Universidad Zaragoza

Trabajo de Fin de Grado

# ANÁLISIS Y OPTIMIZACIÓN DE UN PIPELINE DE VIDEOVIGILANCIA


AUTOR

Borja Aguado Díez

PONENTE

Darío Suárez Gracia

DIRECTOR

Francisco Feijoo Cano


ENERO 2021

# RESUMEN

En los últimos años, se ha experimentado un desarrollo enorme en el campo de la detección y clasificación de objetos. Ahora es posible dotar a un sistema de videovigilancia de la capacidad de distinguir los objetos que está monitorizando, pudiendo actuar en consecuencia. En parte, este avance ha sido posible gracias a las redes neuronales. Estas se han beneficiado de la enorme cantidad de imágenes etiquetadas disponibles para su entrenamiento y a la gran potencia computacional del hardware actual, lo que ha permitido que las redes neuronales alcancen una gran precisión en la detección de objetos.

Existen muchos modelos de redes neuronales diseñados para tener la mejor precisión. Estos modelos utilizan técnicas novedosas que aumentan poco a poco las necesidades computacionales del modelo, por lo que consiguen avances en precisión cada muy poco tiempo. No obstante, en general, las redes más precisas suelen ser ejecutadas en aceleradores o en GPUs para alcanzar un rendimiento razonable. Sin embargo, en la práctica, no es posible disponer de una GPU de alta gama para cada proyecto de detección de objetos debido a su elevado coste. Además, la precisión que ofrecen es muy elevada, y en muchos casos se puede prescindir de precisión a cambio de reducir la carga computacional.

Este trabajo explora alternativas basadas en redes neuronales a las técnicas de detección existentes en un sistema de videovigilancia, centrándose en la optimización de la red y en el hardware que será capaz de ejecutarla. Se va a centrar en el sistema de videovigilancia de la empresa Buavi.

Las GPUs convencionales consumen mucha energía y ocupan mucho tamaño. No siempre es posible disponer de una GPU para realizar inferencia, por lo que en ciertos entornos es necesario usar CPUs que permitan clasificar objetos con una precisión suficiente a la vez que mantienen el consumo de energía bajo. Normalmente estas soluciones son necesarias en entornos remotos o en equipos ya instalados que no dispongan de GPU u otros aceleradores.

El resultado del trabajo ha sido el despliegue de una red que se puede ejecutar sobre CPU, y que permite detectar personas. La red es capaz de monitorizar varias cámaras a la vez, por lo que se han alcanzado los objetivos planteados al comienzo de este proyecto.

# ÍNDICE

# 1 INTRODUCCIÓN

## 1.1 MOTIVACIÓN

Las redes neuronales utilizadas para la detección de objetos funcionan muy bien, se ha conseguido avanzar enormemente los últimos 10 años. No obstante, los resultados óptimos suelen conseguirse utilizando redes muy extensas, que requieren de mucha capacidad computacional para realizar la inferencia de una imagen.

> "Convolutions are so compute hungry that they are the main reason we need so much compute power to train and run state-of-the-art neural networks."
>
> Alex Burlacu [1]

Si lo que se quiere es detección en tiempo real, es decir, procesar el vídeo a la misma velocidad con la que se está captando, el estándar suele ser procesar 24 imágenes por segundo, con imágenes de 512x512. Sin embargo, para gran cantidad de aplicaciones, no se necesitan procesar tantos fotogramas por segundo. Para un sistema de videovigilancia común, en el que no hay objetos moviéndose demasiado rápido, con 4 FPS es suficiente. Si se quisieran detectar objetos que se mueven más rápido, habría que aumentar la frecuencia de detección.

El hardware necesario para procesar las imágenes en las redes más populares como YOLO [2] o R-CNN [3] son GPUs de gama alta. Si lo que se necesita es detección a tiempo real en una cámara, por ejemplo, tener que adquirir una GPU de gama alta podría ser algo desmesurado. Existe hardware alternativo, como el acelerador Google Coral [4] o aceleradores gráficos integrados en la CPU de los móviles. Estos pueden ejecutar redes más pequeñas con menos precisión, pero no tienen la potencia suficiente para utilizar las redes más grandes y precisas.

Por este motivo, en este trabajo se intentan explorar las posibles optimizaciones, redes más pequeñas, y hardware alternativo para conseguir una detección a tiempo real, pero sin el consumo y el coste que supone una GPU de altas prestaciones.

## 1.2 OBJETIVOS Y PLANIFICACIÓN

El objetivo de este trabajo es conseguir un sistema de clasificación de objetos basado en redes neuronales, que esté lo más optimizado posible de cara a su posible despliegue a nivel comercial. La empresa que utilizará este sistema desarrolla equipamiento, entre otras cosas, de sistemas de seguridad. Por lo tanto, es importante que el resultado sea fiable, y que al mismo tiempo, consuma la menor cantidad de recursos posible, debiendo funcionar con imágenes en tiempo real. Con este objetivo en mente, se ha dividido el trabajo en 6 tareas:

1. Análisis del estado del arte en aprendizaje automático para videovigilancia.
2. Experimentación y puesta en funcionamiento de una red neuronal sencilla.
3. Uso del dataset Open Images Dataset [5] para el entrenamiento de redes neuronales, en paralelo con la realización de pruebas y evaluación de rendimiento de los distintos tiempos de redes neuronales.
4. Familiarización con distintos tipos de hardware según su potencia y coste.
5. Evaluación del rendimiento de distintos tipos de hardware con las redes neuronales.
6. Extracción de resultados y conclusiones.

## 1.3 ALCANCE

La solución de la empresa para la detección de personas en su sistema de videovigilancia emplea detección clásica de movimiento (cambio en píxeles de un fotograma a otro) y algoritmos de tracking en la imagen, antes de utilizar el módulo de detección y clasificación de objetos.

Este tipo de técnicas son sencillas y efectivas, pero presentan ciertos inconvenientes; el algoritmo no puede determinar si lo que aparece en la imagen es de interés para el sistema de videovigilancia o no. Para estos algoritmos no existe diferencia entre un intruso y, por ejemplo, un pájaro. Además, ciertos artefactos de vídeo pueden desencadenar la detección de movimiento sin que haya nada relevante que detectar.

Sin embargo, combinarlas con redes neuronales en el módulo de detección solventa esos inconvenientes. El algoritmo estima qué es lo que aparece en el vídeo, y puede reaccionar de manera distinta dependiendo de si es un gato o un coche. La empresa ya utiliza una

solución basada en redes neuronales que se va a explicar más adelante. Este trabajo se ha enfocado a la optimización de este módulo para conseguir más precisión con menor coste.

Existen soluciones fáciles de implementar que funcionan muy bien pero que requieren gran potencia computacional. Como algunos de los clientes de la empresa ya disponen de su infraestructura hardware sin GPUs de alta gama, se han buscado soluciones que requieran únicamente de CPUs para hacer el cómputo y así poder dar soporte en dichas plataformas.

## 1.4 ESTRUCTURA DEL DOCUMENTO

En los siguientes capítulos se tratará más en profundidad el estado del arte de las redes neuronales convolucionales. Se describirán los principales algoritmos de detección existentes, y las diferencias entre las implementaciones de estos. También se analizarán los datasets disponibles de forma pública para entrenar las redes.

A continuación, se hablará de la metodología y se introducirán ciertos conceptos importantes para entender la solución propuesta, seguidos de explicaciones sobre con qué hardware y de qué modo se han realizado las pruebas.

Por último, se hablará en detalle de la solución propuesta, explicando los pasos realizados hasta la obtención de un modelo óptimo que resuelva los problemas planteados. Se evaluarán los resultados obtenidos y se comentará brevemente la integración de un modelo final en el sistema de videovigilancia de la empresa Buavi.

Cabe destacar que este trabajo se centra en optimizar la inferencia de las imágenes, no el entrenamiento del modelo.

# 2

# ESTADO DEL ARTE

Este capítulo se centra en la tecnología utilizada actualmente para desarrollar sistemas basados en Redes Neuronales Convolucionales (RNC). Se abarcan distintos algoritmos, sus implementaciones, y algunos conjuntos de datos existentes para entrenar un modelo.

> "The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm — a Convolutional Neural Network."
>
> Sumit Saha [6]

## 2.1 REDES NEURONALES PARA LA DETECCIÓN

En las redes neuronales, las capas más importantes, y las que contienen más información, son las capas convolucionales. Estas aplican ciertas operaciones a la información de la capa anterior y la transmiten a la siguiente capa. Estas operaciones suelen ser multiplicaciones de parámetros de la red, y en redes modernas existen millones de parámetros. Para acelerar el procesamiento de estas capas, las operaciones se transforman en una multiplicación de matrices, que se puede realizar rápidamente gracias a las extensiones vectoriales avanzadas en CPUs [7] o mediante el paralelismo masivo de datos de las GPUs o ASICs.

Existe una gran cantidad de algoritmos para la clasificación de objetos basados en RNC. En este trabajo se ha decidido utilizar YOLO (You Only Look Once) [2], debido a su popularidad y su implementación en distintos frameworks. No obstante, se han estudiado otras alternativas que se describen a continuación.

### 2.1.1 *You Only Look Once: YOLO*

YOLO es un algoritmo de detección y clasificación de objetos creado en 2015. Se convirtió rápidamente en uno de los algoritmos más populares, y fue evolucionando hasta llegar a su cuarta versión en 2020 (YOLOv4) [8].

El éxito que tuvo fue debido en parte al hecho de ser un detector de una fase, en contraposición a detectores de dos fases como R-CNN [3], que eran los más utilizados antes de la publicación de YOLO.

Estos detectores de dos fases separan el problema de detección en dos etapas: la extracción de zonas de interés y la clasificación de objetos en dichas zonas. Cada zona de interés detectada en la primera fase se pasa a la red neuronal, como se aprecia en la Figura 2.1. Esto retrasa mucho la detección porque suelen extraerse del orden de 2000 zonas de interés.

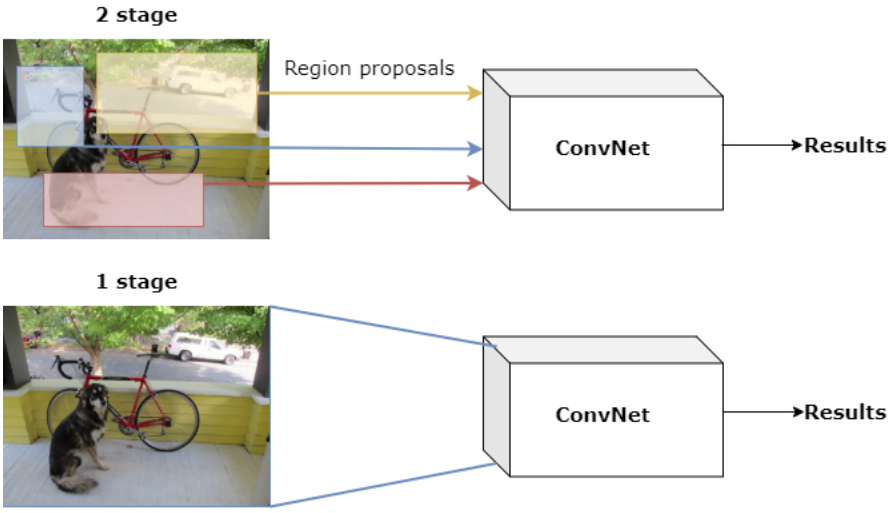



Figura 2.1: Diferencias entre detectores de dos fases (arriba) y de una fase (abajo).

En contrapartida, detectores de una fase como YOLO realizan la inferencia de la imagen en un solo paso, con una única red neuronal convolucional, por lo que son significativamente más rápidos.

### 2.1.2 *Otras Redes*

Una alternativa bastante conocida es Faster R-CNN [9], la evolución de R-CNN. Al ser bastante conocida, existen implementaciones en frameworks como Keras [10], que facilitan mucho el desarrollo de pruebas y la implementación de un detector funcional.

Otra alternativa que se ha estudiado es CenterNet [11]. Se trata, al igual que YOLO, de un detector de una fase. Además de predecir la localización del objeto, predice su centro, que utiliza para determinar con mayor precisión si la detección del objeto ha sido correcta o no.

## 2.2 FRAMEWORKS DE TRABAJO CON REDES

### 2.2.1 *Darknet*

Darknet [12] es el framework original que utiliza el algoritmo YOLO para la detección y clasificación de objetos. Está desarrollado por los autores de este algoritmo, y se ha implementado en C y CUDA.

Este framework es extremadamente popular. Se trata del framework más utilizado a la hora de entrenar modelos, y existe una gran comunidad participando activamente en añadir nuevas funcionalidades y optimizar su funcionamiento.

### 2.2.2 *Tensorflow*

Existen más frameworks, además de Darknet, que implementan el algoritmo YOLO. Una de las herramientas más utilizadas es Tensorflow [13].

Tensorflow es una biblioteca de código abierto creada por Google, orientada a desarrollar aplicaciones con aprendizaje automático. Cuenta también con muchos usuarios activos y es una de las herramientas que se han planteado utilizar en este trabajo. El lenguaje más extendido para desarrollar aplicaciones con esta biblioteca es Python.

Al ser un lenguaje compilado, C ofrece más rapidez a la hora de ejecutar programas. Por otra parte, utilizar Python y Tensorflow para programar aplicaciones de aprendizaje automático es mucho más sencillo y rápido que hacerlo en C. Por estas razones, en este trabajo se han explorado ambas opciones.

### 2.2.3 *OpenVino*

Intel ha desarrollado un kit de herramientas llamado OpenVino [14]. Estas herramientas permiten optimizar diferentes modelos de redes neuronales para que se ejecuten en procesadores Intel, aprovechando al máximo su arquitectura y el repertorio de instrucciones de estas CPUs. Permite transferir modelos de frameworks distintos a su propio formato.

Además de los frameworks mencionados anteriormente, este trabajo también analiza los resultados obtenidos con este kit de herramientas, comparándolos a los demás.

## 2.3 HARDWARE

Para el entrenamiento de modelos basados en RNC hace falta mucha potencia computacional, por lo que se utilizan GPUs muy potentes. Las más utilizadas son las de la serie GeForce RTX de NVIDIA, siendo el modelo RTX 3090 la más potente hasta la fecha, y la más recomendada para esta tarea.

> "Nvidia GPUs are widely used for deep learning because they have extensive support in the forum software, drivers, CUDA, and cuDNN."
>
> Prathmesh Patil [15]

Para realizar inferencias una vez el modelo está entrenado, GPUs como la Jetson Nano de NVIDIA son muy populares, porque son más potentes que una CPU sin costar tanto como GPUs de gama alta.

Sin embargo, existen alternativas para realizar inferencias en CPUs. El kit de herramientas OpenVino, permite ejecutar modelos de RNC en procesadores Intel, siendo posible conseguir buenos resultados con CPUS como Intel core i5 o i7, sin tener que depender de GPUs.

## 2.4 ANÁLISIS DE DATASETS

Un dataset, en el contexto de las RNC, es un conjunto grande de imágenes, acompañadas de anotaciones que indican qué objetos aparecen en la imagen, y dónde se encuentran. Son una parte esencial del desarrollo de un sistema basado en RNC, porque cuantas más imágenes estén disponibles a la hora de entrenar una red, mayor tiende a ser su precisión final.

Además, un algoritmo de detección basado en RNC es inútil sin un conjunto de datos con los que entrenar un modelo. Por eso, en este apartado se presentan los datasets que se han utilizado en este trabajo.

### 2.4.1 *Common Objects in Context*

El dataset COCO es el más extendido en el mundo de las redes neuronales convolucionales [16]. Todos los desarrolladores de nuevos algoritmos lo usan para evaluar su rendimiento, siendo el dataset de referencia, utilizado para comparar el rendimiento de distintos modelos.

Cuenta con imágenes de 80 clases distintas, que se aprecian en la Figura 2.2.

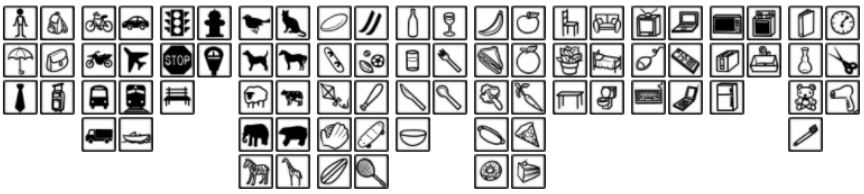


Figura 2.2: Clases representadas en el dataset COCO.

### 2.4.2 *Open Images Dataset V6*

Construido por Google, es el dataset con anotaciones de localización más grande que existe [5]. Cuenta con casi 2 millones de imágenes, en las que aparecen 600 clases distintas.

### 2.4.3 *Virat*

El dataset de Virat [17] está construido con imágenes de videovigilancia. Contiene una gran variedad de escenas y situaciones obtenidas de sistemas de de vigilancia reales, centrándose principalmente en personas, coches y otro tipo de vehículos, como se aprecia en la Figura 2.3

(a) (b)

Figura 2.3: Ejemplos de imágenes en Virat

Este es el dataset ideal para entrenar un modelo que vaya a utilizarse en un sistema de videovigilancia, ya que cuanto más parecidas sean las imágenes del entrenamiento a las imágenes finales, mejores resultados se obtendrán del modelo.

## 2.5 PIPELINE DE BUAVI

La plataforma de seguridad de la empresa Buavi, SharpView [18], realiza los siguientes pasos para la monitorización. Se obtiene el vídeo de las cámaras de seguridad, y se descodifican los fotogramas del vídeo para ser procesados. A continuación se realiza un pre-análisis

muy básico, para detectar cambios en la imagen (esto es rápido). Si se detecta actividad, se procesan con más detalle las imágenes, clasificando los objetos (este es el módulo más costoso). Por último, si se detecta un tipo de objeto no deseado se generan alarmas. Estos pasos están resumidos en la Figura 2.4.

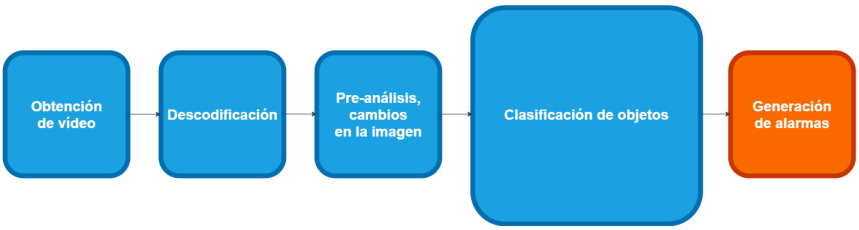



Figura 2.4: Pipeline de videovigilancia de la plataforma SharpView.

Este trabajo centra sus esfuerzos en optimizar el cuarto módulo, el de detección y clasificación de objetos utilizando redes neuronales convolucionales.

# 3

# METODOLOGÍA

## 3.1 MÉTRICAS DE INTERÉS

Para comprender las comparativas entre los distintos modelos, se van a explicar brevemente tres métricas importantes.

### 3.1.1 *Intersection Over Union*

Intersection Over Union, o IoU, es una métrica que indica qué porcentaje de área comparten dos rectángulos. Para entender la importancia de esta métrica, es necesario conocer la manera en la que una RNC como YOLO localiza y clasifica un objeto.

Al introducir una imagen en el modelo, YOLO realiza una serie de operaciones, y para cada objeto que encuentra en la imagen, extrae dos coordenadas y la clase a la que pertenece. Estas dos coordenadas son el extremo superior izquierdo y el inferior derecho de una 'caja' (*bounding box* en inglés). Las *bounding boxes* son rectángulos imaginarios que sirven para delimitar dónde se encuentra un objeto en la imagen. Si esta caja se superpone a la imagen, indica dónde se encuentra el objeto, tal y como muestra la Figura 3.1.

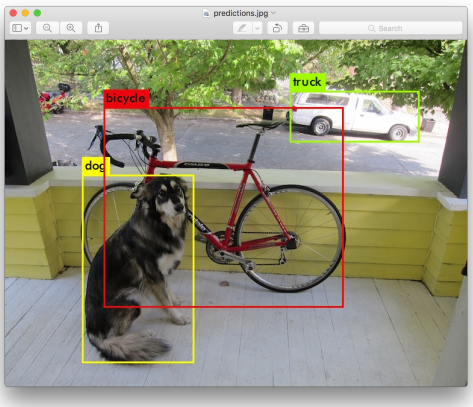



Figura 3.1: Generación de *bounding boxes* correspondientes a la localización de los objetos en la imagen.

La medida IoU coge esa caja, y mide cuánto se parece a la caja que contiene al objeto, como muestra la Figura 3.2. Es decir, mide el

porcentaje de precisión de la red neuronal a la hora de decidir dónde está un determinado objeto en la imagen.

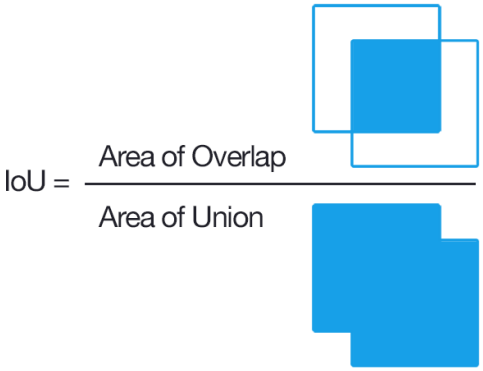



Figura 3.2: Cálculo de Intersection over Union dadas dos cajas; la correcta y la detectada [19]

En un sistema de detección, se suele considerar que una detección es correcta si su IoU es superior a un valor fijo. Típicamente se utiliza IoU = 0.5, es decir, la detección es correcta si las cajas comparten al menos la mitad del área.

### 3.1.2 *Average Precision*

Definimos la AP (precisión media) como una función de los verdaderos positivos, falsos positivos y falsos negativos en las detecciones. Corresponde a la integral de la curva precisión-recall, aunque en este contexto se puede entender como la precisión de un sistema de detección.

### 3.1.3 *Mean Average Precision*

Mean Average Precision, o mAP, es una medida muy utilizada para evaluar la precisión de un modelo basado en RNC.

La mAP se define como la media de las AP para distintos valores de IoU. Por ejemplo, se realiza la medida AP considerando como correcta una predicción si su IoU es 0.5 o más. Después, se realiza la medida AP pero con un IoU de 0.55 ... así hasta 0.9. Finalmente se hace la media de estas medidas, y se obtiene la mAP.

## 3.2 HARDWARE

El sistema principal utilizado para entrenar e inferir durante todo este trabajo ha sido un computador del grupo de Arquitectura de Computadores denominado `socarrat.unizar.es`. Cuenta con un procesador Intel Core i7 de 6ª generación y 4 GHz, y dispone también de una gráfica Nvidia Geforce GTX Titan X de 12 GB de memoria dedicada.

Para monitorizar el uso de la CPU, se han utilizado las herramientas *perf htop*, y para el uso de la GPU, la herramienta para gráficas de NVIDIA *nvidia-smi*.

## 3.3 SOFTWARE

Socarrat tiene el sistema operativo CentOS 7, con la versión de kernel 3.10.0-693. Todas las pruebas de este trabajo se han realizado en este sistema, para obtener las comparativas más objetivas posibles.

Para realizar comparativas de precisión entre modelos distintos, las imágenes que se infieren son exactamente las mismas, con la misma resolución. Las pruebas se han ejecutado entre 5 y 10 veces por modelo, para detectar inconsistencias producidas por algún otro proceso que esté ejecutándose. Además, las imágenes utilizadas para entrenar los modelos son distintas a las utilizadas para evaluar la precisión.

Se ha monitorizado en todo momento el uso de CPU y GPU, y comprobado que su uso fuese el esperado. Los repositorios utilizados para realizar las pruebas de Darknet, Tensorflow y OpenVino son, respectivamente, darknet [20], tensorflow-yolov4-tflite [21] y OpenVINO-YOLOV4 [22]. Las versiones utilizadas han sido Tensorflow 2.3.0 y OpenVino 2020R4.

# 4

# SOLUCIÓN PROPUESTA

Al comienzo de este proyecto, se decidió utilizar el modelo YOLOv3 (versión 3 de YOLO) [23], pero ya que fue publicada una mejora del mismo, YOLOv4, se cambió al nuevo modelo.

YOLOv4, al igual que sus versiones anteriores, engloba al modelo principal (yolov4) y a distintas variantes de la red convolucional. Una de estas variantes es la versión 'tiny' del modelo (yolov4-tiny), que es una versión muy reducida en número de capas convolucionales. Por tanto, permite una inferencia mucho más rápida, pero con menor precisión.

Este trabajo persigue un modelo ligero y preciso, capaz de clasificar objetos ejecutándolo con el menor consumo posible, idealmente una CPU. En este capítulo se evalúan los modelos yolov4 estándar y yolov4 'tiny', realizando comparativas de rapidez y de precisión. Se compara la precisión de modelos entrenados con distintos parámetros, y cómo el número de clases afecta a la rapidez. Por último, se evalúan distintos frameworks sobre los que ejecutar el modelo óptimo.

El modelo resultante es yolov4 tiny, entrenado para reconocer una sola clase con una resolución de 512x512. Este modelo, ejecutado en Tensorflow y OpenVino, es el ganador de las pruebas y un modelo que puede ejecutarse sin ningún problema en una CPU estándar, soportando la detección de objetos con rendimiento de hasta 24 fotogramas por segundo.

También realizaron pruebas de detección utilizando CenterNet [11], pero se decidió descartar este modelo por estar desarrollado sobre Tensorflow, y estar solo disponible para GPU y no en CPU como se buscaba.

## 4.1 MODELOS COMPARADOS

Para no tener que entrenar un modelo entero antes de realizar pruebas, los autores de Darknet ponen disposición modelos pre-entrenados para reconocer objetos. Estos modelos están almacenados en ficheros de "pesos" que representan los distintos parámetros de la red neuronal, y son los valores que la red ajusta durante su entrenamiento.

Ofrecen un modelo para cada versión de YOLO, por tanto contamos con el modelo por defecto de yolov4, y además contamos con yolov4-tiny. Estos se han entrenado con el dataset de COCO, por lo que reconocen 80 clases distintas.

Estos modelos podrían utilizarse en el sistema tal y como vienen por defecto, pero lo ideal es re-entrenarlo con imágenes de videovigilancia, ya que se ajustan más a las que se van a utilizar en el sistema (Figura 4.1). El riesgo que se corre es over-fitting, es decir, entrenar demasiado a la red hasta el punto en el que solo reconozca bien las imágenes con las que se le ha entrenado.

(a) Dataset COCO [16]

(b) Dataset Virat

Figura 4.1: Cuanto más se parezcan las imágenes de entrenamiento a las imágenes del sistema de videovigilancia, mejor funcionará, pero también se corre el riesgo de sobre-entrenar.

Además, para reducir el tiempo de ejecución de la red, es interesante no utilizar un modelo que detecte más clases que las que queremos monitorizar. En lugar de detectar 80 clases distintas, como hace el modelo por defecto de YOLOv4, nos interesa detectar solamente una: la clase 'persona'.

Al reducir el número de clases de un modelo, el tiempo de inferencia se reduce. Esto es porque el número de parámetros en ciertas capas de la red es proporcional al número de clases, y al reducir el número de clases se reducen las operaciones realizadas. Por ejemplo, en el modelo estándar de YOLOv4, para 80 clases se utilizan 255 filtros, lo que genera un total de 456960 parámetros, o 'pesos', que la red tiene que calcular. Si reducimos el número de clases a 1, se utilizan 18 filtros, lo que genera 32256 parámetros. Es decir, se reducen los parámetros 14× y se requieren menos operaciones.

En este trabajo se comparan modelos de 80 clases con modelos de 1 clase, para ver en qué medida afecta esta reducción de clases al resultado final.

## 4.2 COMPARATIVA DE PRECISIÓN

### 4.2.1 *Versión del Modelo*

A la hora de comparar la precisión de dos modelos, se ha decidido mostrar la curva AP - IoU, ya que indica de forma intuitiva la precisión que se obtiene de cada modelo para valores de IoU bajos y altos. Esto es, para detecciones menos o más exigentes. En la Figura 4.2 se aprecia la diferencia de precisión entre el modelo estándar y el tiny.

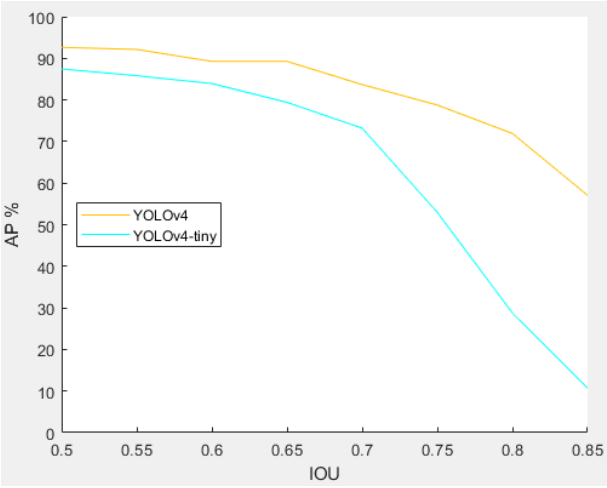


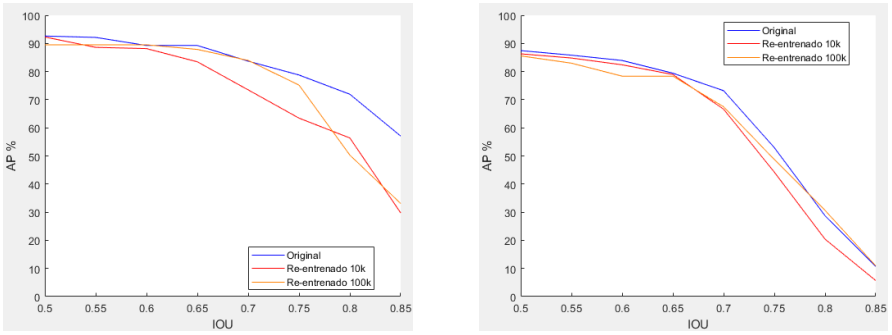

Figura 4.2: Curva AP-IoU del modelo estándar y tiny de YOLOv4

La precisión de yolov4-tiny, si bien es menor que la de yolov4, puede ser suficiente para un sistema de detección, especialmente teniendo en cuenta que el tiempo de inferencia será menor. Además, esta precisión corresponde a los modelos por defecto. ¿Se podría obtener una precisión superior reentrenando un modelo con nuevas imágenes?

### 4.2.2 *Cantidad de Imágenes en el Entrenamiento*

Partiendo de los pesos originales, se ha reentrenado la red yolov4 dos veces; una con 10.000 imágenes, y otra con 100.000, para comparar cómo la cantidad de imágenes afecta al entrenamiento de este modelo en concreto.

Al hecho de empezar el entrenamiento con los pesos de un modelo ya entrenado se le conoce como *fine-tuning*. Si se hubiera partido de un fichero de pesos aleatorio, el entrenamiento habría tardado mucho más en alcanzar un estado aceptable, ya que los parámetros habrían tenido que variar mucho más.




(a) Modelo estándar (b) Modelo tiny

Figura 4.3: Comparativa de precisión entre los modelos por defecto y los re-entrenados con 10.000 y 100.000 imágenes

Como se aprecia en la Figura 4.3, incluso al aumentar el número de imágenes del entrenamiento, no parecemos ser capaces de llegar a la precisión del modelo por defecto. Vamos a intentar cambiar algunos parámetros del entrenamiento y comparar la precisión que se obtiene.

*(Las siguientes pruebas se realizan con yolov4-tiny, ya que entrenar el modelo estándar tarda más de un día, y el modelo tiny termina en unas pocas horas.)*

### 4.2.3 *Resolución de Entrenamiento*

Tras investigar acerca de los parámetros del entrenamiento que podrían mejorar la precisión del modelo, se decidió entrenar el modelo yolov4-tiny con más resolución. En el modelo por defecto, las imágenes de entrada se redimensionan a 418x418 píxeles, antes de empezar a realizar los cálculos.

Por tanto, se decidió probar a entrenar el modelo con una resolución superior, 512x512 píxeles. En teoría, esto debería permitir al modelo detectar objetos más pequeños, ya que la imagen tiene más resolución.

Como se aprecia en la Figura 4.4, se ha conseguido una precisión superior re-entrenando la red con más resolución. Utilizar una resolución mayor afecta ligeramente al tiempo de inferencia, pero el cambio es lo suficientemente pequeño como para que no importe sacrificar un poco de velocidad, obteniendo a cambio más precisión.

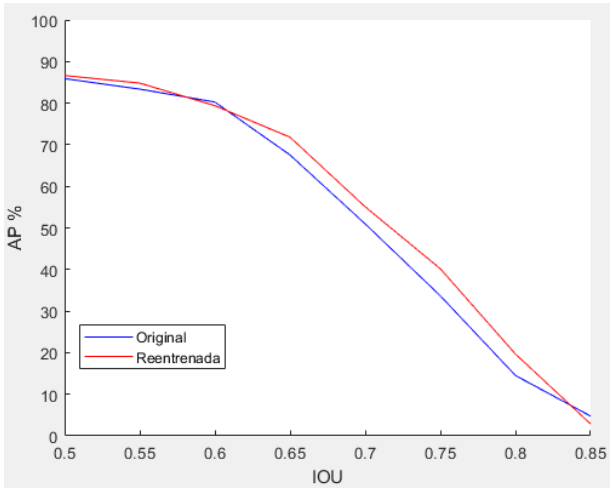



Figura 4.4: Comparativa de precisión entre el modelo original y el re-entrenado con mayor resolución

### 4.2.4 *Número de Clases*

Además, se decidió entrenar con los mismos parámetros un modelo tiny de una sola clase, en vez de 80, para comprobar si el número de clases afecta a la precisión final con el mismo entrenamiento.

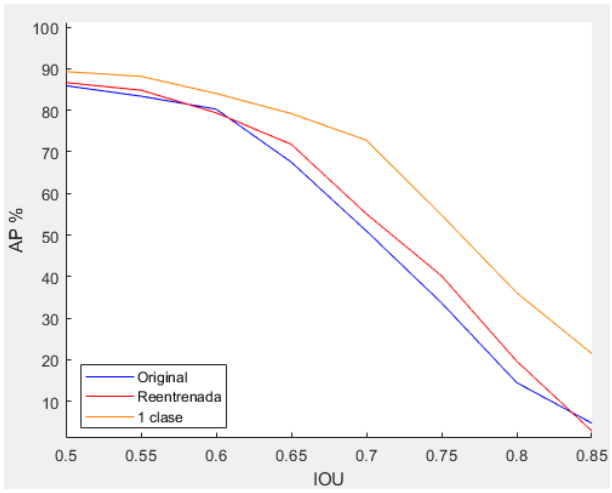



Figura 4.5: Comparativa de precisión entre el modelo original, el re-entrenado con 80 clases y el re-entrenado con una sola clase

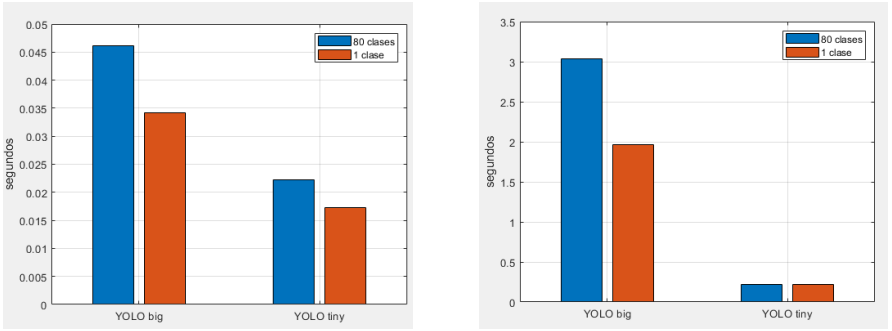
Disminuir el número de clases no siempre conlleva un aumento en la precisión, sino que depende del problema específico y del modelo. En nuestro caso, como se ve en la Figura 4.5, la precisión ha aumentado, de modo que hemos conseguido un modelo más preciso y más rápido que el de por defecto.

La precisión que se obtiene con el modelo tiny, si bien no es tan buena como la del modelo estándar, es suficiente para un entorno de

videovigilancia. Para darnos cuenta de cuanta velocidad se obtiene sacrificando un poco de precisión, vamos a realizar la comparativa de rapidez de los modelos.

## 4.3 COMPARATIVA DE TIEMPO DE EJECUCIÓN

Teniendo en cuenta que modelos con distinta cantidad de clases no tardan lo mismo en inferir imágenes, se decidió comparar la rapidez de yolov4 y la de yolov4-tiny en el framework Darknet, con modelos de 80 clases y de 1 clase para cada uno. Se ha medido el tiempo medio de inferencia por imagen para un conjunto de 26 imágenes escogidas al azar del dataset de COCO.




(a) GPU (b) CPU

Figura 4.6: Comparativa del tiempo de ejecución de los modelos estándar y tiny. En CPU la diferencia es mucho más grande.

| | Fotogramas por Segundo | | | |
|---|---|---|---|---|
| | GPU | | CPU | |
| clases | Standard | Tiny | Standard | Tiny |
| 80 | 21 | 44 | 0.32 | 4.5 |
| 1 | 29 | 57 | 0.5 | 4.6 |

Cuadro 4.1: Comparativa de fotogramas por segundo

Queda claro, tanto en la Figura 4.6 como en el Cuadro 4.1, que el modelo óptimo para este problema es la versión tiny de YOLOv4. Tiene una precisión suficiente para este sistema, y alcanza los 4.6 FPS ejecutándose en CPU con Darknet, lo que ya sería suficiente para la detección en una única cámara de personas u objetos que no se muevan a alta velocidad.

A continuación se van a analizar los distintos frameworks sobre los que se podría ejecutar el modelo para conseguir la mayor velocidad posible y detectar objetos en varias cámaras, o ejecutar la detección de objetos más rápidos en una cámara.

## 4.4 COMPARATIVA DE FRAMEWORKS

Los frameworks que se van a comparar son Darknet, Tensorflow y OpenVino. Se muestran pruebas de inferencia en CPU, ya que se ha comprobado en el apartado anterior que una CPU es suficiente para este problema. Además, no haría falta adquirir una GPU por cada sistema de videovigilancia instalado, lo que supone un ahorro importante para la empresa.

Existe una variante de Tensorflow optimizada para dispositivos móviles llamada Tensorflow Lite. Las pruebas de velocidad se han realizado también con este framework, para ver si se consigue un rendimiento superior.

Para cada framework, se ha realizado la inferencia de 200 imágenes con el modelo yolov4-tiny de 80 clases.

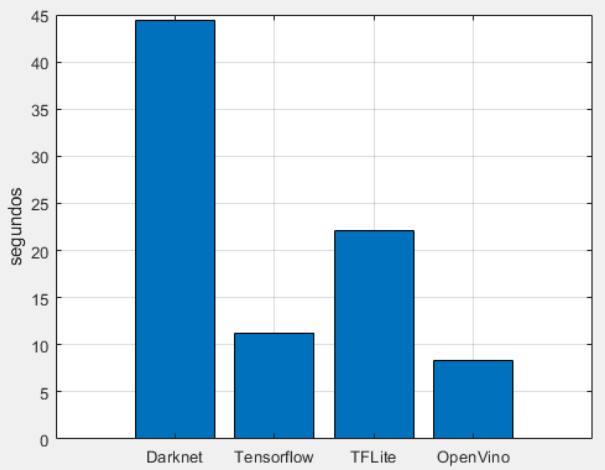



Figura 4.7: Comparativa de rapidez entre los distintos frameworks para la inferencia de 200 imágenes.

Como se observa en la comparativa de la Figura 4.7, Darknet está muy por debajo de los demás en velocidad, ya que se trata de un framework optimizado para GPUs. En la Figura 4.8 se muestra una comparativa ampliada de las demás opciones.

| Framework | FPS |
|---|---|
| Darknet | 4.5 |
| Tensorflow | 17.77 |
| TFLite | 9.06 |
| OpenVino | 24.09 |

Cuadro 4.2: Comparativa de fotogramas por segundo para yolov4-tiny en distintos frameworks.

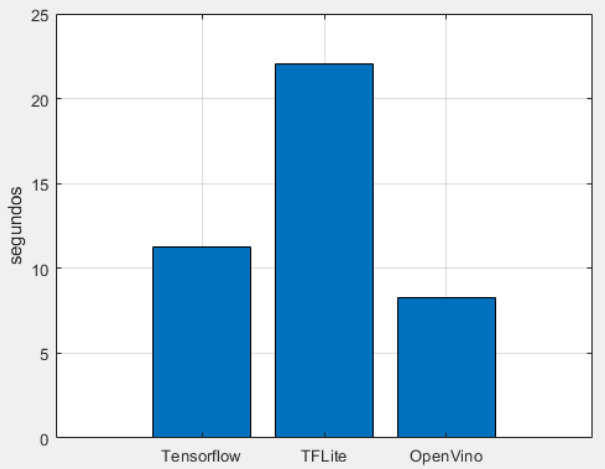



Figura 4.8: Comparativa de rapidez entre los distintos frameworks para la inferencia de 200 imágenes.

Tensorflow Lite ha salido perdiendo, tardando casi el doble que el modelo en Tensorflow. La razón es que TFLite está optimizado para procesadores ARM (con el set de instrucciones vectoriales NEON). Al ejecutarlo sobre una CPU de Intel, no se utilizan las instrucciones vectoriales de la CPU y el rendimiento es peor.

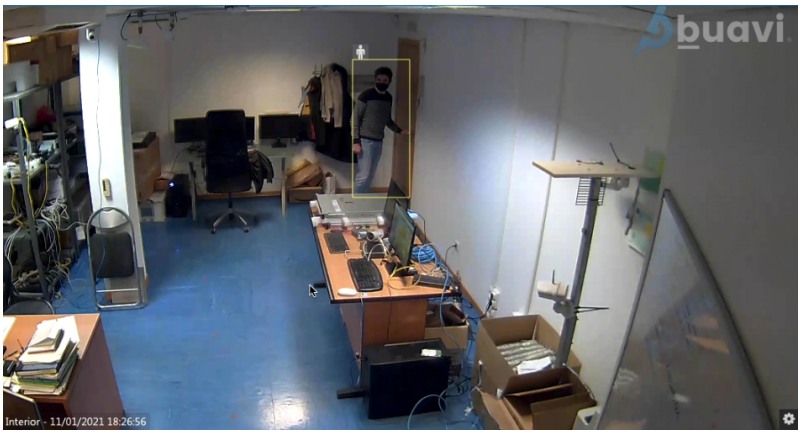
De hecho, esta es la razón de que el ganador en velocidad sea OpenVino; este kit de herramientas aprovecha lo más posible la arquitectura del procesador de Intel, siendo la opción más optimizada si se quiere ejecutar el modelo en CPUs de esta marca.

Si, por el contrario, no se quiere depender de CPUs de Intel por el precio o las opciones que ofrecen otras marcas, Tensorflow es claramente el framework a utilizar para este problema, ya que funciona muy bien en otros dispositivos como GPUs o aceleradores discretos.

En definitiva, el mejor modelo observado para este problema es yolov4 tiny, entrenado para reconocer una sola clase con una resolución de 512x512. Se puede ejecutar en OpenVino para procesadores Intel, y en Tensorflow para otras CPUs. Consigue detección en tiempo real en varias cámaras, y si se quieren detectar objetos más rápidos, es capaz de procesar 24 fotogramas por segundo. Obtiene precisiones muy elevadas para una red de su tamaño.

# 5 INTEGRACIÓN

Como paso final, la solución se ha integrado en SharpView, la plataforma de videovigilancia de Buavi. El nuevo modelo consigue una mayor precisión que el modelo anterior utilizado por la empresa manteniendo el rendimiento. Además, se han explorado alternativas casi igual de rápidas para dispositivos de computo que no sean de Intel, ya que OpenVino soporta unos pocos dispositivos de esta compañia https://docs.openvinotoolkit.org/latest/openvino_docs_IE_DG_supported_plugins_Supported_Devices.html . En la Figura 5.1 se muestran capturas del sistema de detección actuando a tiempo real.




(a) Interior


(b) Exterior con infrarrojos

Figura 5.1: Imágenes de detección usando el modelo final en el sistema de Buavi.

# 6

# CONCLUSIONES

El objetivo del trabajo era conseguir optimizar el módulo de detección y clasificación de objetos de la plataforma SharpView, haciendo que se pueda ejecutar en CPUs para consumir la menor cantidad de energía posible.

La solución propuesta ofrece a la empresa una mayor optimización en el módulo de detección y clasificación de objetos de su sistema de videovigilancia. Se han explorado también alternativas para no tener que depender de procesadores de Intel, de modo que se ha encontrado una solución versátil y rápida, independientemente del hardware utilizado.

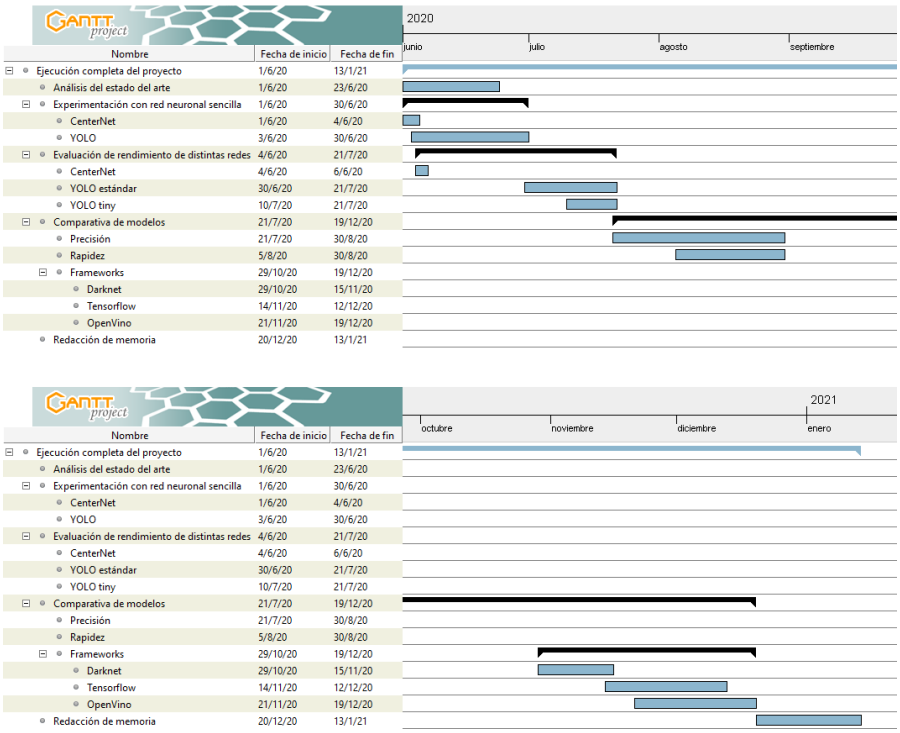
Desde hace unos años, los procesadores Intel y AMD cuentan con extensiones vectoriales avanzadas (AVX [7]). Gracias a ello, hoy en día la potencia de una CPU estándar es suficiente para obtener detección en una cámara en tiempo real. Aún así, si se necesitan monitorizar más cámaras, se puede utilizar una GPU de bajo coste, obteniendo resultados aún mejores debido al aumento de potencia computacional. El modelo es lo suficientemente pequeño para que, utilizando una GPU, se consigan monitorizar varias cámaras a la vez.

Como trabajo futuro, se plantea la posibilidad de emplear el dataset de Virat para ver si el modelo resultante se ajustaría más a las imágenes de videovigilancia y respondería a todas las necesidades de la plataforma de Buavi.

# ANEXO

## A.1 DIAGRAMA DE GANTT

A continuación se muestra el diagrama de Gantt, Figura A.1 y el cuadro A.1 con la distribución de horas por tarea. El bloque más importante ha sido el estudio de las técnicas y búsqueda de información relevante al que se le ha dedicado prácticamente un cuarto del TFG.




| Nombre | Fecha de inicio | Fecha de fin |
|---|---|---|
| Ejecución completa del proyecto | 1/6/20 | 13/1/21 |
| Análisis del estado del arte | 1/6/20 | 23/6/20 |
| Experimentación con red neuronal sencilla | 1/6/20 | 30/6/20 |
| CenterNet | 1/6/20 | 4/6/20 |
| YOLO | 3/6/20 | 30/6/20 |
| Evaluación de rendimiento de distintas redes | 4/6/20 | 21/7/20 |
| CenterNet | 4/6/20 | 6/6/20 |
| YOLO estándar | 30/6/20 | 21/7/20 |
| YOLO tiny | 10/7/20 | 21/7/20 |
| Comparativa de modelos | 21/7/20 | 19/12/20 |
| Precisión | 21/7/20 | 30/8/20 |
| Rapidez | 5/8/20 | 30/8/20 |
| Frameworks | 29/10/20 | 19/12/20 |
| Darknet | 29/10/20 | 15/11/20 |
| Tensorflow | 14/11/20 | 12/12/20 |
| OpenVino | 21/11/20 | 19/12/20 |
| Redacción de memoria | 20/12/20 | 13/1/21 |

Figura A.1: Diagrama de Gantt del proyecto.

## A.2 RESULTADOS DE CENTERNET

Se realizaron pequeñas pruebas con CenterNet, antes de descartarlo debido a la imposibilidad de ejecutarlo sobre CPU, y a la escasa documentación disponible, especialmente comparado a otras alternativas.

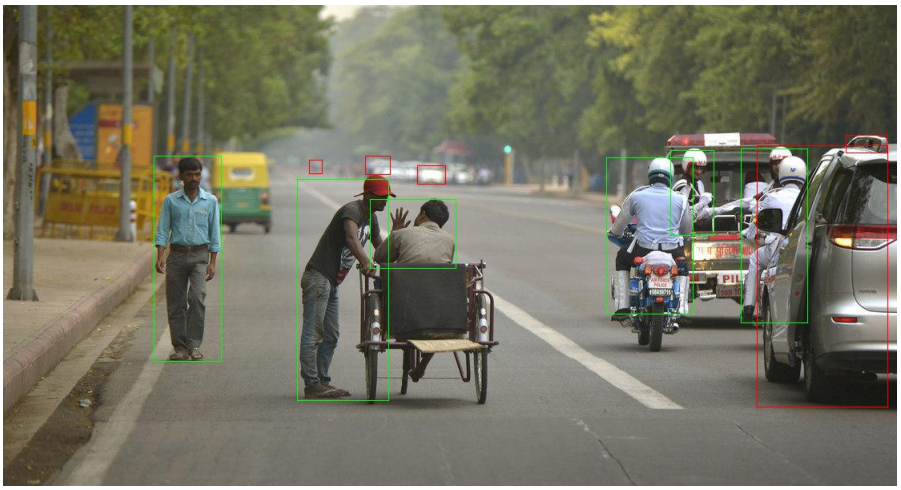
| | Horas |
|---|---|
| Análisis del estado del arte e investigación | 90 |
| Experimentación y rendimiento de CenterNet | 15 |
| Experimentación con YOLO | 40 |
| Rendimiento de YOLO estándar | 40 |
| Rendimiento de YOLO tiny | 20 |
| Comparativas de precisión | 20 |
| Comparativas de rapidez | 15 |
| Comparativas de distintos frameworks | 60 |
| Redacción de bitácora | 30 |
| Redacción de memoria | 30 |
| Reuniones | 30 |
| Total | 390 |

Cuadro A.1: Horas dedicadas al proyecto

Centernet es un modelo de red neuronal que detecta, además de la caja de predicción, un punto extra, que se corresponde con el centro de dicha caja. Este punto se detecta por separado (la predicción de la caja no influencia la predicción del centro). La idea detrás de este cálculo extra es descartar resultados potencialmente erróneos; si el centro detectado de un objeto en particular cae en el centro de una caja detectada del mismo objeto, se acepta la predicción de dicho objeto. Si se aleja demasiado del centro de la caja predicha, se descarta.

Los resultados obtenidos con CenterNet fueron muy buenos, a la altura de YOLOv4. Obtuvo una media de fotogramas por segundo de 24, frente los 21 de YOLOv4. Sin embargo, no cuenta con una versión pequeña del modelo, lo que nos ha sido muy útil en este trabajo. Las figuras A.2 y A.3 muestran las predicciones para imágenes del dataset de Virat y el de COCO, respectivamente.

Figura A.2: Predicciones de CenterNet para una imagen del dataset de Virat, las personas en verde y los coches en rojo.



(a)

(b)

Figura A.3: Predicciones de CenterNet para imágenes del dataset de COCO, las personas en verde y los coches en rojo.

# BIBLIOGRAFÍA


[1] *Speeding up Convolutional Neural Networks*. URL: https://towardsdatascience.com/speeding-up-convolutional-neural-networks-240beac5e30f.

[2] *YOLO: Real-Time Object Detection*. URL: https://pjreddie.com/darknet/yolo/.

[3] *Rich feature hierarchies for accurate object detection and semantic segmentation*. URL: https://arxiv.org/abs/1311.2524.

[4] *Coral USB Accelerator*. URL: https://coral.ai/products/accelerator.

[5] *Open Images Dataset*. URL: https://opensource.google/projects/open-images-dataset.

[6] *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[7] *Extensiones Vectoriales Avanzadas*. URL: https://es.wikipedia.org/wiki/Extensiones_Vectoriales_Avanzadas.

[8] *YOLOv4: Optimal Speed and Accuracy of Object Detection*. URL: https://arxiv.org/abs/2004.10934.

[9] *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. URL: https://arxiv.org/abs/1506.01497.

[10] *Keras API*. URL: https://keras.io/.

[11] *CenterNet: Keypoint Triplets for Object Detection*. URL: https://arxiv.org/abs/1904.08189.

[12] *Darknet: Open Source Neural Networks in C*. URL: https://pjreddie.com/darknet/.

[13] *An end-to-end open source machine learning platform*. URL: https://www.tensorflow.org/.

[14] *OpenVINO™ Toolkit Overview*. URL: https://docs.openvinotoolkit.org/latest/index.html.

[15] *Why GPUs are more suited for Deep Learning?* URL: https://www.analyticsvidhya.com/blog/2020/09/why-gpus-are-more-suited-for-deep-learning/.

[16] *COCO Dataset Explorer*. URL: https://cocodataset.org/#explore.

[17] *The VIRAT Video Dataset*. URL: https://viratdata.org/.

[18] *Plataforma de seguridad SharpView*. URL: https://buavi.com/es/sharpview/.

[19] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. URL: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[20] Alexey Bochkovskiy. *Darknet (YOLOv4)*. URL: https://github.com/AlexeyAB/darknet.

[21] hunglc007. *Tensorflow YOLOv4 TFLite*. URL: https://github.com/hunglc007/tensorflow-yolov4-tflite.

[22] Tianwen Wu. *OpenVino YOLOv4*. URL: https://github.com/TNTWEN/OpenVINO-YOLOV4.

[23] *YOLOv3: An Incremental Improvement*. URL: https://arxiv.org/abs/1804.02767.