Deep Learning: Neural Networks for Object Detection

Alejandro Garcia Miñano

Resumen– En la actualidad se está llevando a cabo unan gran mejoría en los sistemas basados en detección de objetos, con el incremento de potencia de las GPUS, cada vez es más viable realizar entrenamientos en conjuntos de datos lo suficientemente grandes como para lograr una buena precisión, los detectores de una fase se están imponiendo ante los detectores de dos fases por su principal ventaja de obtener tiempos de inferencia mucho menores, esto abre una gran cantidad de usos para este tipo de detectores como pueden ser sistemas de vigilancia en tiempo real, control del tráfico,etc. En este articulo se analiza el detector de una fase conocido como RetinaNet, que utiliza las técnicas ya conocidas por los detectores de una fase pero que introduce como novedad una función de error capaz de incrementar la precisión obtenida hasta este momento con los detectores de una fase, solventando el mayor problema que comparten todos ellos, que es el desbalanceo en la detección de objetos.

Palabras clave– RetinaNet,Red neuronal convolucional,Deteccion de objetos, Detector de una fase, keras, cuadro delimitador, FPN, resnet, tensorflow, caja de anclaje,IoU

Abstract– Currently, object detection systems have been improved a lot, with the increasing power of GPUS, it is becoming more viable to train in a big enough datasets in order to reach a good precission, one-stage-detector are imposing against two-stage-detector due to their main advantage of taking much less inference time, this opens a lot of uses for that kind of detectors such as real-time surveillance systems,traffic control and so on. In this article we analyze RetinaNet, wich is a one-stage-detector that uses well-known techniques by one-stage-detectors but it also introduces a new loss function that allows the network to increase the precision obtained until that moment by the one-stage-detectors, solving the biggest problems that share all of them, which is the class unbalance in object detection.

Keywords– RetinaNet,CNN,Object detection, one-stage-detector, keras, bounding box, FPN, resnet, tensorflow, anchor boxes,IoU

1 Introducción

La detección de objetos es uno de los ámbitos más populares en el ámbito de la visión por computador, esto es así ya que tiene numerosas utilidades como por ejemplo detección de vehiculos para el control del tráfico, detección de cualquier tipo de obstáculo para tareas de conducción autónoma,etc.
Se define la detección de objetos como una clasificación y además una localización, además, se necesitan modelos capaces de detectar varios objetos dentro de una sola imagen, incluso objetos que se encuentran en la misma posición de la imagen solapándose entre sí, la forma habitual de mostrar esta información es mediante los conocidos como **bounding boxes** [5], en la Figura 1 se puede observar el tipo de resultado esperado en una red capaz de detectar objetos.

Por tanto, es un problema con una complejidad añadida en comparación con la clasificación de imágenes, existen diversos métodos para detectar objetos en imágenes, básicamente se pueden clasificar en dos tipos, los detectores de una fase y los detectores de dos fases, los primeros son detectores enfocados a un uso en tiempo real, pues al realizar todo el proceso de detección en una única fase, el tiempo a la hora de realizar inferencia suele ser mucho menor, en cambio, los detectores de dos fases, antes de proporcionar la imagen a la red neuronal encargada de realizar la detección

• E-mail de contacto: alejandro.garciami@e-campus.uab.cat
• Mención realitzada: Computación
• Trabajo tutorizado por: Fernando Vilariño Freire (Ciencias de la computación)
• Curso 2017/18

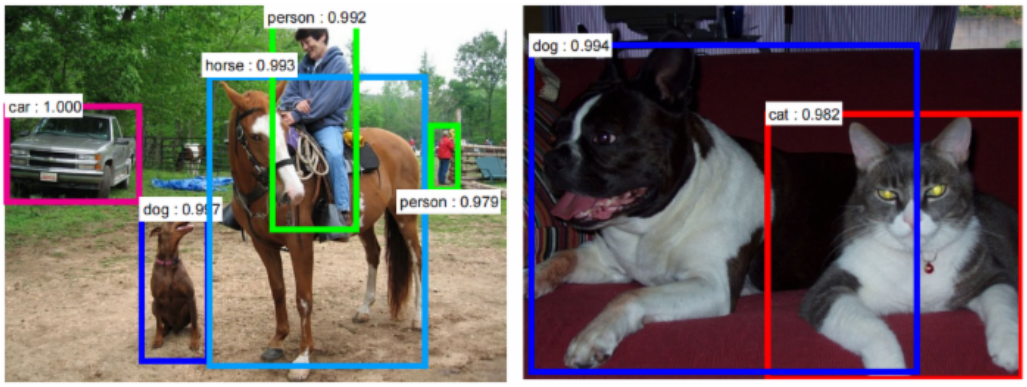


Fig. 1: Resultado de un detector de objetos

de objetos, se pasa por otra red neuronal que se encarga de obtener las regiones más interesantes de la imagen.

Para realizar el estudio de los detectores de una fase, el articulo se ha dividido en diversas secciones, en las primeras secciones se va a detallar los objetivos del estudio, estado del arte, y las metodologías empleadas, en las siguientes secciones se pasará a analizar algunas de las herramientas básicas que utilizan los detectores de objetos como pueden ser las FPN y los anchor boxes. Para finalizar, se presentará RetinaNet [11], y se analizarán los resultados obtenidos.

2 Objetivos

Los objetivos definidos y sus respectivos pesos, es decir, la importancia que tienen para este proyecto son los siguientes:

1. Instalación y conFiguración de la librería Keras sobre Tensorflow con el soporte para GPUS (20 %)

2. Análisis y conFiguración de hiperparámetros con utilizando redes neuronales convolucionales soportadas por keras (20 %)

3. ConFiguración, uso y análisis de una red neuronal capaz de detectar objetos (RetinaNet) (50 %)

4. Pequeña aplicación que haga uso de la red neuronal entrenada (10 %)

3 Estado del arte

Para la evaluación del estado del arte, se van a comparar los diferentes detectores, de 1 y 2 fases, todos ellos utilizando la métrica estándar de MS-COCO [15], que tiene en cuenta diferentes tamaños de imagen así como diferentes IoU [5], se va analizar el estado del arte con diferentes redes, comparando especialmente los detectores de una fase, ya que son el principal foco de atención de este articulo, como detectores de dos fases se compara el método Faster R-CNN [25] con diversas variantes, en la Figura 2 se puede ver el esquema de funcionamiento de un detector de dos fases..

Como detectores de una fase, principales competidores de RetinaNet [11], encontramos los siguientes detectores:

YOLO: YOLO [13] es un detector de una fase cuya principal característica reside en la velocidad, no agrega estructuras complejas por tal de mejorar al máximo posible la velocidad de inferencia, su arquitectura se basa en una darknet [26] para la mayor velocidad posible, la version 3 de YOLO, aumenta la efectividad del detector en objetos pequeños.

SSD: SSD [24] es un detector de una fase con una gran velocidad de inferencia, no tan rápido como YOLO [13], como arquitectura principal cuenta con una FPN [1] y un backbone que se puede escoger, aunque en sus primeras versiones utilizaba una VGG19 [8] como backbone.

RetinaNet: RetinaNet [11], es la red escogida para analizar en este articulo, se caracteriza por ser un detector de una fase con tasas de AP incluso superiores a los detectores de dos fases que son más lentos realizando inferencia, utiliza técnicas como FPN y una novedosa función de error que le permite situarse como la mejor red en relación a velocidad y precisión que proporciona la red.

En la Figura 3 se pueden observar los resultados de las diferentes redes, siendo RetinaNet [11] la que mejores resultados ofrece, utilizando como backbone la CNN [22] resnext101 [23], la cual es una modificación de la popular resnet [7].

4 Metodología

Todo el trabajo se ha realizado con python utilizando la librería Keras [3] y Tensorflow [4], con la libreria Keras se pueden construir arquitecturas de redes neuronales complejas como es el caso de RetinaNet de forma intuitiva y comprensible.

Para la división del trabajo a realizar, se ha gestionado mediante metodologías ágiles, en este caso en concreto realizando sprints de 2 semanas de duración, definiendo las tareas de forma clara para realizar en casa sprint.

Para la presentación de resultados, se ha empleado tensorboard, que es una herramienta que forma parte de Tensorflow y permite escribir en un fichero todas las operaciones que se van realizando, para posteriormente poder visualizarlas a través de la aplicación web, no obstante, tensorboard no permite un análisis tan detallado como a veces se requiere, en este caso analizar el comportamiento de la red por cada tipo de objeto detectado, por este motivo, se recurre a herramientas externas como puede ser Microsoft Excel, o bien utilizando la librería pyplot de python.

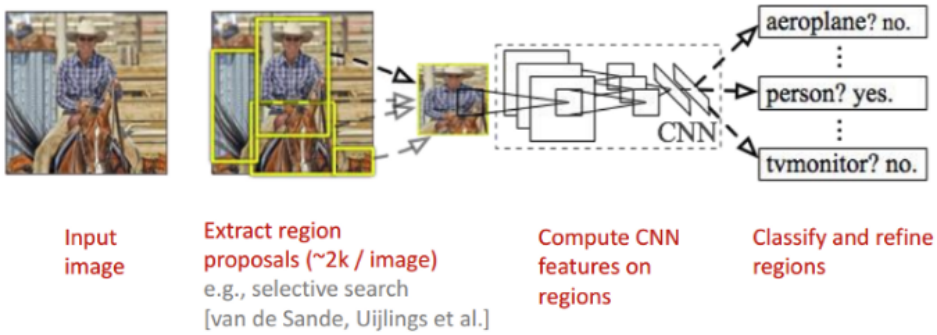


Fig. 2: Proceso de detección de objetos de un detector de dos fases

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

Fig. 3: Tabla comparativa del estado del arte en la detección de objetos, en ella se comparan varios detectores de una fase y otros de dos fases.

4.1. Métricas de evaluación

Para la clasificación de objetos, una etiqueta indica si el objeto clasificado es correcto o incorrecto, no obstante, para la detección de objetos no es tan sencillo, ya que además de indicar a que clase pertenece el objeto, también se debe indicar su posición mediante un bounding box, habitualmente se proporcionan las posiciones de los bounding boxes mediante coordenadas cartesianas (x1,y1,h2,w2), es decir, mediante la coordenada superior izquierda y además su altura y anchura,por tanto, una posible manera de evaluar como se ha clasificado el objeto seria comparar directamente estas posiciones, sin embargo, en la práctica, es muy improbable que coincidan la salida del bounding box de la CNN coincida exactamente con la del Ground truth, por tanto se utiliza una métrica conocida como IoU (intersect over union), la cual se define mediante la siguiente formulación:

$$IoU = \frac{\text{area intersección}}{\text{area unión}}$$

Dicha métrica proporciona una forma muy natural de determinar como de buena es una predicción, ya que es de la forma en que los humanos lo percibimos.

Tal como se puede observar en la Figura 4, a partir de una predicción cercana al 0,7 ya se considera una buena predicción y por tanto sería muy difícil para un humano distinguir si la predicción es del 0,9 o bien 0,7 sin que nos mostrasen el bounding box del Ground truth, generalmente se puede considerar correcta una predicción mayor al 0,5 de IoU, ya que acercarse a valores cercanos a 0,9 puede ser realmente difícil. La métrica explicada se utiliza en la mayoría de competiciones de detección de objetos, a

Fig. 4: Ejemplo de 3 predicciones y sus respectivos valores de bounding box

continuación se va a explicar como se evalúa la detección en la competición MS-COCO.

Average Precision (AP)

- **AP:** Métrica principal de la competición, para calcularla se realiza una media entre las predicciones realizadas con un IoU de al menos 0,5, valor que se ve incrementado en 0,05 hasta un máximo de 0,95
- **APIoU=,75:** AP para objetos detectados con un IoU de al menos 0,75
- **APsmall:** AP para objetos pequeños: área menor a 322.
- **APmedium:** AP para objetos medianos: área entre 322 y 962.
- **APlarge:** AP para objetos grandes: área mayor a 962.

La competición realiza esta distinción entre tamaños de objetos porque generalmente los objetos pequeños suelen ser más difíciles de detectar y así se proporciona un extra de información a los competidores.

5 ANCHOR BOXES

Para poder llevar a cabo la detección de objetos, las redes neuronales convolucionales realizan subsampling de la imagen hasta que se llega a la última capa.
Dentro de cada posición del mapa de características, pueden haber 0, 1 o más objetos.

Fig. 5: Figura con mapa de caracteristicas de 3×3

Observando la Figura 5, se puede ver que el centro de la persona y el centro del coche están situados en la misma partición, cada partición pasa a través de una red neuronal que realiza la clasificación del objeto y una red neuronal que realiza una regresión para obtener las coordinadas del bounding box.
Por tanto, se define la salida de la red neuronal de clasificación de la siguiente manera:

$$Y = \begin{bmatrix} Wi \\ Hi \\ K \end{bmatrix}$$

Donde Wi es el ancho del mapa de características, Hi es la altura del mapa de características, y K es el número de clases que se clasificarán. Mientras que la salida de la red neuronal de regresión es la siguiente:

$$Y = \begin{bmatrix} Wi \\ Hi \\ X1 \\ Y1 \\ H1 \\ W1 \end{bmatrix}$$

X1,Y1 se refieren a los puntos en coordenadas cartesianas donde se sitúa la esquina superior izquierda del bounding box, mientras que H1 y W1 hacen referencia a la altura y anchura que tendrá el bounding box
Con la salida proporcionada, lo que ocurriría si simultáneamente hay dos objetos en la misma partición de la imagen como ocurre en la Figura 5 es que no sería posible detectar ambos objetos, ya que la salida nos permite detectar un único objeto. Una posible solución sería incrementar la dimensionalidad de la salida, añadiendo N columnas en función de la cantidad de objetos que se quieran detectar, el problema de esta solución es que no se puede saber a que columna corresponde cada objeto, y por tanto ajustar la función de error correctamente, para solucionar esto, se utilizan los denominados **anchor boxes** [14], estos elementos son figuras de diferentes formas que pueden asemejarse a objetos del mundo real, a las que se asociaran los objetos de la imagen, en la Figura inferior se puede observar un ejemplo de 9 anchor boxes.

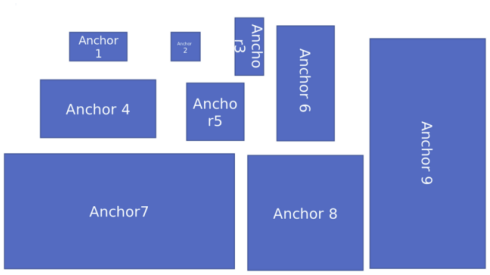


Fig. 6: 9 anchor boxes de diferentes tamaños y relaciones de aspecto, de forma que sean capaces de parecerse al máximo número posible de objetos reales

La forma y el tamaño de los anchor boxes se puede elegir a libre disposición, siempre teniendo en cuenta cubrir todas las posibles relaciones de aspecto diferentes y tamaños. Para el caso de la Figura 5, se asignaría el anchor box 9 al peatón y el anchor box 7 al coche,la asignación se realiza de la siguiente manera:
Si el objeto tiene al menos 0,7 IoU con el anchor box o bien es el anchor box con mayor IoU con el objeto.
Si el objeto tiene menos de 0,3 IoU con el anchor box, se descarta, en caso de que el objeto no supere el 0,3 IoU con ningún anchor box, el objeto no contribuirá al entrenamiento ya que no podrá ser detectado.

La salida de la red para el regresor utilizando anchor boxes será la siguiente:

$$y = \begin{pmatrix} A1 & A2 & \dots & A9 \\ Wi & Wi & \dots & Wi \\ Hi & Hi & \dots & Hi \\ X1 & X2 & \dots & X9 \\ Y1 & Y2 & \dots & Y9 \\ H1 & H2 & \dots & H9 \\ W1 & W2 & \dots & W9 \end{pmatrix}$$

Para cada región de la imagen, se pueden detectar 9 objetos de forma simultanea, en la práctica, es poco frecuente que más de dos objetos se sitúen en la misma región de la imagen.

6 FEATURE PYRAMID NETWORK

En algunas competiciones de clasificación, se discrimina por el tamaño de los objetos a detectar, como por ejemplo en MS-COCO [15], esto se debe a que detectar objetos de pequeño tamaño es más difícil que hacerlo con objetos más

grandes, y esta información adicional es interesante de cara a mejorar el funcionamiento del detector de objetos si el mismo está cometiendo muchos errores con objetos pequeños.

La arquitectura conocida como **feature pyramid network** [2], de aquí en adelante **FPN**, permite obtener a partir de una imagen, predicciones a distinta escala, una manera de realizar esto sería disponer de varias medidas de la misma imagen, por ejemplo 800×800, 600×600, 400×400, 256×256, reduciendo su tamaño y así obteniendo predicciones de distintos tamaños de todos los objetos de la imagen, el problema es que realizar esta operación tiene un coste computacional muy elevado, ya que se tendrían que proporcionar 4 imágenes distintas a la red neuronal, multiplicando el tiempo necesario para realizar la inferencia en 4.

La FPN permite realizar esta operación aprovechando la estructura natural de una CNN, ya que cuando se avanza a través de la red se va reduciendo el tamaño de la imagen de entrada a principalmente a través de las capas de max pooling, implícitamente, esto proporciona una estructura de pirámide y disponer de la misma imagen a diferentes escalas, básicamente en una CNN existen dos tipos de fases, la fase de convolución, donde generalmente el tamaño de la imagen de entrada no se ve afectado, y la fase de reducción, donde se aplica una capa max pooling para reducir el tamaño de la imagen, para conseguir una FPN, basta con seleccionar varias capas convolucionaeles conectarlas lateralmente con una capa adicional que se utilizará para hacer predicciones en ese nivel de la red, habitualmente la capa convolucional que se selecciona la previa a reducir el tamaño de la imagen, ya que es la que contiene mayor información de la misma.
No obstante, esta metodología no funciona correctamente, y es que las últimas capas de una CNN tienen una semántica diferente de las primeras capas y por tanto también hay que conectar las capas adicionales que se han creado para la estructura FPN, tal como se ilustra en la siguiente Figura:

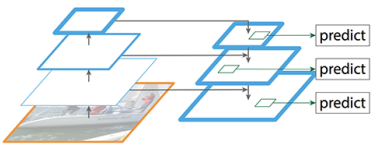


Fig. 7: Arquitectura de FPN, a la izquierda se muestra la CNN[22] y las 3 capas seleccionadas para interconectarse lateralmente con las capas de pirámide.

Como se puede ver en la Figura 7, las capas de la pirámide están simultáneamente a la vez con las capas de la CNN y con las propias capas de pirámide, el motivo de interconectar las capas de pirámide entre si es debido a las diferencias en resolución y semántica dentro de las capas de la CNN, de esta manera se consigue que todas las capas de pirámide tengan una semántica similar.

6.1. Conexión de las capas de pirámide

Para lograr una semántica similar en todas las capas de la FPN, se sigue la arquitectura que se muestra a continuación:

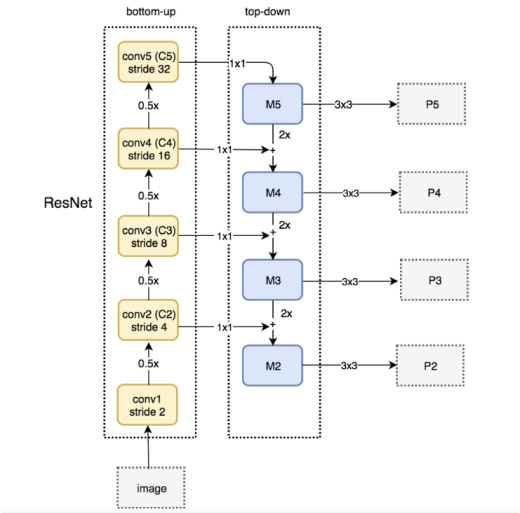


Fig. 8: Arquitectura completa de FPN, donde se muestran las conexiones entre las capas de pirámide y la CNN, en este caso ResNet

Tal como se muestra en la Figura 8, el camino que sigue la imagen a través de la CNN[22] tiene el nombre de **bottom-up path**, mientras que el camino que sigue a través de las capas intermedias de pirámide, tiene el nombre de **top-down path**.
Creación de la capa P5: Para la creación de la capa P5, simplemente se realiza una convolución 1×1 de la última capa convolucional de la CNN, posteriormente se le aplica una convolución 3×3 para generar la capa final P5.
Creación de las capas P2, P3 y P4: Para la creación de las capas P2,P3 y P4, se necesita crear las capas intermedias M2, M3 y M4, la capa M4 se crea a partir de realizar **upsampling** [16] x2 de la capa M1, se utiliza el algoritmo nearest neighbor[17], el motivo de utilizar este algoritmo es su simplicidad, ya que algoritmos más sofisticados ralentizarían la velocidad de la red. El upsampling[16] es necesario para que la capa M4 tenga el mismo tamaño que la capa C4, la capa M4 finalmente se construye tras aplicar una convolución 1x1 a la capa C4 y sumarla al resultado del upsampling[16], posteriormente se le aplica una convolución 3×3 a la capa M4 para reducir el efecto aliasing generado por el upsampling y formar la capa final P4, de la misma manera se forman el resto de capas P2 y P3.
La capa P1 no se genera ya que requeriría un coste computacional muy elevado, ya que la capa C1 tiene todavía un tamaño demasiado grande.

Hay que destacar que la FPN por si sola no es capaz de detectar objetos, es simplemente una arquitectura que permite una manera muy eficiente de detectar objetos a diferentes escalas, para la detección de objetos, se debe añadir una red neuronal de regresión y una red neuronal de clasificación a cada capa de pirámide (P2,P3,P4 y P5).

7 RETINANET

Retinanet es detector de objetos de una etapa que utiliza la arquitectura FPN, a la cual le añade dos pequeñas CNNs, una de ellas utilizada para la clasificación de bounding boxes[5], y la otra utilizada para regresión. La red puede entrenarse utilizando las redes neuronales más po-

pulares como backbone (ResNet [7],VGG [8],DenseNet [9],MobileNet [10]) El backbone se encarga de computar la imagen que se le proporciona como input, la subred de clasificación se encarga de predecir a que clase pertenece el bounding box. La arquitectura completa de RetinaNet se muestra en la Figura 9

7.1. Red de clasificación

La CNN[22] de clasificación tiene la función de asociar cada cada detección con una clase, la arquitectura de esta red consta de 4 capas convolucionales 3×3 con 256 filtros con activaciones ReLU[18], tras estas 4 capas se añade otra capa convolucional 3×3 que tiene como salida K A filtros, siendo K el número de clases a predecir y A el número de anchor boxes, que se fija en 9, como última capa se utiliza una activación sigmoidea por cada localización del mapa de características, tomando como referencia localizaciones de 50×50, la salida de la red seria un vector (2500,9K).

Como dato adicional, los parámetros de la red se comparten entre todas las capas de la pirámide, pero no se comparten con los parámetros de las redes de regresión a pesar de compartir la misma estructura.

7.2. Red de regresión

La red de regresión comparte la estructura principal con la red de clasificación, la principal diferencia reside en que esta red se encarga de predecir 4A outputs correspondientes a las coordenadas de los bounding boxes[5] por cada anchor box y por cada localización del mapa de características, por tanto la salida de la red seria un vector de dimensiones (2500,36)

7.3. Focal loss

La verdadera novedad introducida por la red RetinaNet se trata de una nueva función de error llamada **focal loss**. Antes de explicar en que consiste esta función, se va a explicar la necesidad de la misma en la detección de objetos de una fase. En cada imagen, la mayoría de elementos del mapa de características son background, esto quiere decir que ese elemento no es ningún objeto que el detector que se está entrenando es capaz de detectar, por tanto, esto produce un **desbalance** en cuanto al número de clases, para que una CNN funcione lo mejor posible, debe haber aproximadamente el mismo número de elementos de cada clase, es decir, los conjuntos deben ser balanceados, en el caso de un detector de objetos de dos fases, se obtienen las regiones de interés de la imagen antes de proporcionarla a la CNN, por este motivo estos detectores no tienen el mismo problema que los detectores de una fase.

Si el conjunto de datos está desbalanceado, el detector puede basar su función de error en aciertos fáciles, por ejemplo, un detector que asignase como background todos las posiciones del mapa de características, acertaría en la mayoría de ocasiones, pero nunca sería capaz de aprender de detectar objetos, para solventar este problema, RetinaNet hace uso de la función focal loss.

La función de error focal loss consiste en ”no ajustar demasiado los pesos basándose en aciertos fáciles”, para comprobar como funciona, se va a comparar con la función más utilizada habitualmente: cross entropy[20].

La función CE se define de la siguiente manera:

$$CE(pt) = -log(pt) \tag{1}$$

Donde $pt = p$ si $y = 1$ en caso contrario $pt = 1 - p$

Se ha definido la función focal loss de la siguiente manera:

$$FL(pt) - (1 - Pt^{\gamma} * \alpha * log(pt)) \tag{2}$$

Básicamente, se añaden las variables γ y α como regularizadoras, para penalizar los ejemplos fácilmente clasificados, para compararlo, se ha separado el comportamiento de las funciones en tres escenarios diferentes, para todos los casos se utiliza $\alpha = 0,25$ y $\gamma = 2$, ya que son los valores que mejor se comportan.

Escenario 1: Objeto clasificado fácilmente
Supongamos un objeto fácilmente clasificado con p = 0,9, la funcion CE calcularía lo siguiente:

$$CE = -log(0,9) = 0,1053$$

Mientras que focal loss:

$$FL = -1 * 0,25 * (1–0,9)^2 * log(0,9) = 0,00026$$

Si comparamos los resultados obtenidos, podemos observar que el error que se asigna mediante la función focal loss es $0{,}1053/0{,}00026 = 384$ **veces menor**.

Escenario 2: Ejemplo mal clasificado
Supongamos ahora un objeto al que se le ha asignado $p = 0,1$ por tanto se trataría de una clasificación errónea.

$$CE = -log(0,1) = 2,3025$$

Mientras que focal loss obtiene el siguiente resultado:

$$FL = -1 * 0,25 * (1–0,1)^2 * log(0,1) = 0,4667$$

En este caso, focal loss reduce el resultado obtenido en $2,3025/0,4667 = 5$ **veces menos**

Escenario 3: Ejemplo clasificado muy fácilmente

Se clasifica un objeto con $p = 0,99$, por tanto, CE devuelve lo siguiente

$$CE = -log(0,99) = 0,01$$

Mientras que focal loss:

$$FL = -1x0{,}25x(1–0{,}99) * *2log(0{,}99) = 2{,}5 * 10^{-7}$$

En este caso, la diferencia es todavía más grande, ya que $0{,}01/0{,}00000025 = 40{,}000$ veces mas pequeño.

Conclusion: Focal loss reduce el valor de la función de error en todos los casos respecto a CE[20], pero en el caso de los ejemplos clasificados fácilmente lo hace en un

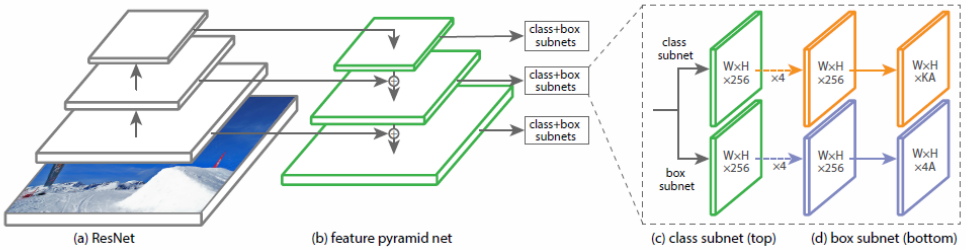


Fig. 9: Arquitectura completa de la red RetinaNet, que utiliza una arquitectura FPN a la que se le añaden dos pequeñas CNNs[22] de regresión y clasificación

factor mucho mayor, por tanto lo que se consigue con esto es ”balancear”las clases y que los ejemplos clasificados fácilmente no tengan tanto impacto en el entrenamiento de la CNN.

8 Equipo experimental

Para la realización de los experimentos, se ha utilizado como hardware un equipo con la siguientes características:

- **CPU:** Intel core I7 6700k.
- **RAM:** 16 GB DDR4.
- **HDD:** SSD Samsung Evo 840.
- **GPU:** Nvidia GTX 1080ti.

En cuanto a los datasets, se han escogido MS-COCO[15] y udacity[21] y los siguientes hiperparametros fijados, pues son los que mejores resultados se obtienen con RetinaNet
Learning Rate: 10^{-5} ajustándose automáticamente en caso de no mejorar.
Parámetros de Focal Loss: $\alpha = 0,25$ $\gamma = 2$

9 Resultados

En esta sección se comentarán los resultados tras entrenar en udacity[21] y MS-COCO[15], también se han variado los hiperparámetros de la red para estudiar su comportamiento.

9.1. Dataset Udacity

El primer dataset que se va a analizar es udacity, que consta de 15.000 imágenes diferentes, de las cuales se han seleccionado 12.000 para el entrenamiento y 3.000 para la validación, resultando en una partición de 80:20, cada época de entrenamiento ha llevado aproximadamente 40 minutos en completarse, los pesos de la red se han inicializado con los proporcionados por imagenet, por tanto se trata de un entrenamiento empleando la técnica **Fine Tuning** [19], de esta forma se reduce el tiempo de entrenamiento y generalmente consigue una mejora de AP.

En la Figura 10 se puede observar como la red asciende su AP rápidamente en las primeras épocas hasta situarse en aproximadamente 0,45, principalmente se debe a la clasificación de camiones, que al principio obtiene un AP bajo, no obstante, a partir de aproximadamente la época 15 no se

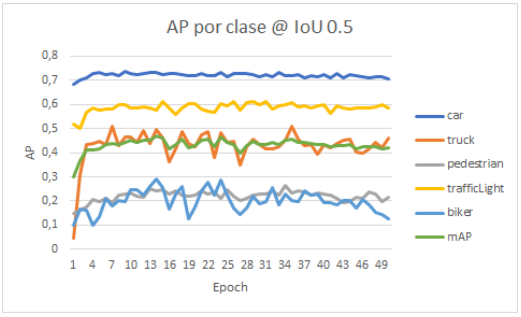


Fig. 10: AP por clase empleando el dataset udacity

obtiene una mejoría notable,y si se continua entrenando se puede apreciar que el AP comienza a descender, lo que desvela que la red está sobreentrenando, en cuanto a la función de error, se obtiene lo siguiente:

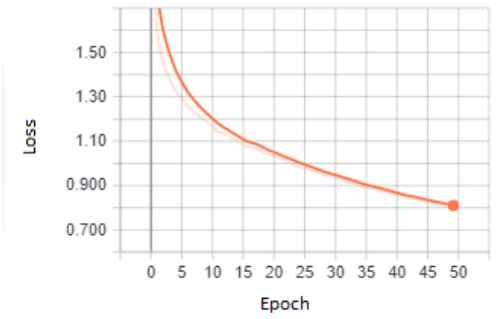


Fig. 11: Función de loss tras 50 épocas de entrenamiento en el dataset udacity

Una vez más, la función de error nos muestra que su valor desciende mientras entrena, pero sin embargo no se obtienen mejorías en AP, continuar entrenando la red provocaría aumentar el número de errores, para entrenar este dataset con el backend resnet101, serian necesarias de 10-15 épocas para conseguir los mejores resultados, esto se debe a que el dataset contiene tan solo 15.000 imágenes. Sin embargo, si se entrena el dataset sin emplear la técnica fine tuning, utilizando los mismos hiperparametros en la red que los que se han utilizado empleando los pesos de imagenet, y utilizando el mismo backbone, se obtiene el siguiente resultado: Como se puede observar, el mAP ha descendido de forma notable, se mantiene el mismo patrón, es decir, las clases que obtienen menos AP utilizando fine tuning son las mismas que

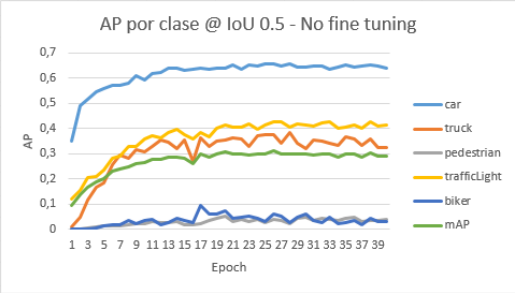


Fig. 12: AP por clase empleando el dataset udacity sin emplear la técnica fine tuning

obtienen menos AP cuando no se utiliza, no obstante, su AP disminuye de forma considerable, el máximo mAP obtenido sin realizar fine tuning ha sido de 0,3, aproximadamente 0,15 menos que utilizando fine tuning, se ha detenido el entrenamiento en las 34 épocas ya que no se mejoraban los resultados obtenidos. La siguiente Figura ayuda a contrastar los datos anteriores, midiendo la función de loss para el conjunto de validación:

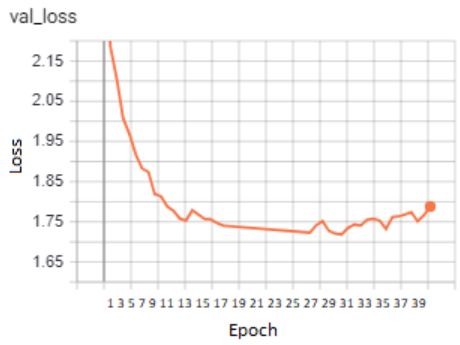


Fig. 13: Loss del conjunto de validación respecto al numero de épocas

En la Figura 13, se observa como el loss comienza a incrementarse en las últimas épocas, la Figura 12 ya había proporcionado esa información mediante el mAP, pero junto con el análisis de la función de loss, se puede determinar que la red no puede mejorar su mAP más allá de 0,3 sin utilizar fine tuning.

En la tabla 1 se puede observar el conjunto de pruebas que se han realizado con el dataset udacity y el mAP obtenido en todas ellas.

9.2. Dataset Coco

En cuanto al dataset coco, cuenta con 120.000 imágenes para el conjunto de train y 5000 imagenes para el conjunto de validación, hay 80 categorías de objetos diferentes, empleando RetinaNet, un batch size de 1 y un step size de 10.000 se tardan aproximadamente 96 horas en completar 50 épocas de entrenamiento, por este motivo se ha escogido udacity como el dataset principal en el que realizar la mayoría de pruebas. El resultado del entrenamiento mencionado, utilizando como backbone la red resnet101 es el siguiente:

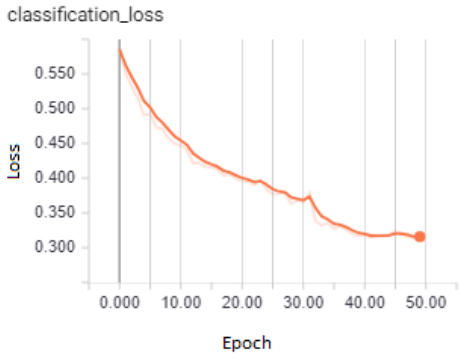


Fig. 14: Función de loss de clasificación tras 50 épocas de entrenamiento en el dataset coco

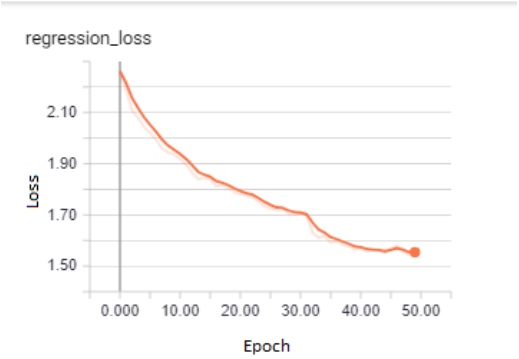


Fig. 15: Función de loss de regresión tras 50 épocas de entrenamiento en el dataset coco

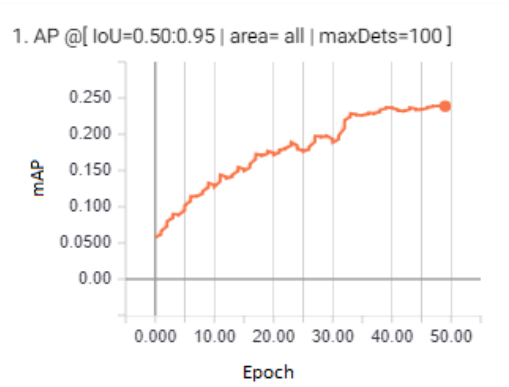


Fig. 16: mAP tras 50 épocas de entrenamiento en coco

9.3. Resultados realizando inferencia

En esta sección se va a analizar como se comporta RetinaNet ante una imagen que no conoce y los motivos por los cuales puede cometer errores, la detección y muestra de las detecciones tarda alrededor de unos 2 segundos dependiendo del tamaño de la imagen, en la siguiente figura se puede observar como es el resultado final:

TAULA 1: TABLA DE RESULTADOS OBTENIDOS EN EL DATASET UDACITY

Resultados en udacity con retinanet						
Backbone	Tamaño de imagen	train/val	Focal loss	Data augmentation	Pesos iniciales	mAP(%)
Resnet101	1400x1200	80/20	si	si	imagenet	43,6
	800x800	80/20	si	no	imagenet	25,9
	1200x1200	80/20	si	no	imagenet	36,4
	1400x1200	80/20	si	no	imagenet	**45,7**
	1920x1200	80/20	si	no	imagenet	45,6
	1920x1200	70/30	si	no	imagenet	44,2
	1920x1200	80/20	si	no	Random	31
	1920x1200	90/10	si	no	imagenet	42,4
	1920x1200	90/10	si	no	ms-coco	32,6
	1920x1200	80/20	no	no	imagenet	44,7
VGG19	1920x1200	80/20	si	no	imagenet	43,8
Densenet	1400x1200	80/20	si	no	imagenet	24

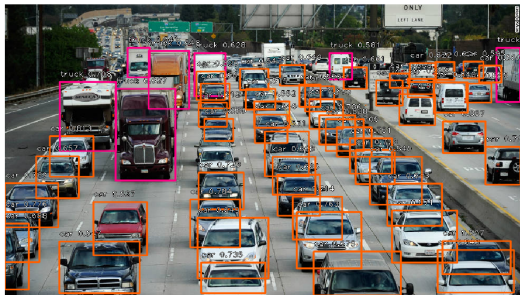


Fig. 17: Prueba de inferencia de RetinaNet en una imagen con tráfico denso.

10 CONCLUSIONES

Como conclusión de este estudio, se determina que el uso de librerías como Keras y Tensorflow hacen posible la creación y modificación de arquitecturas complejas como RetinaNet de forma sencilla, pudiendo centrar el esfuerzo en analizar y mejorar los modelos creados.

En cuanto a los resultados obtenidos, de ellos se puede extraer que para el caso del dataset udacity[21] es muy importante la realización de fine tuning[19], pues si no se realiza el dataset es demasiado pequeño como para que la red consiga generalizar correctamente y el AP desciende considerablemente, la resolución de las imágenes también es un parámetro que se debe tener en cuenta, pues si se reduce demasiado el tamaño los objetos pequeños no podrán ser detectados fácilmente y repercute de manera significativa en el AP en dichos objetos.
En cuanto a focal loss, la función de error utilizada por RetinaNet únicamente varia los resultados en un 1 % para este caso, lo que indica que efectivamente funciona mejor que entropia cruzada[20], no obstante no parece un incremento significativo para este dataset.

En el caso del dataset coco, los creadores de RetinaNet afirman que utilizando focal loss se incrementa el mAP de 0.3 a 0.4, en este estudio no se ha podido comprobar debido al exigente hardware requerido para poder entrenar el dataset coco de forma apropiada.

10.1. Trabajo futuro

Como ampliación al estudio realizado, sería muy interesante aumentar la compatibilidad de RetinaNet con otras redes neuronales como backbone, por ejemplo la NasNet [12], que promete unos resultados mejores que cualquiera de las redes existentes actualmente,no obstante, su adaptación a la arquitectura FPN está fuera del alcance de este proyecto.

AGRADECIMIENTOS

Mi más sincero agradecimiento a mi tutor Fernando Vilariño, por todo el soporte y la ayuda prestada, tanto de forma computacional, por el hardware requerido para este proyecto, como por el soporte ofrecido a nivel de conocimientos. Agradecer también a Jesús López Barrientos su ayuda prestada especialmente para los primeros pasos trabajando con la librería keras.

REFERENCIAS

[1] Jonathan hui
https://medium.com/@jonathan_hui/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c

[2] Tsung-Yi Lin1,2, Piotr Dollar, Ross Girshick1, Kaiming He1, Bharath Hariharan1 and Serge Belongie2, 2016
https://arxiv.org/pdf/1612.03144.pdf

[3] François Chollet, keras, 2015
https://keras.io/

[4] Google Brain, tensorflow, 2015
https://www.tensorflow.org/

[5] https://computersciencewiki.org/index.php/Bounding_boxes

[6] Koustubh Sinhal Sep 18,2016
https://medium.com/ilenze-com/object-detection-using-deep-learning-for-advanced-users-part-1-183bbbb08b19

[7] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun, Deep Residual Learning for Image Recognition.
https://arxiv.org/abs/1512.03385, 2015

[8] Karen Simonyan andAndrew Zisserman, 2014, http://arxiv.org/abs/1409.1556

[9] Gao Huang and Zhuang Liu and Kilian Q. Weinberger Densely Connected Convolutional Networks,2016
http://arxiv.org/abs/1608.06993

[10] Andrew G. Howard and Menglong Zhu and Bo Chen and Dmitry Kalenichenko and Weijun Wang and Tobias Weyand and Marco Andreetto and Hartwig Adam, 2017
http://arxiv.org/abs/1704.04861

[11] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He and Piotr Dollár
https://github.com/fizyr/keras-retinanet

[12] Google Inc, 2017
https://ai.googleblog.com/2017/11/automl-for-large-scale-image.html

[13] Matthijs Hollemans,2017, object detection with yolo
http://machinethink.net/blog/object-detection-with-yolo/

[14] Deeplearning.ai ejemplo anchor boxes
https://es.coursera.org/learn/convolutional-neural-networks/lecture/yNwO0/anchor-boxes

[15] Tsung-Yi Lin, Genevieve Patterson MSR
http://cocodataset.org/home

[16] Sweetwater
https://www.sweetwater.com/insync/upsampling/

[17] Steve Marschner, Alexei Efros, Noah Snavely
http://www.cs.toronto.edu/ guerzhoy/320/lec/upsampling.pdf

[18] SAGAR SHARMA
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[19] Felix-YU
https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html

[20] Rob DiPietro
https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/

[21] https://github.com/udacity/self-driving-car/tree/master/annotations

[22] Introduccion a las CNNs
http://cs231n.github.io/convolutional-networks/

[23] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He
https://arxiv.org/abs/1611.05431

[24] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg https://arxiv.org/abs/1512.02325

[25] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun https://arxiv.org/abs/1506.01497

[26] https://github.com/pjreddie/darknet

APENDICE

A.1. Pruebas en imágenes de alta resolución

Se ha probado RetinaNet en imágenes de alta resolución, el resultado se puede observar en la siguiente Figura:

Fig. 18: RetinaNet en una imagen de resolución 3264x2248

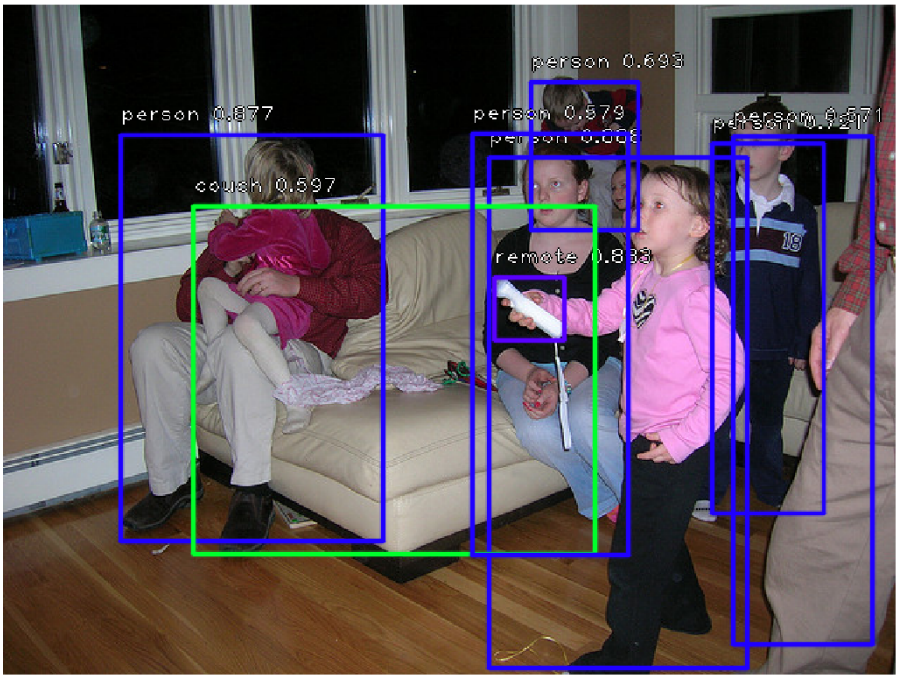
Como se puede observar, RetinaNet no funciona correctamente en imágenes de alta resolución,esto es debido a que los modelos se entrenan con imágenes de menor resolución, la arquitectura fpn[1] permite obtener imágenes de tamaños inferiores a partir de la imagen original, pero no al revés, por este motivo la inferencia no funciona como se espera, entrenar con datasets con imágenes de mayor resolución ayudaría a aumentar su efectividad, no obstante, con el hardware disponible actualmente no es posible realizar entrenamientos con imágenes de alta resolución.

A.2. Control de tráfico utilizando RetinaNet

Uno de los múltiples usos que puede tener un detector de objetos como RetinaNet puede ser el control del tráfico, ya que podemos obtener un número aproximado de los vehículos que se encuentran circulando, y a partir de este número obtener una estimación sobre el tráfico, básicamente se va a categorizar el tráfico en 6 posibles opciones.

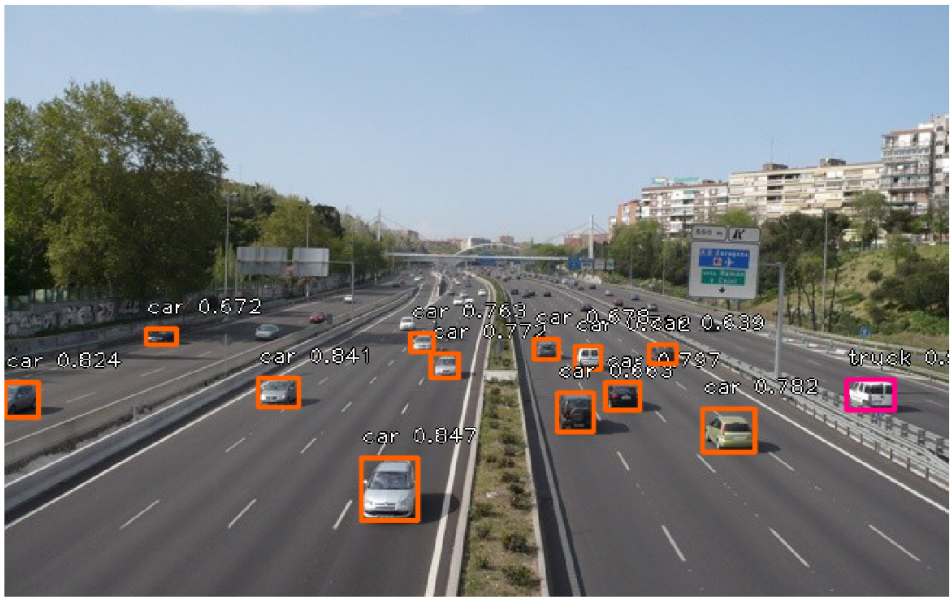
- No hay vehículos en la imagen proporcionada.
- Hay pocos vehículos y el trafico es ligero.
- Hay un tráfico medio.
- Trafico denso.
- Trafico muy denso.
- Imagen tomada desde angulo incorrecto, pues hay mucha densidad y pocos vehículos.

La aplicación no solo tiene en cuenta el número de vehículos en la imagen, también utiliza el área solapada de todos los vehículos para determinar el tipo de trafico de la imagen.



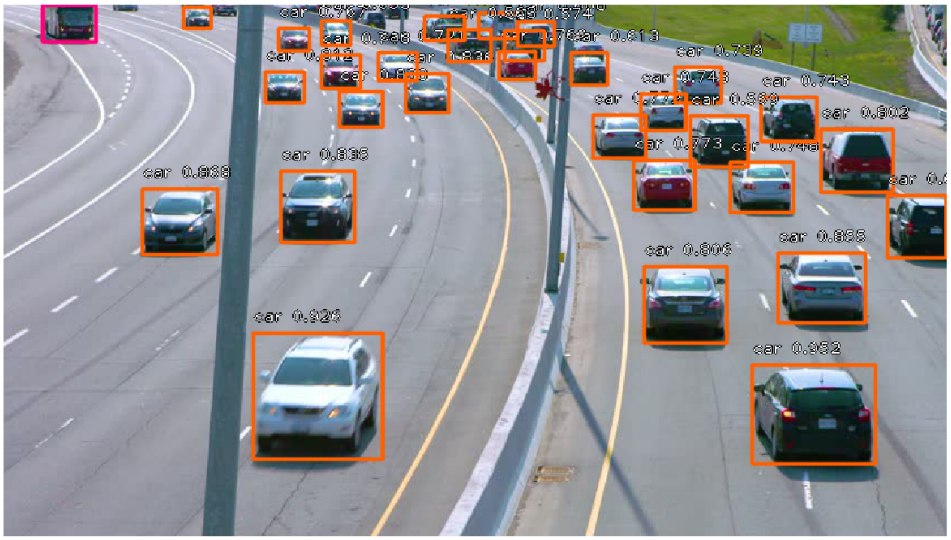
No hay vehiculos en la imagen

Fig. 19: Detección de imagen sin tráfico



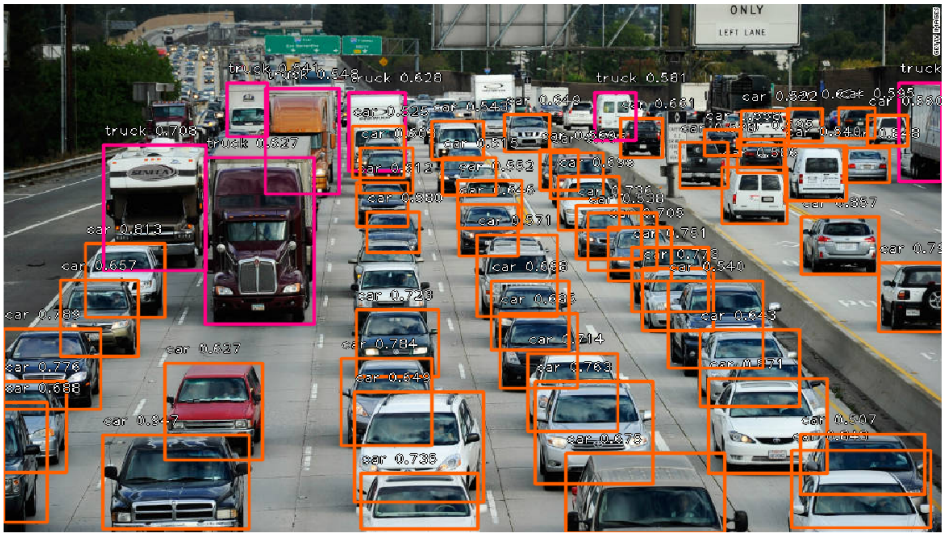
Hay vehiculos cirulando con un tráfico bajo

Fig. 20: Detección de poco tráfico



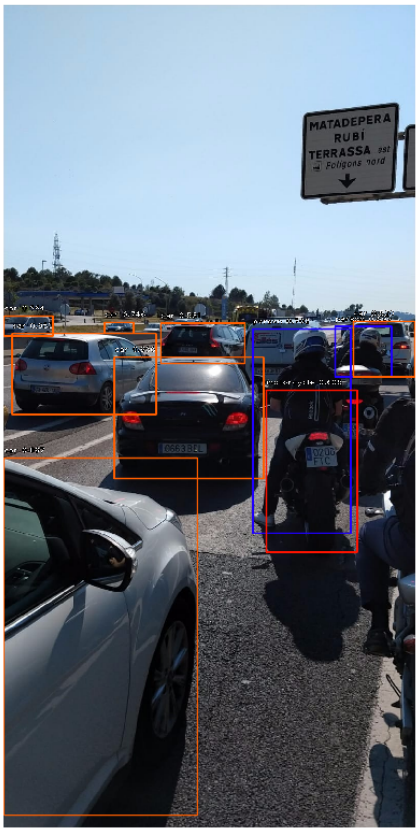
Hay vehiculos circulando con un tráfico medio

Fig. 21: Detección de tráfico medio



Hay muchos vehiculos circulando con un tráfico denso

Fig. 22: Detección de un tráfico alto



Hay pocos vehiculos muy solapados, se ha tomado la imagen desde un angulo incorrecto

Fig. 23: Detección de trafico utilizando un angulo incorrecto para la captura de la imagen