UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

**TFG del Grado en Ingeniería Informática**

# Aplicaciones de Visión Artificial en Dispositivos de Edge Computing Documentación Técnica

Presentado por Miriam Torres Calvo
en Universidad de Burgos — 10 de septiembre de 2022
Tutor: Bruno Baruque Zanón

# Índice general

**Índice general** i

**Índice de figuras** iii

**Índice de tablas** iv

**Apéndice A Plan de Proyecto Software** 1
A.1. Introducción . . . . . 1
A.2. Planificación temporal . . . . . 1
A.3. Estudio de viabilidad . . . . . 1

**Apéndice B Especificación de Requisitos** 3
B.1. Introducción . . . . . 3
B.2. Objetivos generales . . . . . 3
B.3. Catalogo de requisitos . . . . . 3
B.4. Especificación de requisitos . . . . . 3

**Apéndice C Especificación de diseño** 5
C.1. Introducción . . . . . 5
C.2. Diseño de datos . . . . . 5
C.3. Diseño procedimental . . . . . 5
C.4. Diseño arquitectónico . . . . . 5

**Apéndice D Documentación técnica de programación** 7
D.1. Introducción . . . . . 7
D.2. Estructura de directorios . . . . . 7
D.3. Manual del programador . . . . . 12

D.4. Compilación, instalación y ejecución del proyecto . . . . . 13
D.5. Pruebas del sistema . . . . . 15

**Apéndice E Documentación de usuario** **17**
E.1. Introducción . . . . . 17
E.2. Requisitos de usuarios . . . . . 17
E.3. Instalación . . . . . 17
E.4. Manual del usuario . . . . . 17

**Bibliografía** **19**

# Índice de figuras

# Índice de tablas

D.1. Bibliotecas utilizadas y sus versiones. . . . . 16

Apéndice $A$

# Plan de Proyecto Software

## A.1. Introducción

En este apéndice se va a mostrar la planificación del proyecto, la cuál es la base sobre la crea el proyecto *software*. Desde el punto de vista de la temporalidad y viabilidad. Siendo está una parte fundamental del proyecto, ya que permite visualizar el escenario en el que se desarrollará, de tal forma que podamos realizar una alineación estrategica de los elementos que deben de ser completados, con el objetivo de finalizarlo correctamente.

## A.2. Planificación temporal

La planificación temporal se

## A.3. Estudio de viabilidad

### Viabilidad económica

### Viabilidad legal

*Apéndice* $B$

# Especificación de Requisitos

## B.1. Introducción

En este apéndice se recogen las necesidades funcionales que deberán de ser soportadas por el sistema que va a ser desarrollado. Con el objetivo de obtener una buena documentación, deben de identificarse y describirse los requesitos que tienen que ser satisfacidos por el sistema, pero sin entrar en su proceso de realización.

## B.2. Objetivos generales

## B.3. Catalogo de requisitos

## B.4. Especificación de requisitos

*Apéndice* $C$

# Especificación de diseño

## C.1. Introducción

En este apéndice se va a exponer cómo se han resuelto los objetivos anteriormente comentados. Así como la definición de datos que se utilizan en la aplicación, procedimientos ...

## C.2. Diseño de datos

## C.3. Diseño procedimental

## C.4. Diseño arquitectónico

Apéndice $D$

# Documentación técnica de programación

## D.1. Introducción

En este apéndice van a describirse de forma detallada la documentación técnica de programación. Se describirá la estructura de directorios que posee, la instalación y ejecución, así como las pruebas que se han llevado a cabo.

El repositorio del proyecto puede ser consultado en el siguiente enlace: https://github.com/mtc1003/TF_Keras_TFG

## D.2. Estructura de directorios

- `/`: es la raíz del proyecto dónde se encuentran tanto el README, la licencia y las carpetas contenedoras del código, documentación y las pruebas previas.
- `/codigo`: es la carpeta que contiene todo el código funcional del proyecto.
- `/codigo/checkpoints`: es la carpeta contenedora de los modelos de detección en formato Tensorflow, Tensorflow Lite y Tensor-RT.
- `/codigo/checkpoints/custom-416`: carpeta que contiene el modelo de detección de las matrículas, que posee el tamaño 416.
- `/codigo/checkpoints/custom-416/saved_model.pb`: modelo en formato Tensorflow(.pb) de las matrículas.
- `/codigo/checkpoints/heads-416`: carpeta con el modelo de las cabezas en formato Tensorflow.

- `/codigo/checkpoints/heads-416/keras_metadata.pb`: punto de control del modelo de conversión a .pb.
- `/codigo/checkpoints/heads-416/saved_model.pb`: modelo detector de cabezas en formato Tensorflow(.pb)
- `/codigo/checkpoints/yolov4-416`: carpeta con el modelo oficial de YOLOv4 en formato Tensorflow.
- `/codigo/checkpoints/yolov4-416/keras_metadata.pb`: punto de control del modelo de conversión a .pb.
- `/codigo/checkpoints/yolov4-416/saved_model.pb`: modelo detector de YOLOv4 en formato Tensorflow(.pb)
- `/codigo/checkpoints/custom_tfl-416`: carpeta con el modelo de detección de las matrículas en formato Tensorflow previo a la conversióna TensorFlow Lite.
- `/codigo/checkpoints/custom_tfl-416/saved_model.pb`: modelo detector de matrículas en formato Tensorflow(.pb) preparado para su conversión a .tflite.
- `/codigo/checkpoints/custom_tflv2-416`: carpeta con el modelo de detección de las matrículas en formato Tensorflow previo a la conversióna TensorFlow Lite.
- `/codigo/checkpoints/custom_tflv2-416/saved_model.pb`: modelo detector de matrículas en formato Tensorflow(.pb) preparado para su conversión a .tflite.
- `/codigo/checkpoints/custom-416-int8.tflite`: modelo de detección de las matrículas en formato TensorFlow Lite(.tflite).
- `/codigo/checkpoints/custom-416v2.tflite.tflite`: modelo de detección de las matrículas en formato TensorFlow Lite(.tflite)
- `/codigo/checkpoints/models_trt.txt`: fichero con los enlaces de los modelos de TensorRT.
- `/codigo/core`: carpeta con los ficheros de configuración utilizados durante el proyecto.
- `/codigo/core/backbone.py`: fichero de Python que contine las funciones relacionadas con la red YOLOv4
- `/codigo/core/commom.py`: fichero de Python que contine la clase BatchNormalization, para los ajustes de la red YOLOv4
- `/codigo/core/config.py`: fichero de Python que contine permite la selección de los ficheros de etiquetas de cara al uso del modelo
- `/codigo/core/functions.py`: fichero de Python que contine las funciones utilizadas en a lo largo de la detección de los objetos.
- `/codigo/core/utils.py`: fichero de Python que contine las funciones relacionadas con la red YOLOv4.

- `/codigo/core/yolov4.py`: fichero de Python que retorna el modelo de YOLO correspondiente.
- `/codigo/data`: carpeta que contine la información necesaria para la detección.
- `/codigo/data/anchors`: carpeta que contiene los anchors de lass diferentes redes.
- `/codigo/data/anchors/basline_anchors.txt`: fichero de anchors.
- `/codigo/data/anchors/basline_tiny_anchors.txt`: fichero de anchors tiny.
- `/codigo/data/anchors/yolov3_anchors.txt`: fichero de anchors yolov3.
- `/codigo/data/anchors/yolov3_anchors.txt`: fichero de anchors yolov4.
- `/codigo/data/classes`: carpeta que contiene los fichero de etiquetas de los diferentes modelos.
- `/codigo/data/classes/coco.names`: fichero con las etiquetas del modelo oficial de YOLOV4.
- `/codigo/data/classes/custom.names`: fichero con las etiquetas del modelo de detección de matrículas.
- `/codigo/data/classes/heads.names`: fichero con las etiquetas del modelo de detección de las cabezas.
- `/codigo/data/classes/voc.names`: fichero de etiquetas del modelo de voc.
- `/codigo/data/classes/yymnist.names`: fichero de etiquetas del modelo de yymnist, detector de números.
- `/codigo/data/dataset`: carpeta que contiene los ficheros etiquetados a la hora de evaluar un modelo (ruta de la imagen posición detectada y valor de la clase).
- `/codigo/data/dataset/head.txt`: fichero de evaluacion del módelo de las cabezas.
- `/codigo/data/dataset/license_plate.txt`: fichero de evaluación del modelo de las matrículas.
- `/codigo/data/dataset/val2017.txt`: fichero de evaluación del modelo coco.
- `/codigo/data/images`: carpeta que contiene diferentes imagenes para su detección.
- `/codigo/data/video`: carpeta que contien diferentes vídeos para su detección/contabilización.
- `/codigo/deep_sort`: carpeta que contiene los diferentes ficheros en Python para su evaluacion con Object Tracking.

- `/codigo/deep_sort/detection.py`: fichero Python que contiene las funciones de detección para Obejct Tracking.
- `/codigo/deep_sort/iou_matching.py`: fichero Python que tiene las funciones de la maedida iou para Object Tracking.
- `/codigo/deep_sort/kalman_filter.py`: fichero Python que contiene el algoritmo del filtro de Kalman[1].
- `/codigo/deep_sort/linear_assignment.py`: Fichero Python que contiene las funciones relacionadas con la asignación linear.
- `/codigo/deep_sort/nn_matching.py`: Fichero Python con funciones de ajuste del algorimto de vecinos más cercanos[2].
- `/codigo/deep_sort/preprocessing.py`: Fichero Python con las funciones del preprocesado para Object Tracking.
- `/codigo/deep_sort/track.py`: fichero Python que contiene las funciones necearias para detectar los objetos y sus etiquetas correspondeitnes, con su respectivo número de identificación.
- `/codigo/deep_sort/tracker.py`: fichero Python que contiene las funciones necearias para detectar los objetos y sus etiquetas correspondeitnes, con su respectivo número de identificación.
- `/codigo/detections`: carpeta con el lso resultados de las detecciones obtenidas mediante la línea de comandos.
- `/codigo/detections/images`: carpeta con el resultado de las imagenes detectadas.
- `/codigo/detections/videos`: carpeta con el resultado de los vídeos detectados.
- `/codigo/mAP`: carpeta que contiene lso resultados de la evaluaciones de los modelos, así como scripts de ayuda para ello.
- `/codigo/mAP/extra`: carepeta con los cripts de ayuda para la evaluación del modelo.
- `/codigo/mAP/extra/intersect-gt-and-pred.py`: fichero Python que calcula la intersección entre la posición real del objeto y la obtenida por el modelo, con el objetivo de evaluar la calidad del modelo.
- `/codigo/mAP/extra/remove_space.py`: fichero Python que elimina lso espacios de las etiquetas de las clases de los modelos.
- `/codigo/mAP/ground-truth`: carpeta que contiene los ficheros .txt de cada imagen a evalaur con sus posiciones originales en formato YOLO, junto el nombre de la etiqueta que le corresponde.
- `/codigo/mAP/predicted`: carpeta que contiene los ficheros .txt de cada imagen a evaluar con sus posiciones detectadas en formato YOLO, junto el nombre de la etiqueta que le corresponde.
- `/codigo/mAP/results_custom_tf_complete`: carpeta con los resultados de la evalaución del modelo de las matrículas.

- `/codigo/mAP/results_heads_tf_complete`: carpeta con los resultados de la evalaución del modelo de las cabezas.
- `/codigo/mAP/main.py`: fichero Python que representa el resultado de la evalaución del modelo.
- `/codigo/model_data`: carpeta que contiene el modelo mars-small128.pb, utilizado en la inicialización de Obejct Tracking.
- `/codigo/static`: carpeta que contiene los ficheros 'estaticos' para Flask.
- `/codigo/static/css`: carpeta que contiene los diferentes ficheros de estilos[3] usados a lo largo de la app Flask.
- `/codigo/static/js`: carpeta que contiene los diferentes scripts de JavaScript[4] utilizados a lo largo de la app Flask.
- `/codigo/static/detections`: carpeta que contiene las imagenes, videos etiquetados tras su detección, así como los ficheros CSV de las posiciones.
- `/codigo/static/imgs`: carpeta con todas las imagenes usadas a lo largo de la app Flask.
- `/codigo/temp`: carpeta que almacena los ficheros de detección temporales, generados al inicio de las detecciones en la app Flask.
- `/codigo/templates`: carpeta que contienelos ficheros .html usados a lo alrgo de la app Flask.
- `/codigo/tools`: carpeta que contiene los scripts Python utilizados cómo herramientas a la hora de detectar.
- `/codigo/tools/freeze_model.py`: script Python que convierte el gráfico del modelo de TensorFlow a uno con extensión .pb.
- `/codigo/tools/generate_detections.py`: script Python que obtiene las 'cajas' en las cuáles se encuentran los objetos qu ehan sido detectados por el modelo.
- `/codigo/train`: carpeta que tiene los scripts de Python y de GoogleColab, así como los ficheros neecsarios para llevar a cabo el entrenamiento de un modelo de YOLOv4.
- `/codigo/trt`: carpeta que contiene el script de GoogleColab de conversión del fichero de pesos de YOLOv4 (.weights) a un modelo de TensorRT.
- `/codigo/app.py`: fichero de Python que es la propia app de Flask.
- `/codigo/convert_tflite.py`: fichero de Python que convierte el modelo deseado a uno de TensorFlow Lite.
- `/codigo/convert_trt.py`: fichero de Python que convierte el modelo deseado a uno de TensorRT.
- `/codigo/detect.py`: fichero de Python que detecta objetos en una imagen, según un modelo de detección.

- `/codigo/detectVideo.py`: fichero de Python que detecta objetos en una vídeo, según un modelo de detección.
- `/codigo/evaluate.py`: fichero de Python que evalua un modelo de detección, con el objetivo de medir su calidad a l hora de predeccir.
- `/codigo/objectTracker.py`: fichero de Python que contabiliza objetos en un vídeo, según un modelo de detección.
- `/codigo/preprocessDataEvaluate.py`: fichero de Python que obtiene las posiciones de las imágenes en el formato necesario apra su evalaución.
- `/codigo/save_model_tflite.py`: fichero de Python que convierte un fichero de pesos en formato .weights a un modelo de TensorFlow Lite.
- `/codigo/save_model.py`: fichero de Python que convierte un fichero de pesos en formato .weights a un modelo de TensorFlow.

## D.3. Manual del programador

En esta subsección se describen todos los recursos utilizados para poder llevar a cabo el proyecto. De tal forma que un futuro desarrollador/mantenedor del proyecto no tenga inconvenientes a la hora de retomar el proyecto y conocerlo.

### Entorno de desarrollo

Para poder continuar con el desarrollo del proyecto, será necesario contar con el siguiente *software* instalado en el equipo:

- Python 3.7
- Bibliotecas de Python
- VSCode

A continuación, se comentára de forma detallada la instalación de los diferentes requerimientos.

### Python 3.7

La versión de Python 3.7, se encuentra dispnible desde [5]. Es muy importante que los ficheros binarios se encuentren en el PATH del sistema, para evitar así posibles problemas de ejecución.

### Bibliotecas de Python

Este punto, es de los más importantes para hacer funcionar el proyecto, ya que son necesarias unas determiandas librerías y en unas versiones concretas, para que todo se integre correctamente y así funcione todo cómo un sistema homogéneo. Ver Tabla D.1

Las versiones que se indican en la Tabla D.1, son las que se han utilizado a lo largo del desarrollo del proyecto, las cuáles pueden ser actualizadas a versiones futuras, siempre y cuando estás sean compatibles entre sí o con las páginas web con las que trabajan por debajo.

## D.4. Compilación, instalación y ejecución del proyecto

En esta sección, se va a detallar el proceso a seguir para poder poseer el proyecto en local, y así poder utilizarlo y/o modificarlo.

### Adquisición del código fuente

El primer paso, es la obtención del código en el equipo, para ello podremos seguir una de las siguientes aproximaciones:

- Mediante el uso de la terminal.

  1. Apertura de la terminal.
  2. Desplazarse al directorio en donde se desee clonar el repositorio (usando `cd` en Unix o `dir` en Windows).
  3. Hacer uso del siguiente comando:
     `git clone https://github.com/mtc1003/TF_Keras_TFG.git`
  4. Se dispone de una copia idéntica a la alojada en el repositorio de `GitHub`.

- Descarga desde el navegador.

  - Apertura del navegador de preferencia.
  - Introducir en la barra de búsqueda la siguiente dirección:
    `https://github.com/mtc1003/TF_Keras_TFG/archive/refs/heads/master.zip`
  - Aceptar la descarga en caso de tener habilitada la comprobación.
  - Navegar con el Explorador de archivos del sistema hasta el directorio de descarga.

- Uso de `Fork`.
  - Apertura de la aplicación.
  - Hacer *click* en *File* y del desplegable de opciones seleccionar *Clone*.
  - Dentro de la ventana de *Clone*:
    - En *Respository Url* introducir: https://github.com/mtc1003/TF_Keras_TFG.git.
    - En *Parent Folder* introducir: la ruta en la que se clonará el repositorio en local.
    - En *Name* introducir: el nombre que recibirá el proyecto en lcoal.
  - Hacer *click* en *Clone*.

## Creación de entorno virtual de trabajo

Para poder trabajar con este proyecto (independientemente de si es para desarrollo o producción) hacen falta una serie de bibliotecas concretas de Python (en unas versiones determinadas), las cuáles, como es lógico, deben estar en la máquina en la que se va a ejecutar, es decir, en la que se encuentra el código. El proyecto está preparado para crear un entorno de `Conda` propio, de forma que no interfiera con otros proyectos y sea más sencillo de mantener y actualizar.

Se recomienda que los binarios de anaconda o miniconda estén configurados en el `path` del sistema para poder utilizar el comando `conda` desde la línea de comandos.

El proceso de creación del entrono virtual con `Conda` (trabajar con CPU) es el siguiente:

1. Apertura de la terminal.
2. Navegar hasta la raíz del proyecto.
3. Crear el entorno con:
   `conda env create -f OD_MTC_CPU.yml`
4. Cuando se desee utilizar se debe activar:
   `conda activate ODMTC`

Por otro lado, el proceso de creación del entrono virtual con `Conda` (trabajar con GPU) es el siguiente:

1. Apertura de la terminal.

2. Navegar hasta la raíz del proyecto.
3. Crear el entorno con:
   `conda env create -f OD_MTC_GPU.yml`
4. Cuando se desee utilizar se debe activar:
   `conda activate ODMTC_GPU`

También se puede utilizar el procedimiento habitual para importar las bibliotecas al actual `venv` de la sesión de la terminal, pero se desaconseja su uso ya que un entorno 'genérico' antes o después se actualizará por otros proyectos, pudiendo generar incompatibilidades con el proyecto.

## D.5. Pruebas del sistema

En está sección se van a describir las pruebas que se realizan con un modelo entrenado previamente, para así concoer su fiabilidad.

| Biblioteca | Versión | Descripción |
|---|---|---|
| `absl-py` | 0.15.0 | Creación de aplaicaciones sencillas. |
| `flask` | 1.1.2 | Web *framework*. |
| `numpy` | 1.22.3 | Computación de *arrays*. |
| `pandas` | 0.25.1 | Estructuras de datos. |
| `requests` | 2.27.0 | *Requests* para humanos. |
| `keyboard` | 0.13.5 | Interacción del teclado desde Python |
| `pafy` | 0.5.5 | Recuperar contenido y metadatos de YouTube |
| `youtube-dll` | 2020.12.2 | Descargar vídeos de Youtube junto con su información |
| `pandas` | 1.3.5 | Potentes estructuras de datos para análisis de datos, series |
| `numpy` | 1.21.6 | Paquete de computación matricial |
| `opencv-python` | 4.1.1.26 | Visión Artificial para el lenguaje Python |
| `tensorflow` | 2.8.0 | Framework dw Machine Learning |
| `tensorflow-gpu` | 2.3.0 | Framework de Machine Learning para GPU |
| `pillow` | 9.2.0 | Librería de imágenes |
| `easydict` | 1.9 | Acceso a los valores de un *dict* como atributos |
| `matplotlib` | 3.5.3 | Trazado en Python |

Tabla D.1: Bibliotecas utilizadas y sus versiones.

*Apéndice* $E$

# Documentación de usuario

## E.1. Introducción

En este apéndice van a detallarse los requerimientos de la aplicación, su instalación y consejos de cara a usarlo de manera correcta.

## E.2. Requisitos de usuarios

## E.3. Instalación

## E.4. Manual del usuario

# Bibliografía


[1] ScienceDirect, "Kalman filter," 2019. [Online]. Available: https://www.sciencedirect.com/topics/earth-and-planetary-sciences/kalman-filter#:~:text=The%20Kalman%20Filter%20is%20an,the%20uncertainty%20of%20the%20estimates.

[2] IBM, "K-nearest neighbors algorithm." [Online]. Available: https://www.ibm.com/topics/knn

[3] Mozilla, "What is css?" [Online]. Available: https://developer.mozilla.org/es/docs/Learn/CSS/First_steps/What_is_CSS

[4] Moxilla, "Javasript." [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript

[5] "Python download," https://www.python.org/getit/. [Online]. Available: https://www.python.org/getit/