simple media aritmética de matrices.

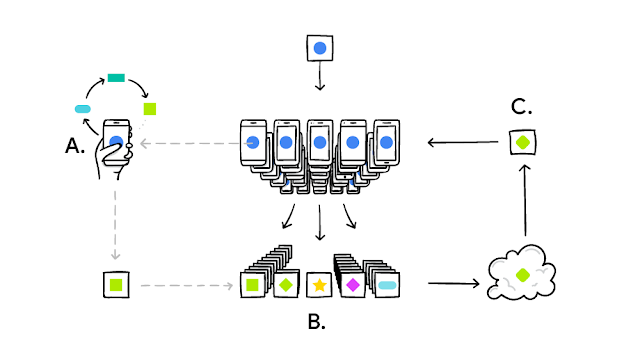
Con esta técnica, Google implementó su teclado predictivo, el GBoard el cual, como primera prueba real, fue un éxito (*ver Figura 17*).

Figura - Sistema de Aprendizaje Federado de Google (Android GBoard) [42]

Se pueden 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 FedAvg que propone Google.

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.

En cuanto a sus usos se destacan 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, el Federated Learning 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 ML que han sido generadas con TensorFlow (*ver Figura 18*), es decir, tiene como objetivo principal mejorar la inferencia que se lleva acabo de los modelos de IA.

Figura - Coral USB Acelerator [43]

Entonces Coral, ayuda a llevar a las 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 (*ver Figura 19*).

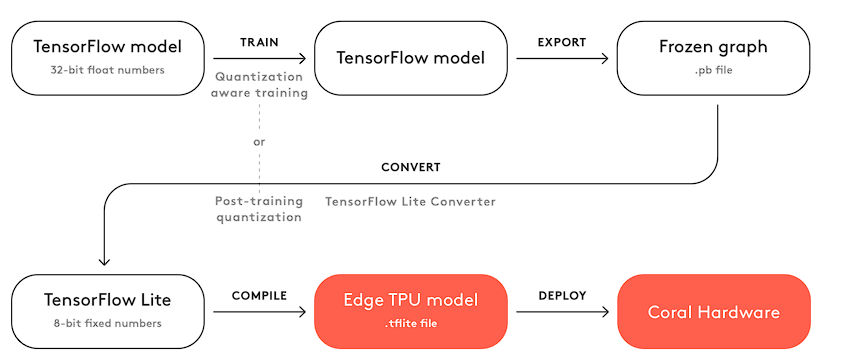


Figura - Proceso de creación de modelo compatible con TPU Edge [44]

Algunos de los usos más populares que se encuentran 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 los problemas de Edge Deep Learning, se busca procesar las imágenes de forma rápida debido a la baja capacidad computacional de estos dispositivos.

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 - Inferencia con dispositivo Coral USB Acelerator

* 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 ML, con una grandísima comunidad con multitud de recursos y tutoriales de acceso gratuito del propio Google, que permite a desarrolladores e investigadores crear e implementar aplicaciones de AA de manera más rápida y sencilla con su implementación con Keras.

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 IA que busca mejorar el rendimiento de la inferencia de los modelos en los dispositivos.

Se usa en el dispositivo Raspberry Pi el Acelerador USB TPU de Coral, para que la inferencia de las imágenes sea lo más rápida posible y dar más seguridad y eficiencia a la aplicación final que hace uso del modelo entrenado, a partir de TensorFlow Lite (tflite) que es una versión ligera de TensorFlow.

Por lo tanto, en la actualidad muchas de las aplicaciones que se utilizan como herramientas de uso cotidiano, utilizan modelos creados y entrenados con TensorFlow, como podrían ser la mejora en las fotos que se toman con los smartphones, el teclado predictivo de los móviles o, incluso el marketing personalizado, donde los anuncios que se ven dependen del uso y búsquedas que se realizan.

Figura - Características de TensorFlow

* 1. GitHub para desarrollo

Una de las principales necesidades que se tiene al inicio del desarrollo de un proyecto es saber el cómo organizar el desarrollo de código en paralelo, sin perder actualizaciones y facilitar una forma ágil para desarrollar y desplegar el código.

Como solución a esta necesidad, se hace uso de Git para tener un control de versiones y poder hacer uniones del código desarrollado en paralelo.

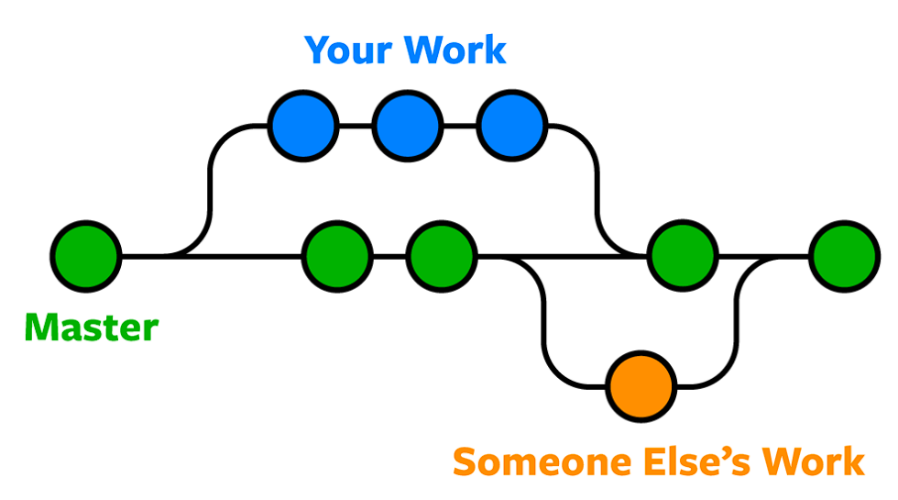
Git es el sistema de control de versiones más utilizado, el cual te permite hacer un seguimiento de los cambios que se realizan en los archivos, de modo que se tiene un registro de lo que se ha hecho y, se puede volver a versiones específicas si se necesita, por lo que facilita la colaboración además de permitir que los cambios realizados por varias personas se fusionen en una sola fuente (*ver Figura 21*).

Figura - Ejemplo de GitHub [45]

De esta forma se inicializa un repositorio para el proyecto, para poder subir las diferentes versiones y subir las diferentes versiones y desplegarlas en el dispositivo Raspberry Pi.

Los principales comandos que utilizados 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 se quiere clonar 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 algún 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 permite saltar (cambiar) entre diferentes ramas actualizando el directorio de trabajo.
  8. **GIT MERGE**: permite tomar las líneas independientes de desarrollo creadas por git branch e integrarlas en una sola rama
  9. **GIT PUSH**: 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 explica cómo se ha puesto en marcha el modelo de DL y la Arquitectura Federada propuesta, con el objetivo de generar un modelo colectivo a partir de los modelos individuales de los dispositivos.

2. 1. Modelo original

El modelo trata de resolver ciertas dificultades que tienen las personas con discapacidad visual a la hora de salir a la calle, sobre todo debido al tráfico que hay en las ciudades y los peligros que ello conlleva, donde se busca mejorar su calidad de vida haciendo uso de la IA. Además, este modelo inicial podría seguir su desarrollo de forma modular con nuevas funcionalidades, como avisar de objetos próximos u otorgando facilidades a la hora de hacer la compra. Llegando a proporcionar más información sobre su entorno.

Se busca diseñar un primer modelo con VA que identifique si un individuo que utiliza este dispositivo va en dirección a una carretera de forma inadecuada, ya sea por cruzar directamente por la calzada o desviarse del paso de peatones hacia la calzada. (*ver Figura 22*).



Figura - Ejemplo del problema

Para ello, se usará un dispositivo Raspberry Pi con la cámara incorporada que obtendrá imágenes en tiempo real para ser inferidas por el modelo e indicar el resultado de la predicción (*ver Figura 23*). Este dispositivo estará situado en la zona central del pecho.

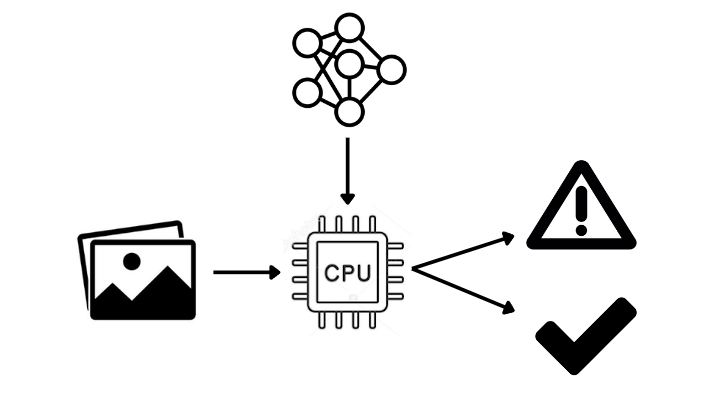


Figura - Esquema del modelo propuesto

El modelo estará formado por todas las capas del modelo MobileNetV2, con la diferencia de que se elimina la última capa de salida y se introducen tres capas unidas a la penúltima capa al modelo preentrenado.

Además, también se establece la primera capa de entrada, donde se encuentra el primer punto de flexibilidad, ya que se tiene que definir según los objetivos finales. Esto es debido a que, si las entradas de la red son pequeñas, implica que de forma genérica va a ir más rápido, pero se pierde calidad de imagen. Entonces se tiene que encontrar un balance que se ajuste al dispositivo que va a ejecutar este modelo para realizar la inferencia y que funcione de una forma eficiente.

* 1. La Arquitectura Federada

El objetivo principal de la implementación del Sistema Federado (*ver Figura 24*) es el otorgar al modelo de mayor fiabilidad y adaptación al entorno que rodea a la persona que hace uso del dispositivo, lo cual se logra gracias a una combinación de los modelos de los diferentes individuos que pertenezcan a ese mismo entorno.

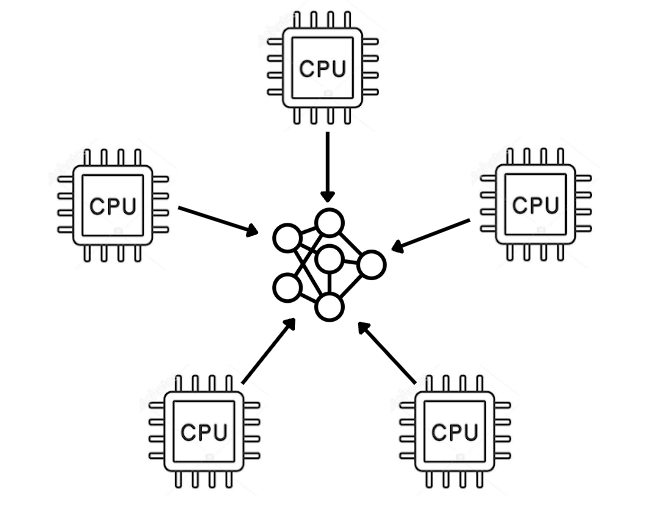


Figura - Ejemplo de modelo federado

Con esto, se otorga al dispositivo de una mayor cantidad de imágenes de entrenamiento en su periodo de adaptación al entorno y, de forma colectiva, se crea un modelo más robusto, con menos sesgos y más seguro, lo cual es primordial para solucionar el problema.

En lo referido al periodo de adaptación al entorno, este deberá de contar con una retroalimentación, es decir, se necesita indicar que las imágenes que se están infiriendo han sido correctas o no en su clasificación, para almacenarlas y tenerlas en cuenta en su posterior entrenamiento (*ver Figura 25*).

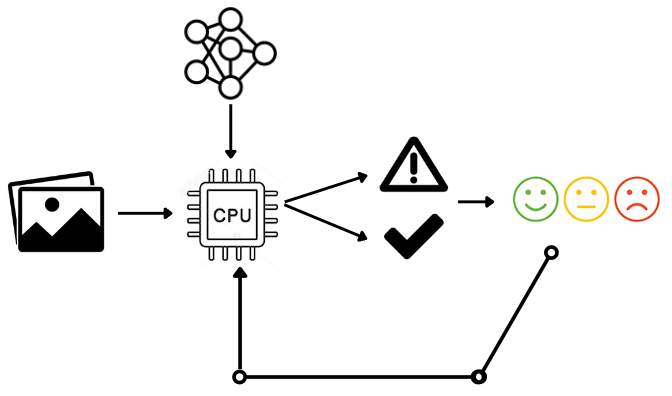
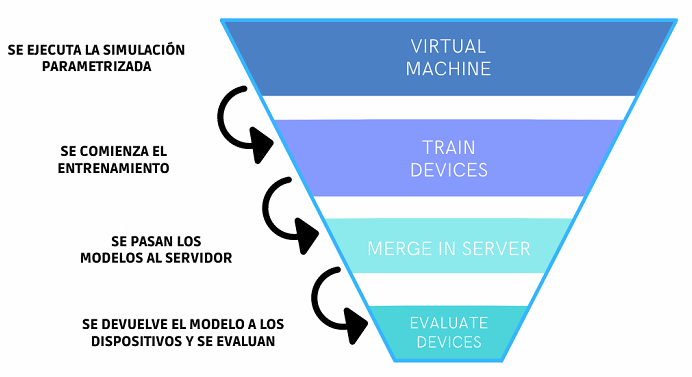


Figura - Ejemplo de modelo con retroalimentación

Sin embargo, debido a que se hace uso de una simulación en un único dispositivo, donde de forma real se necesitan numerosos dispositivos que hagan la función de los diferentes nodos del Sistema Federado, no se abordará su desarrollo en la aplicación final y se usará en su periodo de adaptación las imágenes del dataset obtenido de la ciudad de Huelva.

En cuanto a la simulación, se ha realizado en Python y consta de toda la etapa de entrenamiento federada, con diferente número de dispositivos y que debido al número reducido de Cores en la Raspberry Pi (4, de los cuales solo se dispone de 3 ya que de uno haría uso el OS), se ha realizado de forma secuencial, sin entrar en la creación de hilos ya que el tiempo de desarrollo del Sistema Federado se dispararía y, no se genera ninguna ventaja más que una reducción del tiempo total de la experimentación.

Cuando se habla de simulación secuencial, quiere decir que, lo dispositivos se ejecutan todos en una misma Raspberry Pi, y que cuando un dispositivo termina su entrenamiento, comienza el siguiente hasta acabar con los dispositivos planeados en el entrenamiento (*ver Figura 26*).

Figura - Esquema de Arquitectura Federada básica

La Arquitectura Federada, consta de cuatro 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 las etapas tiene de forma interna un flujo más definido. 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, donde se usa el FedAvg.

Posteriormente, la Máquina Virtual o Virtual Machine (VM), se pasarán los parámetros necesarios a los dispositivos, tales como el ancho y alto de la imagen que se tendrá que procesar, el dispositivo con el que se empieza, o el 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 - Máquina Virtual

Además, en la etapa de entrenamiento los dispositivos harán uso de las imágenes en la propia Raspberry Pi y de las cuales se hará una distinción basada en lotes.

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 – Devices

Cada dispositivo se compone por un modelo en formato h5, ya que el entrenamiento no se puede realizar 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, no se sobrescriben directamente, sino que se renombran con su accuracy (ver Figura 27).

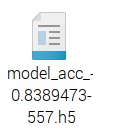
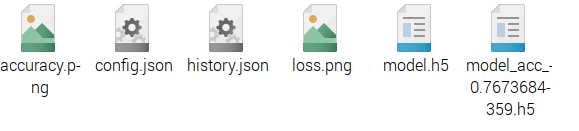


Figura - Ejemplo de fichero de un dispositivo federado

Una vez que el dispositivo completa su etapa de entrenamiento y guarda sus métricas, tales como el accuracy y el loss (Error cuadrático Medio o MSE de sus siglas en inglés) de la etapa de entrenamiento, son guardadas en un fichero CSV para posteriormente utilizar los resultados y sacar gráficas de las experimentaciones. Concretamente el fichero CSV guarda los siguientes campos:

* + 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)

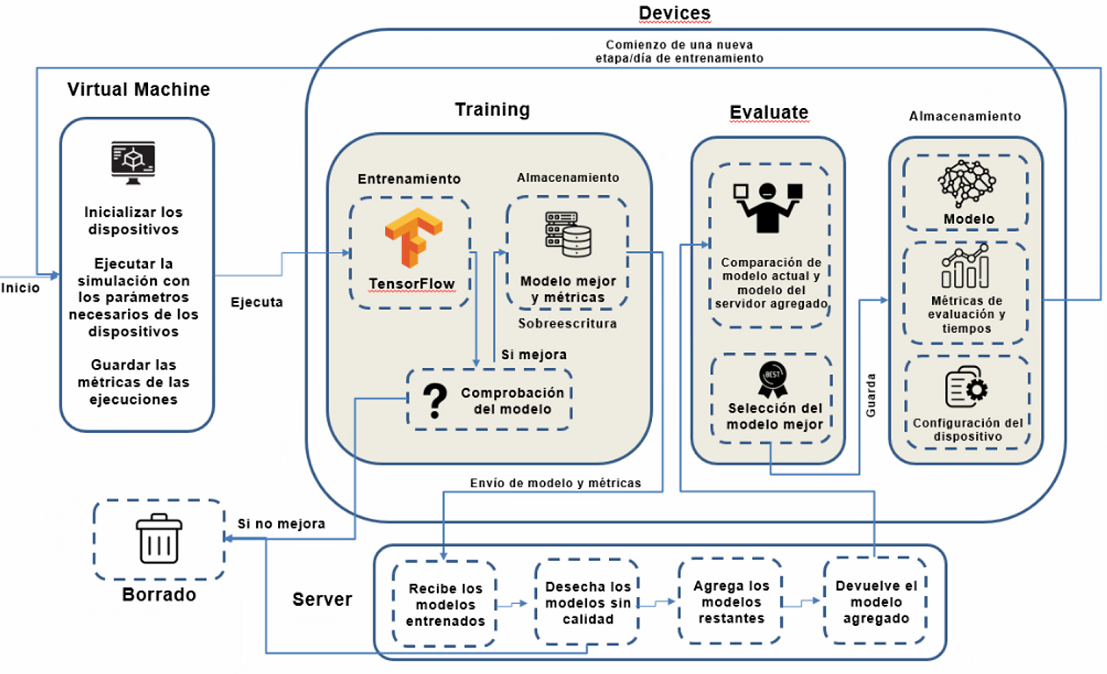
Los modelos que se envían al servidor pasan por diferentes etapas de procesamiento. La primera de ellas desecha aquellos modelos que no han tenido un entrenamiento lo suficientemente satisfactorio como para generalizar su modelo, y que su precisión en los datos de validación sea peor que el modelo inicial.

Posteriormente, dependiendo del tipo de agregación se generan los pesos de cada capa de la red neuronal que conforma el modelo de DL, creando un nuevo modelo con la misma arquitectura interna, pero modificando los pesos, solo en aquellas capas entrenables.

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

Pseudocódigo – Servidor

Además, una vez que el servidor termina de agregar los modelos, este es enviado a todos los dispositivos, por lo que se necesita empezar a ejecutar su etapa de evaluación, donde con sus imágenes de validación para 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. Finalmente se comienza una nueva etapa/día de entrenamiento (*ver Figura 28*).

Figura - 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 las imágenes asignadas, tanto para entrenamiento como para test.

Se presentan los resultados y se aplicará la viabilidad de su aplicación en el mundo real.

1. 1. Arquitectura Raspberry Pi

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 (*ver Figura 29*) que ejecuta un modelo basado en Keras y se entrena con TensorFlow. Esto se acerca a lo que se conoce como Edge Computing.

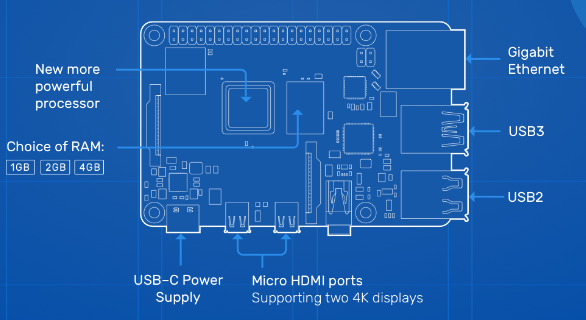
Para ello, se ha tenido que hacer una serie de instalaciones y resolución de problemas típicos que se encuentran en la arquitectura ARM.

Figura - 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, se comienza con la instalación del Sistema Operativo (OS de sus siglas en inglés) el cuál se realiza desde el instalador Raspberry Pi Imager (*ver Figura 30*) que da a elegir entre diferentes OS, y que, dependiendo de las necesidades se selecciona uno u otro [22]. Para el desarrollo del proyecto se utiliza 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 e interfaz.

Figura - Raspberry Pi Imager [46]

* Las librerías

En cuanto al dispositivo, se ha tenido que realizar una serie de instalaciones de librerías, siendo una de las más importantes la librería de TensorFlow. Esta librería es la que permite la capacidad de entrenar de forma sencilla el modelo de DL 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, la versión instalada 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 se utiliza junto a 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, se comienza con la ejecución de las siguientes sentencias en la terminal:

$ 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 - Instalación de TensorFlow

Igualmente, otra de las librerías que se ha instalado y, que facilita la parte de inferencia, ha sido la de PyCoral, la cual ha sido instalada debido al uso del Acelerador USB Coral, una TPU Edge como coprocesador para la Raspberry Pi.

Concretamente 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. Está pensado para hacer uso de un modelo tflite, que es el resultante de una transformación a un modelo ya previamente entrenado.

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 (se utiliza el puerto 3.0 para mayor velocidad)
  + Versión de Python 3.6 a 3.9

En cuanto a su instalación, se ejecutan los siguientes comandos en la terminal de la Raspberry Pi:

# 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 \

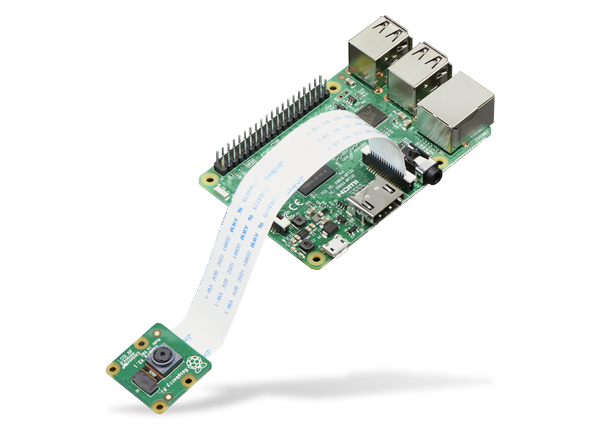
--input test\_data/parrot.jpg

Comandos - Instalación de PyCoral

* Pi Camera

Además de instalar librerías y hacer uso del acelerador USB, se ha instalado una cámara que ayudará a obtener las imágenes en la aplicación final y que serán las que se envíen para inferir con el modelo.

Esta cámara que se instala (*ver Figura 31*), se conecta de forma física, y se tiene que activar su uso en la configuración de la Raspberry Pi.

Figura - Instalación física de la Pi Camera [47]

Para habilitar la cámara se tiene que ejecutar el siguiente comando en la terminal:

$ sudo raspi-config

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

Seguidamente se accede a la opción de Interface Options:

Figura - Configuración de la Pi Camera (I)

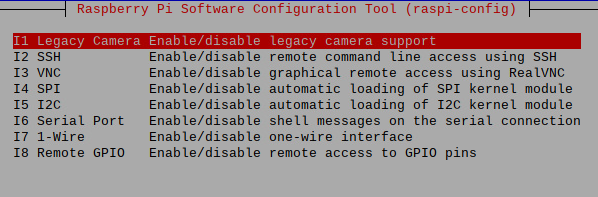
Posteriormente, se configura la opción Legacy Camera a Enable:

Figura - configuración de la Pi Camera (II)

* Errores comunes de instalación

Durante algunas las fases de la instalacion de librerías han surgido diferentes errores como, por ejemplo:

* + ImportError: No module named cv2.

A simple vista parece que la librería no está instalada, sin embargo, tras varios intentos de desinstalaciones, rebuild e instalación de distintas versiones. El problema ocurre con la instalación de ‘*pip3 install opencv*’, ya que no funciona cuando se ejecuta en un environment, para ello hay que ejecutar los siguientes comandos desde la terminal:

$ 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 - Resolución del problema: No module named cv2

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

El error se debe a una falta de dependencias que no se instalan de forma automática, en el caso de que la librería Matplot sea instalada como dependencia de la librería de Keras. Para solucionarlo se ejecuta los siguientes comandos en la terminal:

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

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

Comandos - Resolución del problema: Matplot error

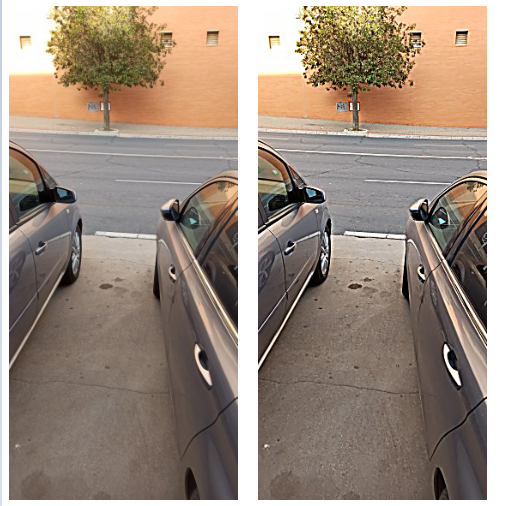
* 1. El Dataset

Durante todo el proceso de entrenamiento, se ha necesitado de un dataset, el cual se ha obtenido desde cero y llegando a sacar 1487 imágenes propias, concretamente 709 de pasos de cebra y 778 de carreteras.

Sin embargo, al tratarse de un dataset pequeño, se pensó utilizar alguna técnica que ayudase a incrementar el número de imágenes y, para ello, se ha aplicado la técnica del Data Augmentation (*ver Figura 34*), que proporciona diversos cambios en las imágenes como rotaciones, variaciones del color, recortes, y/o volteos incrementando la precisión del modelo, evitando el sobreajuste que ocurre cuando una red no llega a generalizar.

Figura - Ejemplo de Data Augmentation [48]

Además, se ha implementado un método para el enfoque de las imágenes, conocido como UnsharpMask (*ver Figura 35*) [Par20], con el objetivo de ajustar aquellas imágenes que presenten desenfoque y generen algún tipo de ruido en el entrenamiento dificultando el aprendizaje del modelo.

Figura - Ejemplo de arreglo del desenfoque (UnsharpMask)

Pese a estas técnicas las primeras pruebas no arrojaron resultados esperados (*ver Figura 36*). Tras la ejecución de 5 dispositivos durante 5 días/etapas, se puede ver una mejora en el accuracy del entrenamiento. Para su validación, el dataset no es capaz de alcanzar una buena precisión, es decir, muestra un modelo sin la capacidad de generalizar, por lo tanto, se propuso aumentar este dataset para poder realizar una mejor etapa de entrenamiento.

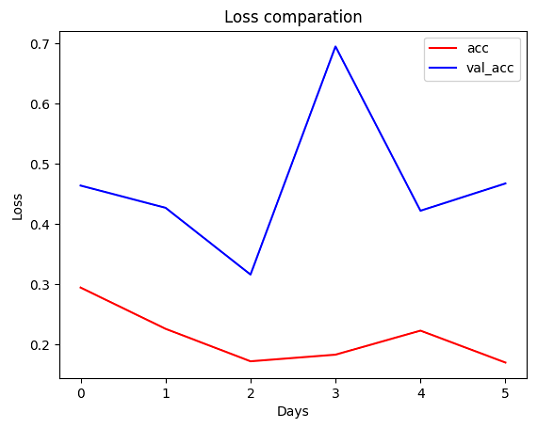
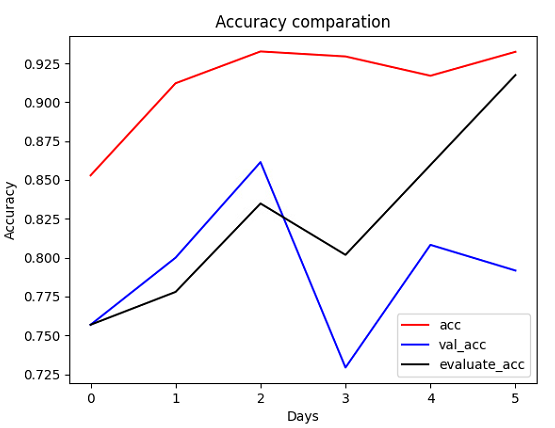


Figura - Primeras pruebas. 5 dispositivos durante 5 días

Estas nuevas imágenes se han obtenido de dos fuentes, la primera [16] se corresponde a imágenes de carreteras, tanto asfaltadas como sin asfaltar, donde 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.

La segunda se corresponde al dataset de pasos de peatones [17], al añadirse al dataset, da lugar a un total de 5.802 imágenes, proporcionando un dataset con un total de 11.407 imágenes, repartidas en 50.86% para imágenes de pasos de peatones y 49.14% para imágenes de carreteras, obteniendo 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 se han realizado por nuestra parte, y las imágenes pertenecientes a otras ciudades. El objetivo principal es entrenar el modelo inicial con las imágenes que no pertenecen a Huelva, para que el Sistema Federado utilice para su proceso de adaptación las imágenes de Huelva, logrando una adaptación al entorno en el que se desenvuelve diariamente.

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, ya que los tiempos de entrenamiento no deben incrementarse excesivamente. El resultado de esta fragmentación da lugar a un 70% para entrenamiento y un 30% para validación de forma aproximada (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.

Cada dispositivo tiene asignado un lote de imágenes pseudoaleatorias por una seed (por el propio número del dispositivo), esto es debido a que no se tiene una gran cantidad de imágenes y en el caso de probar con 20 dispositivos se tendrían que generar 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.

Finalmente, tanto en el modelo inicial como en el federado, se utiliza el Data Augmentation ya que supone un incremento de la precisión para los datos de validación [Wan17] al ser un punto crítico del proyecto.

* 1. Entrenamiento del modelo con TensorFlow

Para la creación de un modelo con Keras y su posterior entrenamiento hay que definir una serie de parámetros. Estos son conocidos como Hiperparámetros, los cuales forman un punto importante en cualquier etapa de entrenamiento y esencial para conseguir unos resultados óptimos, de forma eficiente.

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. Este hiperparámetro esta preconfigurado, al hacer uso del Transfer Learning, donde la red neuronal viene previamente diseñada, aunque posteriormente se han realizado cambios para adaptarse al problema, es decir, se añaden capas para alcanzar los 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 (*ver Figura 37*), sin embargo, dependerá del problema, por lo que se ha optado por hacer uso de un Learning rate adaptativo, concretamente se utiliza el optimizador Adam, que es un método de descenso por gradiente estocástico que comienza en este valor por defecto.

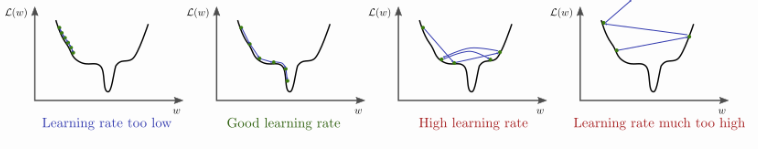


Figura - Diferencia de learning rates [49]

* 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 se utiliza un lote muy pequeño, el algoritmo tendrá un tiempo de cómputo reducido y le resultará difícil 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 aumentaría considerablemente, por lo tanto, hay que encontrar un punto intermedio. De forma genérica se utiliza un valor de 32 a 64 como batch size, sin embargo, en la arquitectura Edge Computing, se tiene problemas de memoria, por lo tanto, se utiliza un batch size de 16.
* Número de épocas de entrenamiento, es la convergencia del modelo. Normalmente se utiliza un número lo más elevado posible y se establece una condición de parada en función a la pérdida (ECM) para evitar el sobreajuste. Para el problema, este hiperparámetro no es clave, ya que lo que se busca es que los dispositivos aprendan de forma rápida y que, envíen al servidor central este modelo entrenado para su agregación federada, por lo que se usará una sola época, donde la convergencia serán el número de “días” que el federado tendrá que estar trabajando.
  1. Generación del modelo de Keras

Para la creación del modelo, se ha tenido en cuenta una dimensión de las imágenes de 256 píxeles, por lo que se necesita una redimensión previa de las imágenes de entrada. Por lo tanto, la capa de entrada está compuesta por una terna (256, 256, 3) que indica la altura y ancho de la imagen, así como el número de matrices (channels), que corresponde a la estructura de las imágenes a color (RGB).

En cuanto a las ultimas capas, tiene añadida una capa 2D Average Pooling con el objetivo de no aumentar drásticamente el tamaño de parámetros de entrenamiento, que es lo que sucede en el caso de usar una capa Flatten (que se encarga de aplanar las dimensiones pero que multiplica los parámetros).

La capa 2D Average Pooling (*ver Figura 38*), trabaja como si se tratase de una reducción de tamaño de una imagen, donde se hace una agrupación promedio y de un bloque de tensores de entrada se genera un bloque más pequeño, por lo que, en sus siguientes conexiones, necesitará de menos parámetros y cantidad de cómputo.

Figura - Ejemplo de 2D Average Pooling [50]

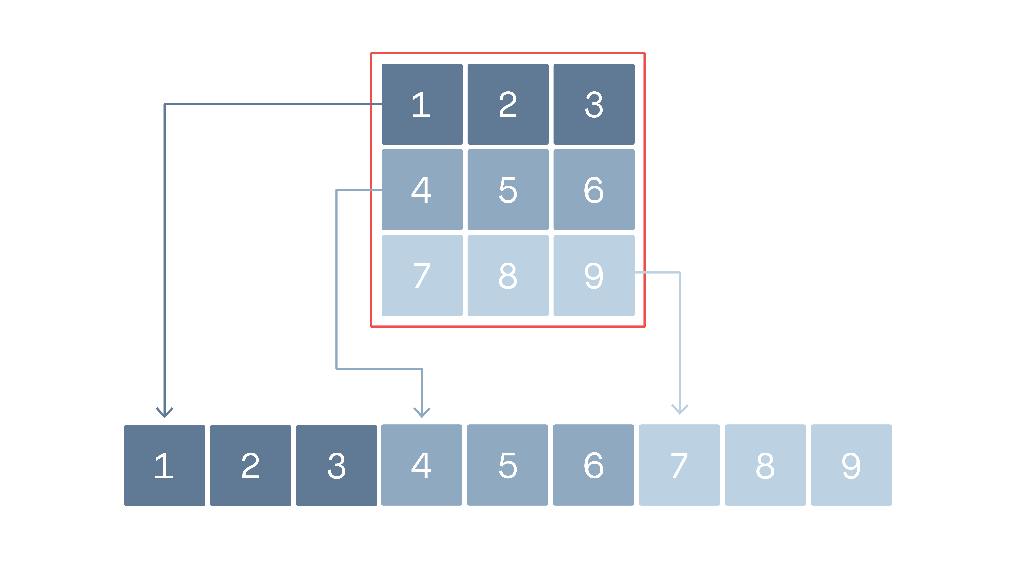
Por lo tanto, se busca generar un modelo liviano que con poco cómputo pueda mejorar de forma rápida. Por ello se utilizará la capa *2D Average Pooling* frente a la *Flatten* (*ver Figura 39*).

Figura - Ejemplo de Flatten [51]

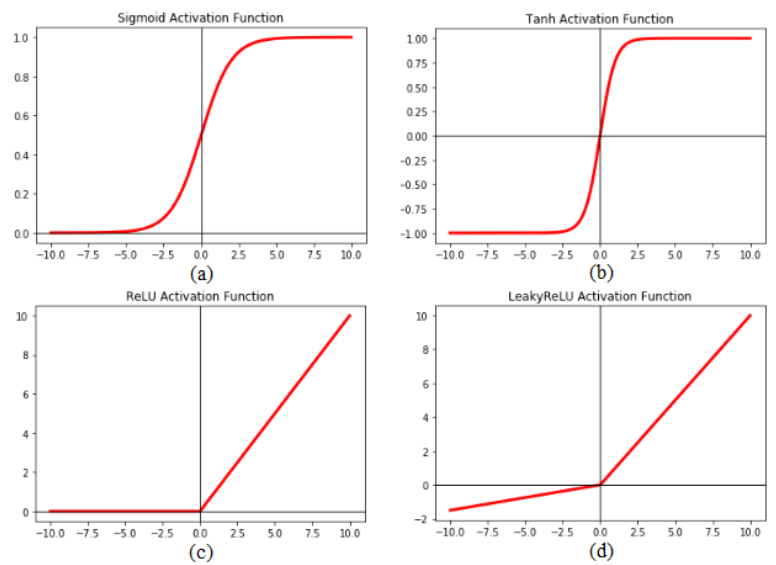
Finalmente el modelo contiene dos capas Dense, la primera de ellas, compuesta por 512 neuronas con una función de activación ReLU (Rectified Lineal Unit), ya que ayuda a que el modelo aprenda de forma más rápida, frente a la tanH (Tangente Hiperbólica) y la sigmoide, evitando las zonas de saturación, 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. Sin embargo, la función ReLU tiene un comportamiento lineal para los positivos y no posee estas regiones de saturación (*ver Figura 40*).

Figura - Comparativa de funciones de activación [52]

La segunda, la 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. La principal ventaja de 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 el problema biclase, es sumamente sencillo de calcular.

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

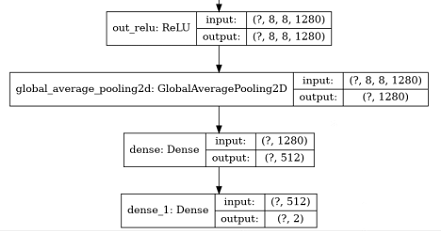
* + Número de parámetros totales: 2.914.882
  + Número de parámetros entrenables: 656.898
  + Número de parámetros no entrenables: 2.257.984

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

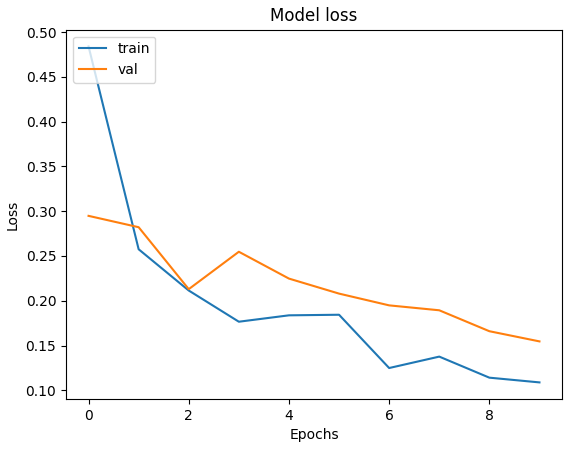
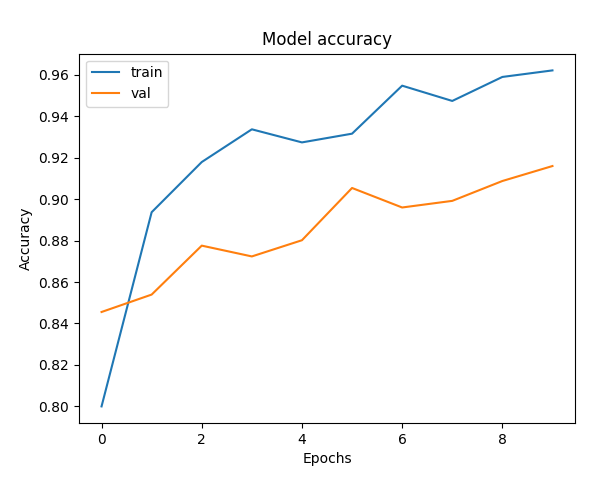
Por lo tanto, con pocos cambios y sin generar una gran cantidad de parámetros entrenables, se establece una red capaz de aprender en menor tiempo y optimizada, para trabajar en dispositivos de bajos recursos tanto para entrenamiento, como para inferencia.

* 1. El modelo propuesto

El modelo no federado de Keras ha realizado su entrenamiento durante 10 épocas utilizando imágenes compuesta por fotografías de diversos lugares no pertenecientes a Huelva.

Se obtienen unos resultados que tienen una precisión aceptable (ver Figura 42), 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.

Figura - Resultados del modelo inicial



Sin embargo, al validar con el dataset formado por imágenes de Huelva (*ver figura 43*), el modelo no es 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.

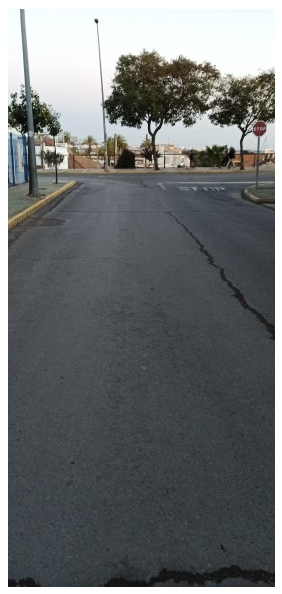


Figura - Ejemplo de imágenes obtenidas en Huelva



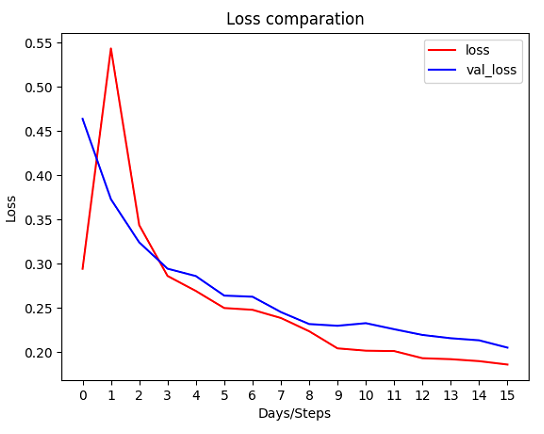
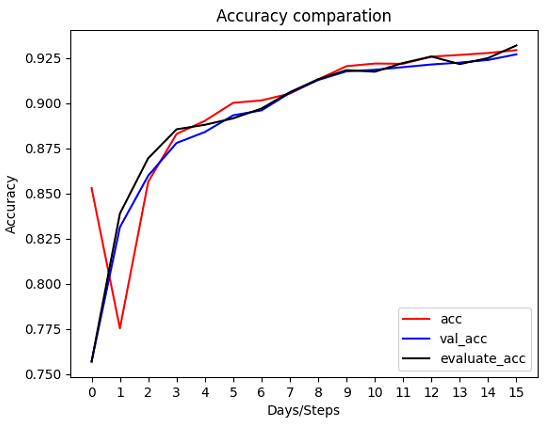
Figura -Ejemplo de imágenes no pertenecientes a Huelva

Como solución a este problema se propone un Sistema Federado, que busca resolver la necesidad de grandes cantidades de imágenes, y de todo el trabajo que conllevaría obtener tal dataset, además de una necesidad de cómputo y tiempo elevados, para resolver problemas que se puede 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, el 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 (*ver Figura 45*). Estas etapas, son el número de veces que se ha comunicado el modelo del dispositivo al servidor y este ha agregado los modelos para generar uno nuevo y redistribuirlo, además del tiempo de evaluación de los modelos nuevos enviados por cada dispositivo. Cabe destacar que el accuracy representado del modelo federado, es un accuracy medio de todos los dispositivos que han formado parte del sistema, ya que cada uno contiene sus propias imágenes para su respectivo entrenamiento, validación y evaluación.

Figura - Resultados del mejor modelo federado



Teniendo en cuenta que 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 **93.19%** 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 **57.14%** de las veces que el servidor le 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 | 92.93 | 92.70 | 93.19 | 2.37 (15 etapas) | 1487 |

Tabla - 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 de una forma rápida, aunque esto podría afectar a la precisión pues no se estarían obteniendo más imágenes del entorno para que este entrenamiento sea más preciso.

Finalmente, se puede concluir que los resultados 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. Experimentación en la arquitectura federada

En busca de probar el modelo propuesto en un entorno real, se propone comparar diferentes números de dispositivos que componen el entorno federado para ver su comportamiento y resultados.

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

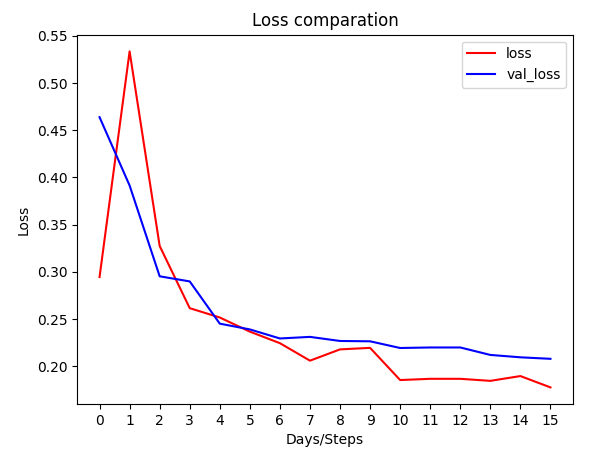
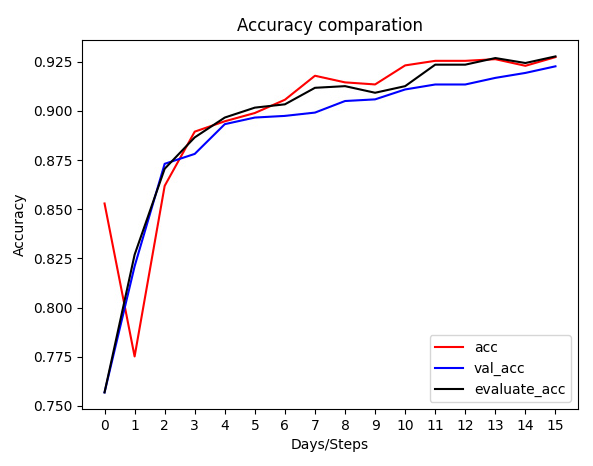


Figura - Resultados con 5 dispositivos federados

Los resultados de la experimentación con 5 dispositivos tiene un resultado aceptable, 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 se tiene 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.

Se observa también una caída en la primera etapa, la cual es debida a la adaptación del modelo no federado al nuevo sistema, haciendo uso de imágenes nuevas, en este caso, las pertenecientes a Huelva. Sin embargo, se observa que el modelo federado se adapta rápidamente.

1. Experimentación con 10 dispositivos

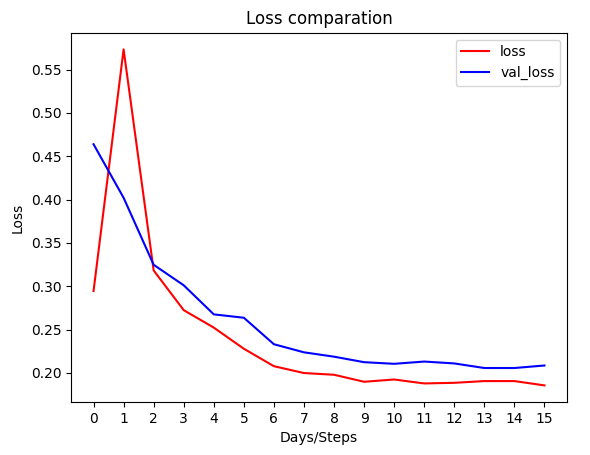
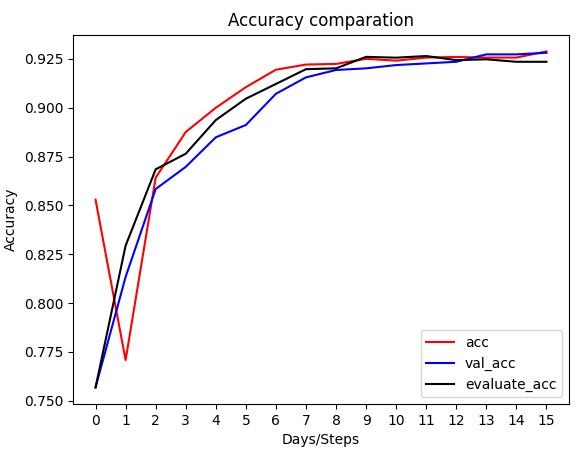


Figura - Resultados con 10 dispositivos federados

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

Esta ligera pérdida del accuracy 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

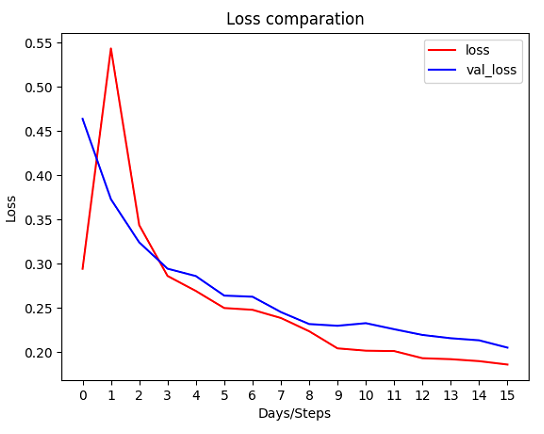
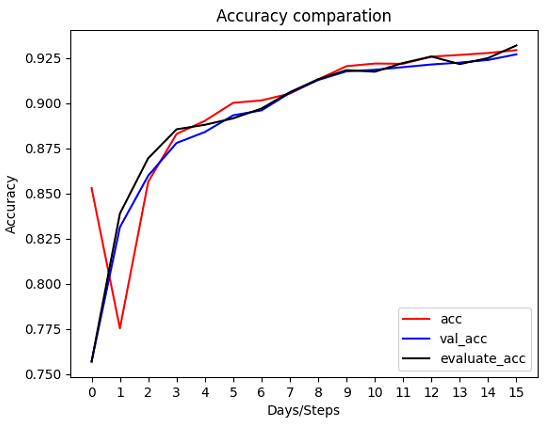


Figura - Resultados con 20 dispositivos federados

Los resultados permiten observar que, al igual que las experimentaciones anteriores, se observa una caída del accuracy en la primera etapa, lo cual refleja la readaptación del modelo federado y, además, se puede ver que en las etapas finales tanto el accuracy para evaluación, como el de entrenamiento y validación son bastante parejos, aunque se alcanza el máximo accuracy de la experimentación con un **93.19%**, donde además se cambia el 57.14% de las veces.

Por lo tanto, el número de veces no depende directamente del número de dispositivos y es probable que se deba al método de agregación y aceptación de modelos para el ensamblado.

Como resumen se puede ver la siguiente Tabla comparativa de los resultados de las experimentaciones:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | No federado | 5 dispositivos | 10 dispositivos | 20 dispositivos |
| Accuracy (%) | 96.91 | 92.73 | 92.88 | 92.93 |
| Validation accuracy (%) | 91.59 | 92.26 | 92.81 | 92.70 |
| Evaluate accuracy (%) | - | 92.77 | 92.35 | **93.19** |
| Cambios (%) | - | 67.10 | 53.64 | 40.19 |
| Tiempo de entrenamiento por dispositivo (h) | 1.25 | 1.37 | 1.39 | 2.39 |

Tabla - Comparativa de resultados

Los tiempos que se representan en la *Tabla 2* están formados por el tiempo que se tarda 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.

Se puede ver que, un mayor número de dispositivos requiere de un mayor tiempo de ejecución, sin embargo, ayuda a obtener una mayor generalización del modelo.

Por otra parte, los resultados muestran que un número pequeño de dispositivos obtendrá peores resultados que un número mayor, lo más probable debido a que se usan un menor número de imágenes totales en la etapa de entrenamiento, perjudicando a la agregación de un nuevo modelo en el servidor, no generalizando lo suficiente para seguir mejorando de forma más eficiente.

Para la experimentación se ha utilizado el FedAvg, aunque se ha probado una media federada ponderada, con objeto de comparar los datos y ver qué resultados son mejores en una experimentación real.

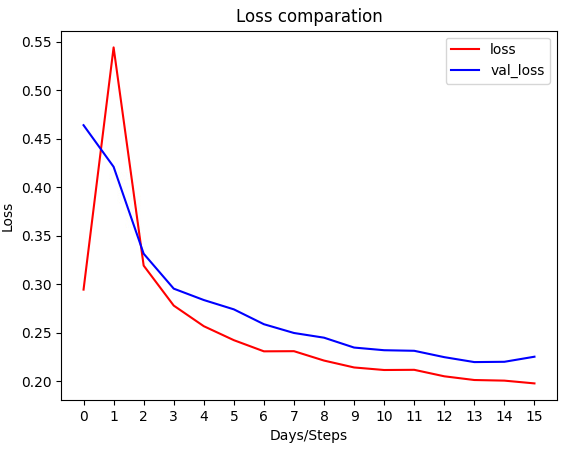
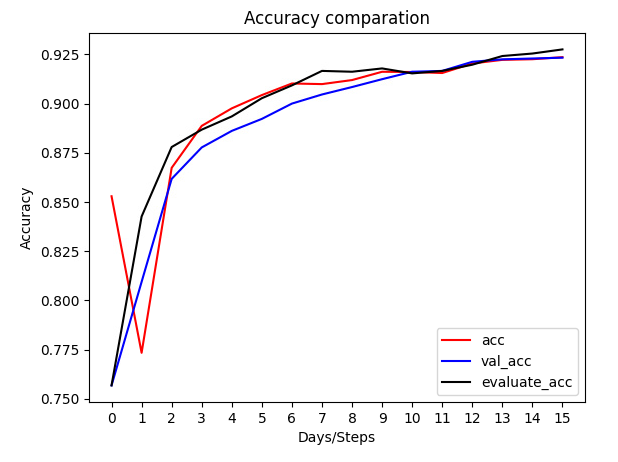


Figura - Resultados con 20 dispositivos federados con media ponderada

Se observa unos resultados aceptables que siguen siendo mejores que el modelo no federado y que obtiene un **92.35%** de accuracy, un **92.33%** de accuracy en validación y un **92.75%** de accuracy en la evaluación, pero que no llegan a mejorar el mejor modelo que usa FedAvg.

* 1. Puesta en marcha de la aplicación real

Tras toda la etapa de experimentación previa, se ha desarrollado una aplicación con el objetivo de poner en práctica el modelo que se ha generado del Sistema Federado en un entorno real, donde se pretende que el dispositivo conectado en este caso a una batería, haga la inferencia de pasos de peatones y carreteras en tiempo real mediante Video Streaming.

El video con la inferencia en tiempo real se puede encontrar en la siguiente url: <https://youtu.be/qQK1nb9n-tg>

En este vídeo se ha grabado lo que captura la cámara mostrando la etiqueta a la que corresponde, como una prueba experimental del objetivo final de todo este proyecto, donde uno se ha encargado de ir andando con la cámara en la posición central del pecho para poder simular como una persona camina y así detectar si va hacia un paso de peatones o una carretera. Mientras que el otro ha estado grabando la pantalla del portátil (conectado con Remote Desktop a la Raspberry Pi) para monitorizar que todo está funcionando correctamente.

Hay que recalcar que las etiquetas se cambian si hay al menos dos imágenes anteriores con la misma etiqueta para evitar saltos por inferencias erróneas por ello, en algunas ocasiones, se tarda en cambiar la etiqueta de la foto que se muestra en el video de la inferencia.

Igualmente, no se trata de tener un video con la inferencia en la aplicación final, sino que se ha tratado de que se vea como la inferencia actúa en un problema real, llegando a hacer una inferencia cada 0.5 segundos aproximadamente, aunque tarde más en cambiar la imagen que se muestra.

A continuación, unas capturas de las inferencias obtenidas en el video:

Figura - Inferencia de una carretera en la aplicación real



Figura - Inferencia de un paso de peatones en la aplicación real

Figura - Inferencia de una carretera en la aplicación real (II)

Capítulo 5. Conclusiones

En este último capítulo se abordará una conclusión final 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, con intención de que cualquiera que lea el Trabajo de Fin de Master, puedo interesarle avanzar por algunas de las ramas comentadas.

1. 1. Conclusiones del estudio

Los resultados obtenidos en la experimentación han sido muy destacables, logrando mejorar el accuracy inicial del modelo no federado de un **91.59%** a un **93.19%,** lo que supone un 1.6% de mejora gracias a la implementación del Sistema Federado.

Además, se ha puesto en práctica el desarrollo de un Sistema Federado que haga uso de la agregación de modelos de DL en una aplicación real, resolviendo incluso otras dificultades en la etapa de entrenamiento de modelos partiendo de un dataset pequeño. Esto deriva en una menor precisión en el modelo y supondría un peligro en una aplicación real de esta índole.

Igualmente, se ha hecho uso de sistema de Edge Computing, que proporciona un punto de vista distinto a los típicos sistemas de creación de modelos de DL basados en un servidor potente y partiendo datasets con una gran cantidad de imágenes de entrenamiento para conseguir buenos resultados.

Finalmente, se puede concluir que el sistema cumple el objetivo de diseñar e implementar un modelo de VA que sea eficiente y, que esté optimizado para su implementación en dispositivos con recursos limitados donde, además, se hace uso de una arquitectura de Aprendizaje Federado, y que su aplicación al mundo real es más que factible, lo que podría ayudar a resolver numerosos problemas de la actualidad como el que se propone, mejorar la autonomía de las personas con discapacidad visual.

* 1. Posibles mejoras y trabajos futuros

Durante la realización del Trabajo de fin de Master, han surgido ideas para mejorar esta aplicación.

En cuanto a las mejoras, 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, se puede trabajar con un entorno en el que la etapa de adaptación utilizase nuevas imágenes propias capturadas cada dispositivo, sin ser compartidas con el resto de dispositivos presentes en el sistema 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 daría un punto de vista distinto sobre el paso de modelos a un servidor junto con su implementación, además se puede conocer los tiempos de este paso de mensajes y cómo afectaría al sistema global.

En lo referente a las líneas futuras de investigaciones y desarrollos, se puede 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 mejor 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.
* Adaptación al entorno mediante feedback del usuario, por ejemplo, haciendo una conexión de los dispositivos usados con los bastones guías.

Referencias bibliográficas

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**[26]** IoT Url: <https://matob.web.id/news/most-popular-iot-technologies/>

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

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

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

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

**[31]** 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>

**[32]** Figura 2 – Diferenciación de tipos de Machine Learning Url: <https://www.ironhack.com/es/data-analytics/que-es-machine-learning>

**[33]** Figura 5 - Etapas del aprendizaje por capas. Url: <https://davidnez.wordpress.com/2017/10/30/el-deep-learning-una-nueva-carrera/>

**[34]** Figura 6 - 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/>

**[35]** Figura 7 – Ejemplo de VA. Url: <https://hqsoftwarelab.com/blog/computer-vision/>

**[36]** Figura 9 - Ejemplo de Edge Computing. Url: <https://es.wikipedia.org/wiki/Proceso_perimetral>

**[37]** Figura 10 - Capa convolucional. Url: <https://www.researchgate.net/figure/Outline-of-the-convolutional-layer_fig1_323792694>

**[38]** Figura 11 - 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]** Figura 12 - Workflow del proceso de una red neuronal. Url: <https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html>

**[40]** Figura 13 - Funcionamiento esquemático del Transfer Learning. Url: <https://www.v7labs.com/blog/transfer-learning-guide>

**[41]** Figura 14 – Arquitectura de MobileNetV2. Url: <https://www.researchgate.net/figure/The-architecture-of-the-MobileNetv2-network_fig3_342856036>

**[42]** Figura 17 - Sistema de Aprendizaje Federado de Google (Android GBoard). Url: <https://ai.googleblog.com/2017/04/federated-learning-collaborative.html>

**[43]** Figura 18 - Coral USB Acelerator. Url: <https://coral.ai/products/accelerator/>

**[44]** Figura 19 - Proceso de creación de modelo compatible con TPU Edge Url: <https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview>

**[45]** Figura 21 - Ejemplo de GitHub. Url: <https://www.nobledesktop.com/learn/git/git-branches>

**[46]** Figura 30 - Raspberry Pi Imager. Url: <https://www.raspberrypi.com/news/raspberry-pi-imager-imaging-utility/>

**[47]** Figura 31 - Instalación física de la Pi Camera. Url: <https://www.mouser.mx/new/adafruit/adafruit-pi-camera-board-v2/>

**[48]** Figura 34 - Ejemplo de Data Augmentation. Url: <https://medium.com/analytics-vidhya/data-augmentation-in-deep-learning-3d7a539f7a28>

**[49]** Figura 37 – Diferencias de learning rates. <Url:> <http://www.bdhammel.com/learning-rates/>

**[50]** Figura 38 - Ejemplo de 2D Average Pooling. <Url:> <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-average-pooling>

**[51]** Figura 39 - Ejemplo de Flatten. Url: <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/flatten>

**[52]** Figura 40 - Comparativa de funciones de activación. Url: <https://www.researchgate.net/figure/Plot-of-different-activation-functions-a-Sigmoid-activation-function-b-Tanh_fig4_339991922>