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 — 12 de septiembre de 2022
Tutor: Bruno Baruque Zanón

Índice general

Índice de figuras

Índice de tablas

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` 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.yml`
4. Cuando se desee utilizar se debe activar:
   `conda activate ODMTC`

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 conocer su fiabilidad.

Lo primero será seleccionar el fichero de etiquetas con el que deseamos trabajar para evaluación, para ello se deberá de modificar la línea *__C.YOLO.CLASSES* del fichero *config.py*, con el fichero de etiquetas con las clases del modelo a evaluar.

```
# YOLO options
__C.YOLO                        = edict()

__C.YOLO.CLASSES                = "./data/classes/heads.names"
```

Figura D.1: Selección del fichero de etiquetas

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.

Tras esto deberemos de crear una carpeta con las imágenes, con las que deseamos evaluar el modelo y con los ficheros *txt* de cada imagen con las posiciones de los objetos detectados

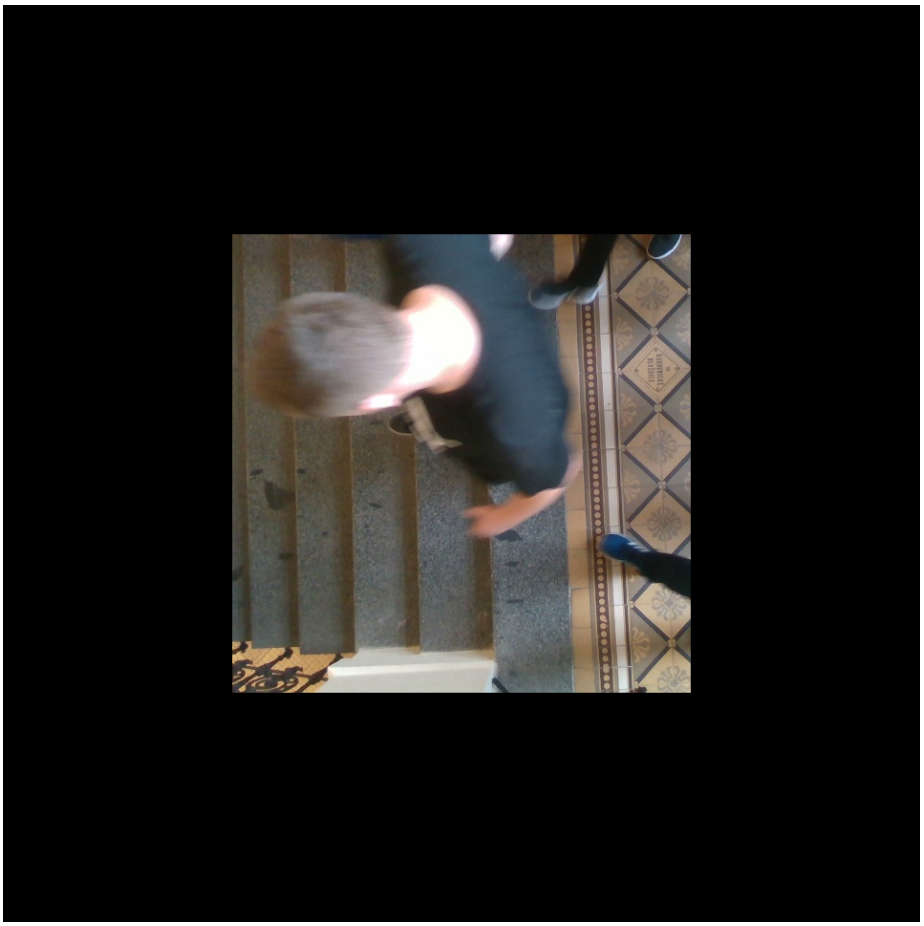

Figura D.2: Ejemplo de imágen que se usará para evaluar el modelo

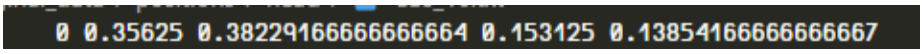

Figura D.3: Ejemplo de etiqueta que se usará para evaluar el modelo

Tras esto, deberemos ejecutar el script *preprocessDataEvaluate.py*, el cuál nos devolverá un único fichero con toda la información de las imágenes en formato YOLO.

```
python .\preprocessDataEvaluate.py --positions .\original_data\positions\head\ --name_txt_export head1.txt
```

Figura D.4: Comando *preprocessDataEvaluate.py* con el flag *positions* con la ruta de las imágenes y el flag *name_txt_export* con el nombre del fichero que se exportará

El fichero que obtendremos será de este estilo:

```
.\original_data\positions\head\CBs_100.png 175,427,305,539,0
.\original_data\positions\head\CBs_101.png 265,164,412,298,0 160,423,285,536,0
.\original_data\positions\head\CBs_102.png 137,444,268,557,0 200,183,409,353,0 174,685,306,779,0
.\original_data\positions\head\CBs_103.png 154,447,272,561,0 170,684,315,795,0 185,165,409,380,0
.\original_data\positions\head\CBs_104.png 223,272,404,447,0 131,467,268,571,0 174,679,332,809,0
.\original_data\positions\head\CBs_105.png 224,661,380,790,0 236,349,418,503,0
.\original_data\positions\head\CBs_106.png 240,361,415,521,0 256,652,390,766,0
.\original_data\positions\head\CBs_107.png 220,373,411,530,0 269,655,409,770,0
.\original_data\positions\head\CBs_108.png 224,373,395,526,0 284,649,445,763,0
.\original_data\positions\head\CBs_109.png 321,655,471,773,0 215,372,392,523,0
.\original_data\positions\head\CBs_110.png 609,136,737,273,0 206,355,358,506,0 454,661,599,768,0
.\original_data\positions\head\CBs_111.png 162,340,336,490,0
.\original_data\positions\head\CBs_112.png 171,338,330,474,0
.\original_data\positions\head\CBs_113.png 153,590,280,703,0 151,326,313,461,0
.\original_data\positions\head\CBs_114.png 140,317,277,451,0 168,624,339,781,0 694,441,814,558,0
.\original_data\positions\head\CBs_115.png 666,433,816,549,0 171,634,365,807,0 134,323,265,454,0
.\original_data\positions\head\CBs_116.png 612,394,787,532,0 230,662,405,788,0 128,332,251,460,0
.\original_data\positions\head\CBs_117.png 559,378,721,522,0 311,684,448,827,0
.\original_data\positions\head\CBs_118.png 546,383,706,514,0 337,691,478,825,0
.\original_data\positions\head\CBs_119.png 519,406,673,557,0
.\original_data\positions\head\CBs_12.png 432,192,585,354,0
.\original_data\positions\head\CBs_120.png 520,455,660,599,0
.\original_data\positions\head\CBs_121.png 506,478,653,608,0
.\original_data\positions\head\CBs_122.png 508,515,649,651,0
.\original_data\positions\head\CBs_123.png 479,560,628,695,0
.\original_data\positions\head\CBs_124.png 486,563,621,704,0
.\original_data\positions\head\CBs_125.png 466,580,613,721,0
.\original_data\positions\head\CBs_126.png 465,625,608,771,0
.\original_data\positions\head\CBs_127.png 462,655,601,782,0
.\original_data\positions\head\CBs_128.png 170,679,275,805,0 145,138,271,269,0
.\original_data\positions\head\CBs_129.png 143,635,308,805,0 146,160,272,304,0
.\original_data\positions\head\CBs_13.png 412,212,571,377,0
.\original_data\positions\head\CBs_130.png 164,597,336,778,0 133,166,268,310,0
.\original_data\positions\head\CBs_131.png 231,521,402,658,0
.\original_data\positions\head\CBs_132.png 337,446,498,597,0
.\original_data\positions\head\CBs_133.png 363,437,529,598,0
.\original_data\positions\head\CBs_134.png 409,438,578,592,0
.\original_data\positions\head\CBs_135.png 496,421,654,575,0
.\original_data\positions\head\CBs_136.png 530,393,700,561,0
.\original_data\positions\head\CBs_137.png 582,303,756,471,0 313,638,459,776,0
.\original_data\positions\head\CBs_138.png 304,576,439,703,0 616,212,774,403,0
.\original_data\positions\head\CBs_139.png 642,176,803,360,0 306,548,446,664,0
.\original_data\positions\head\CBs_14.png 364,277,512,423,0
.\original_data\positions\head\CBs_140.png 291,511,422,631,0
.\original_data\positions\head\CBs_141.png 280,475,411,601,0
.\original_data\positions\head\CBs_142.png 278,468,406,582,0
```

Figura D.5: Fichero con todas las imágenes y sus posiciones en formato YOLO junto con la clase del objeto detectado

En él, encontaremos el PATH a la imagen, la posición del objeto detectado (como x, y, width, height) y el valor de la etiqueta con la que se corresponde el objeto, estos dos últimos valores se repetirán tantas veces como objetos halla en la imagen. El fichero contará con tantas líneas como imagenes para evaluar tengamos en la carpeta.

Con este fichero ya podremos evalaur el modelo, para ello usaremos el script *evaluate.py*

```
python evaluate.py --weights .\checkpoints\heads-416 --annotation_path .\data\dataset\head1.txt
```

Figura D.6: Script *evaluate.py* junto con los flags *weights* con el modelo que se quiere evaluar y *annotation_path* con el PATH del fichero obtenido previamente

Durante la ejecución de la evalaución obtendremos por pantalla el siguiente resultado: Donde contaremos con las posiciones originales, dónde por cada detección veremos el nombre de la clase, así como las posiciones de la imagen, y las posiciones detectadas dónde tendremos el nombre de la clase, el *accuraccy* de la detección y la psoción en la que se encuentra el objeto.

```
=> ground truth of .\original_data\positions\head\CBs_988.png:
        head 672 480 826 645
=> predict result of .\original_data\positions\head\CBs_988.png:
        head 0.6703 661.0 475.0 853.0 646.0
```

Figura D.7: Ejemplo del resultado de una imagen

Por último, para ver el resultado, tendremos que movermos a la carpeta *mAP* y ejecutar el script *main.py*. Donde podremos ver el resultado del *mAP*

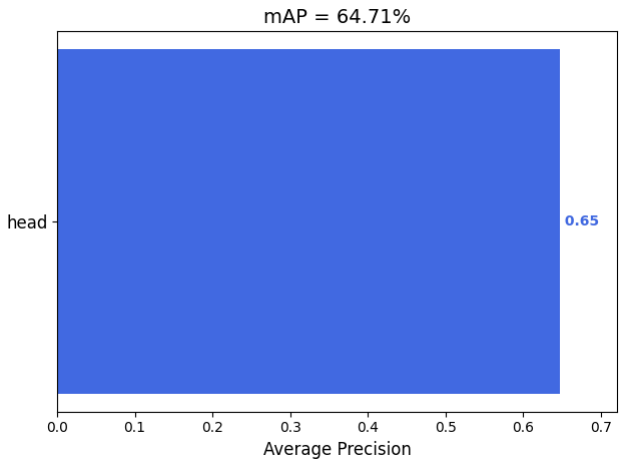

Figura D.8: mAP del modelo de detección de las cabezas

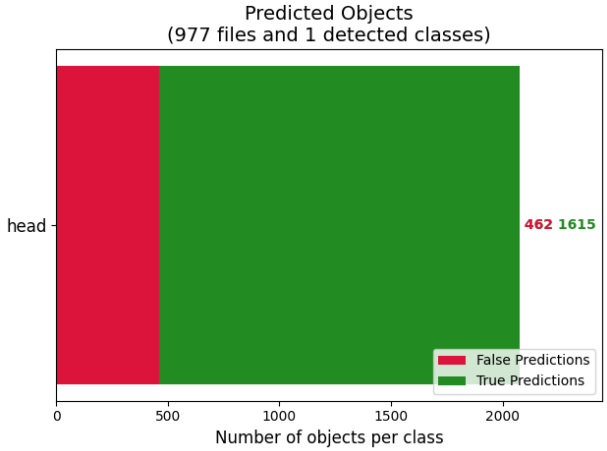

Figura D.9: información sobre la evaluación del modelo

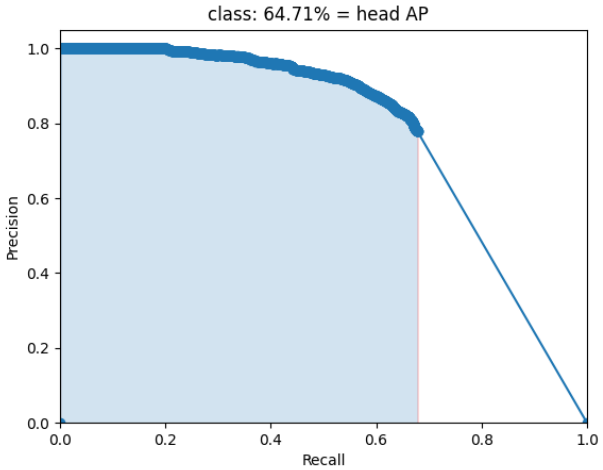

Figura D.10: Precission-Recall de la clase *head*

Apéndice E

Documentación de usuario

E.1. Introducción

En esta sección se detallan los requerimientos de la aplicación, los pasos de instalación, despliegue, así como las indicaciones para su uso adecuado.

E.2. Requisitos de usuarios

Los requesitos para poder hacer uso de la aplicación son:

- Tener instalado Python (minimo la version 3.7) y el resto de las librerías contenidas en el *requeriments.txt*.
- Tener un navegador web compatible con HTML5

E.3. Instalación

Instalar el contenedor docker de la aplicación.

E.4. Manual del usuario

En esta sección se explicarán las diferentes tareas que puede hacer un usario en la aplicación.

Figura E.1: *Home* de la aplicación

Carga de modelo de detección

Esta opción se encuentra en el *home* de la aplicación, correspondiendose esta opción con el primer botón, al hacer *click* sobre él se nos abrirá una modal, la cuál cuenta con dos inputs, uno de tipo text y otro de tipo file(este posee funcionalidad de *click* y de *drag and drop*).

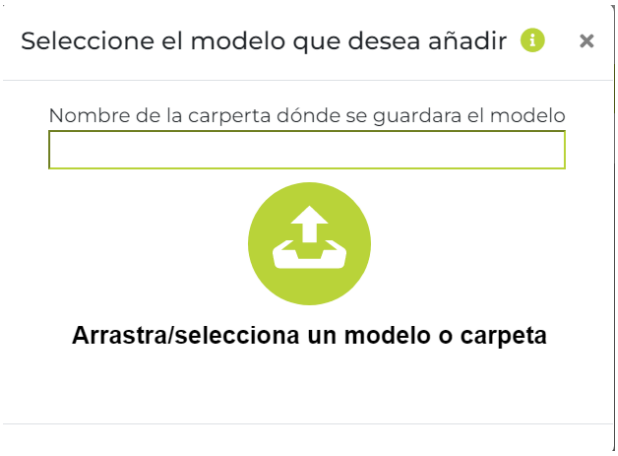

Figura E.2: Modal para añadir modelos

El input de tipo *text*, sirve para introducir el nombre de la carpeta en la cuál se guardará el modelo de detección, de tal forma que sea más fácil encontrarlo posteriormente. En segundo input de tipo *file*, nos permite

escoger el modelo que queremos añadir a la aplicación, ya sea un modelo de *Tensorflow Lite* o una carpeta con el modelo de *Tensorflow*.

Carga de fichero de etiquetas

Está opción se corresponde con el segundo botón de acciones del *home*, al hacer *click* sobre él se abrirá una modal, la cuál, es de un estilo similar a la modal de añadir modelos, en está encontraremos un input de tipo *file*, el cuál funciona sólo por *click*, el cuál permite la carga de archivos cuya extensión sea *names*

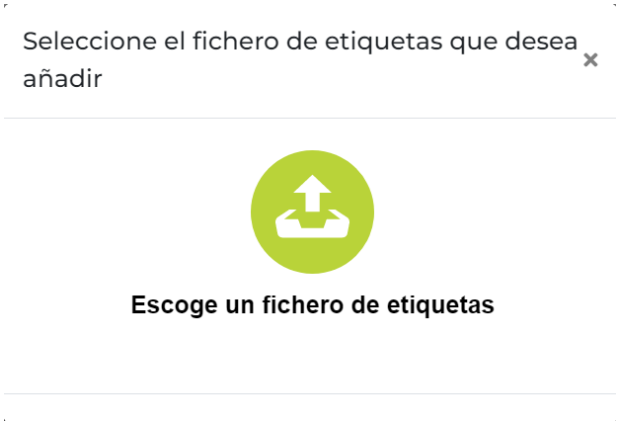

Figura E.3: Modal para añadir ficheros de etiquetas

Cambiar modelo de detección

Cambiar fichero de etiquetas

Detección de objetos en una imagen

Detección de objetos en un vídeo

Contabilizar objetos en un vídeo

Detección de objetos a tráves de la URL de una imágen

Detección de objetos a tráves de la URL de un vídeo de YouTube

Detección de objetos a tráves de webcam

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/