UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO E IMPLEMENTACIÓN DE UNA METODOLOGÍA BASADA EN INTELIGENCIA ARTIFICIAL CAPAZ DE CONTAR Y CLASIFICAR VEHÍCULOS EN LOS ACCESOS DE AEROPUERTOS Y DE RECONOCER VEHÍCULOS AEROPORTUARIOS

Trabajo Fin de Máster

Máster Universitario en Ingeniería Aeronáutica

AUTOR/A: Martinez Samper, Aniceto José

Tutor/a: Real Herráiz, Julia Irene

Cotutor/a externo: ANDRES LOPEZ, LAURA

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

TRABAJO FIN DE MASTER

DISEÑO E IMPLEMENTACIÓN DE UNA METODOLOGÍA BASADA EN INTELIGENCIA ARTIFICIAL CAPAZ DE CONTAR Y CLASIFICAR VEHÍCULOS EN LOS ACCESOS DE AEROPUERTOS Y DE RECONOCER VEHÍCULOS AEROPORTUARIOS

MASTER EN INGENIERÍA AERONÁUTICA

Autor:

Aniceto José Martínez Samper

Tutora:

Julia Irene Real Herráiz

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño

Valencia-Junio de 2022

RESUMEN

Actualmente, el tráfico es parte de un problema a nivel mundial. Existe un crecimiento muy acelerado, creando un problema notable en la sociedad. Es conveniente realizar un estudio de crecimiento de vehículos a lo largo de los años para contrastar el rápido crecimiento de la circulación. Este proyecto se basa en realizar un conteo y una clasificación automática de vehículos, capaz de poder acoplarse a cualquier video de circulación.

Particularizando un poco más, este proyecto se va a centrar principalmente en la composición de un algoritmo para el conteo y clasificación de vehículos en las entradas de aeropuertos. Siendo capaz de poder obtener una clasificación total durante una jornada de tiempo establecida de los vehículos que han transitado el aeropuerto, para conocer la ocupación y la demanda aeroportuaria. En el presente trabajo se realizará el algoritmo de implementación, pero no se realizará el estudio del tránsito de vehículos.

Posteriormente, se realizará un entrenamiento del algoritmo para ser capaz de reconocer vehículos aeroportuarios y llevar a cabo un control del tránsito en los aeropuertos. De esta forma, se puede conocer su ubicación en tiempo real y la posible necesidad de introducir nuevos vehículos.

Para realizar este estudio, se ha trabajado con el lenguaje de programación open CV. Donde se irá evolucionando el programa hasta poder conseguir un resultado óptimo, partiendo de una programación más simple, pasando por métodos básicos de procesamiento de imagen y finalizando en la implementación de redes neuronales (YOLO) para la detección y clasificación de objetos.

PALABRAS CLAVE:

- Inteligencia artificial
- YOLO
- Redes neuronales
- Procesamiento de imagen
- Clasificación de vehículos
- Aprendizaje profundo

ABSTRACT

Traffic is currently part of a global problem. There is a very accelerated traffic growth creating a big problem in society. It is appropriate to develop a study of vehicle growth over the years to compare how fast the traffic growth. This proyect is based on vehicle automatic counting and classification, being able to be adapted to any traffic video.

Specifically, this proyect will be focused mainly in a algorithm composition to vehicle counting and classification at airport's entrance. It will be able to obtain a total vehicle classification during a day, to determinate airport demand and occupation.

On the other hand, the algorithm will be trained to be able to recognize airport vehicles and carry out a traffic control. In addition, the location can be known in real time and the posibility to introduce new vehicles.

The programming language in this proyect is OpenCV. The algotithm will evolve to achieve and optimal solution, starting from a easy and simpler programming, going through basic image processing methods and ending with the implementation of a neural network (YOLO).

INDICE

INDICE DE FIGURAS

ÍNDICE DE CÓDIGO

ÍNDICE DE TABLAS

1. Introducción

En los últimos años se ha podido observar un crecimiento exponencial en la densidad de población y con ello, un aumento del tránsito de vehículos. Este fenómeno produce una congestión continua en calles, autopistas, parkings… en el entorno nacional. Derivando en una disminución de la eficiencia de las infraestructuras de transporte, un mayor consumo de combustible y contaminación del medioambiente.

En la actualidad, el avance tecnológico desarrollado en los últimos años, basado en las aplicaciones de visión artificial son cada día más viable para su desarrollo y puesta a punto como productos capaces de implementarse en entornos reales, como el control de tráfico.

Anteriormente, cuando aún no estaban desarrolladas estas aplicaciones, este tipo de tareas se realizaban de manera manual, siendo un trabajo muy tedioso, o no se realizaban. Gracias a los avances proporcionados en la actualidad, este tipo de trabajos se han automatizado facilitando su ejecución.

El conteo y clasificación de vehículos es una tarea muy habitual en el control de tránsito, pudiendo obtener datos reales y poder realizar un estudio a lo largo del tiempo. Permitiendo una toma de decisiones más eficiente para favorecer la movilidad y facilidad del tránsito. Esto puede ser aplicable por ejemplo a la distribución de semáforos en la ciudad, conocer la ocupación en un periodo de tiempo de un parking, la cantidad de vehículos que transita un aeropuerto… Es decir, la automatización del software consigue grandes beneficios, como el ahorro de ciertos recursos económicos, calidad y cantidad de datos recolectados y permite una toma de decisiones efectiva. Debido a esto, se ha desarrollado este proyecto, cuyo objetivo principal es el desarrollo de un sistema capaz de detectar, contar y clasificar vehículos. Pasando por las diferentes formas que se pueden llevar a cabo la detección de vehículos y los tratamientos de imagen… para finalmente acabar con un programa basado en inteligencia artificial denominado YOLO (You Only Look Once).

Por lo tanto, en este trabajo se va a realizar el diseño y optimización del software para su automática utilización, centrada en el conteo de vehículos en accesos de aeropuertos. Con el objetivo de analizar el volumen de vehículos que transitan, y conocer la ocupación en del aeropuerto dependiendo la hora que se encuentre. Estos datos pueden suponer posibles ampliaciones de terreno por la cantidad de vehículos que pueden transitar por el aeropuerto en determinados días y horas. En este proyecto no se realizará el estudio del tránsito en el aeropuerto, se realizará el algoritmo de implementación.

Posteriormente, también se va a realizar un entrenamiento del algoritmo principal y explicar su desarrollo para poder clasificar y detectar vehículos aeroportuarios.

Esto puede ser beneficioso para conocer la ubicación en tiempo real y la posible necesidad del aumento de vehículos debido a su gran demanda.

1.1. Objetivos

Para el cumplimiento del objetivo principal, se han establecido varios objetivos específicos para su implementación. En primer lugar, se van a desarrollar y analizar los diferentes algoritmos de detección y reconocimiento de objetos, finalizando con YOLO. Una vez estudiado y comparados los diferentes métodos, se implementará el sistema de detección, clasificación y conteo de vehículos. Finalmente, se realizará un estudio de un entorno concreto que permita evaluar el programa desarrollado. Durante el desarrollo del mismo, se abordarán explicaciones sobre inteligencia artificial, redes neuronales… y su funcionamiento, para poder conocer lo máximo posible el trabajo que se está llevando a cabo.

También se dispondrán de fotos de vehículos aeroportuarios para poder procesar y entrenar el algoritmo para ser capaz de reconocerlos.

2. Estado del arte

Para el aforo de tráfico, la tarea de conteo de vehículos es una tarea imprescindible para la implementación y ubicación de infraestructuras, para el control y optimización del tráfico urbano, para la ocupación de diferentes áreas… A lo largo de todo este tiempo se han implementados diversos métodos y procedimientos para realizar el conteo, que poco a poco se han ido mejorando para conseguir un funcionamiento más óptimo. La detección de objetos se compone de una combinación de clasificación y tratamiento de imagen junto a la localización precisa de objetos, que todo junto consigue una completa compresión de la imagen. Antiguamente, se realizaba el conteo mediante extracción manual de características, acompañada por métodos y arquitecturas sin mucha profundidad.

La inteligencia artificial poco a poco ha ido evolucionando, y particularmente la detección genérica de objetos ha sido desarrollada hasta un punto muy lejano. Actualmente, se pueden distinguir diversas categorías como la detección de caras, personas, vehículos… Por lo tanto, cuando se utiliza para la detección de vehículos ha podido dar un gran salto en cuanto a los métodos utilizados antiguamente.

Una vez explicado el desarrollo a lo largo del tiempo, estos métodos se categorizan mediante diferentes ventajas y desventajas.

- **Métodos manuales**: El conteo se realiza mediante un contador o el uso de una plantilla, compuesta por una o varias personas.
- **Métodos automáticos**: El conteo se produce con diferentes métodos y dispositivos.

Sensores	Conteo	Velocidad	Clasificación	Múltiples carriles	Intrusivo
Magnéticos	χ	χ			Si
Neumáticos	χ	χ			Si
Ultrasonido	χ				No
Infrarrojo	χ	χ	χ	χ	No
Rayo acústico	χ	χ		χ	No
Radar de microondas	χ	χ	χ	χ	No
Procesamiento de imágenes	χ	χ	χ	χ	No

Tabla 1: Métodos automáticos para el análisis del tráfico

Como se aprecia, hay diferentes tecnologías para el análisis del tráfico. Cada una de ellas tiene determinadas características de medición, incluyendo ser intrusivo o no. Esto quiere decir si es necesario la clausura temporal de la vía para realizar su instalación.

Dentro de las diferentes tecnologías, el procesamiento de imagen se encuentra entre las de mayores prestaciones. También cabe destacar, que para su implementación se puede utilizar las cámaras de seguridad ya instaladas en la vía pública sin necesidad de realizar ninguna instalación adicional. La eficiencia del mismo dependerá de la calidad de la cámara en uso y de la capacidad del sistema de visión artificial utilizado.

3. Inteligencia artificial

La inteligencia artificial es la parte de la ciencia que se ocupa del diseño de sistemas de computación inteligente, es decir, son las características que se asocian a la inteligencia en el comportamiento humano que hace referencia a la compresión del lenguaje, el aprendizaje, el razonamiento, resolución de problemas…

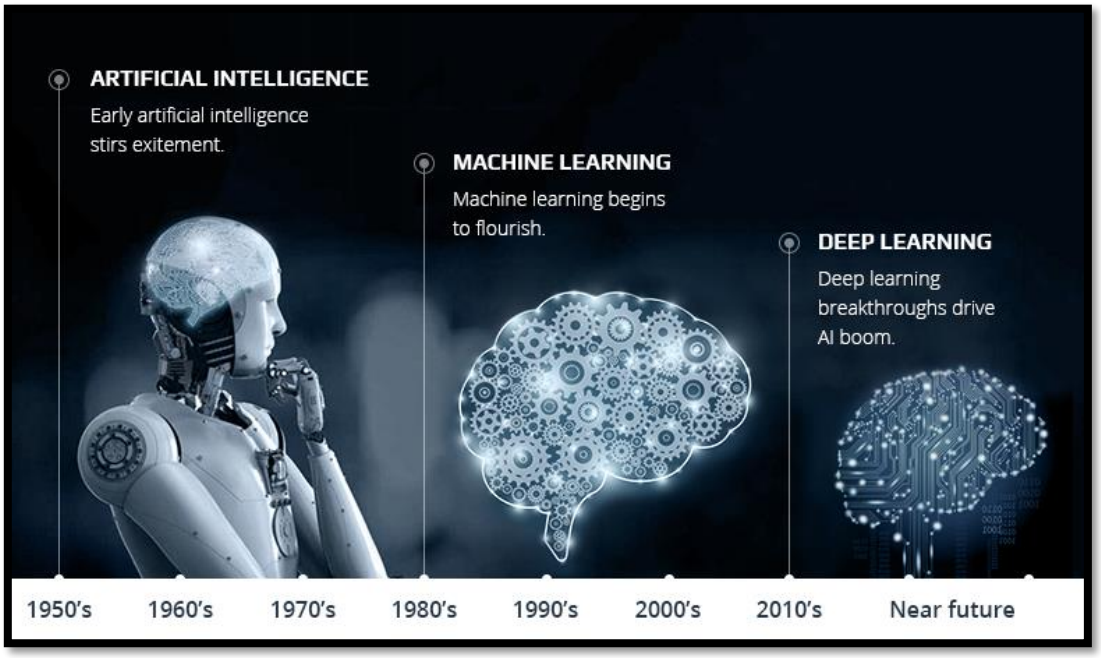


Figura 1: Inteligencia artificial

3.1 Machine Learning o aprendizaje automático

El aprendizaje automático o Machine Learning (ML) es una disciplina científica del entorno de la ingeniería y la ciencia de la computación, donde el objetivo principal es que los sistemas tengan un aprendizaje automático sin necesidad de intervención humana. Al contrario que las redes neuronales artificiales, este aprendizaje no se basa en el aprendizaje humano, experiencia o en la razón, sino es el reconocimiento de patrones complejos dentro de una gran cantidad de datos obtenidos mediante ejemplos. Una vez realizado el proceso de aprendizaje se obtiene un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros, pudiéndose adaptar a la incorporación de nueva información y recalibrar los resultados. Gracias a esto, no es necesarios la participación de personas, los analistas son necesarios para la participación en tareas como revisión y confirmación de decisiones.

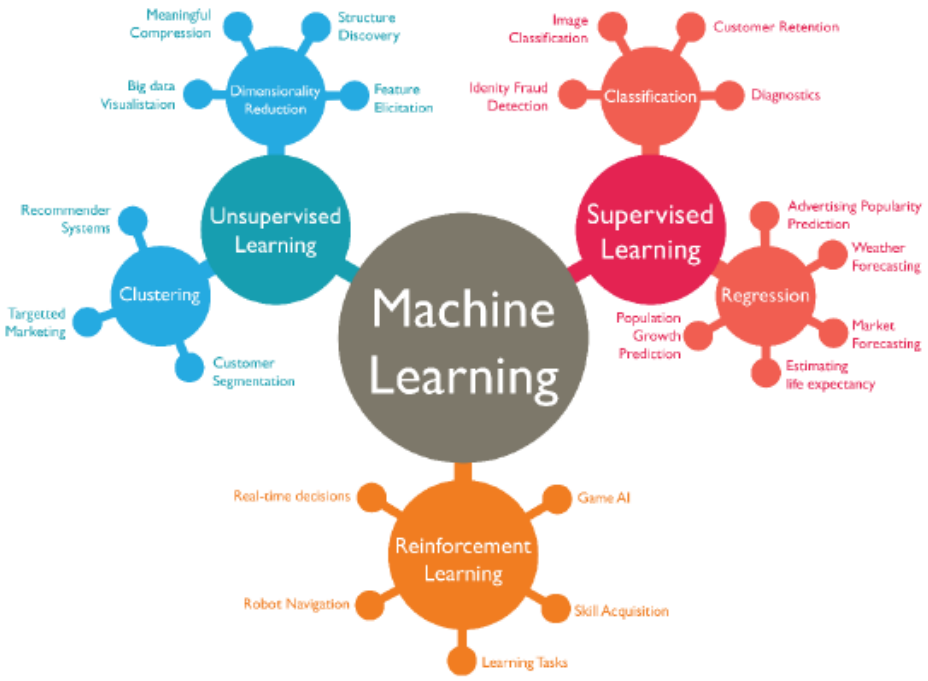


Figura 2: Machine Learning

Algunos de los modelos más conocidos son:

- **Aprendizaje supervisado (Supervised Learning):** se requieren datos previamente etiquetados para aprender a realizar la tarea. Con la introducción de datos se consigue adiestrar al sistema para problemas futuros. Dentro de este apartado, se reconocen dos tipos distintos.
    - **Regresión:** Se trata de un subcampo del aprendizaje automático supervisado cuyo objetivo es establecer un método para la relación entre cierto número de características y una variable objetiva continua.
        - Regresión lineal (utiliza datos continuos).
        - Regresión logística (utiliza datos discretos)
        - Árboles de decisión
        - Deep Learning
    - **Clasificación:** Los algoritmos de clasificación se usan cuando el resultado es una etiqueta discreta. Esto quiere decir que se usa cuando la respuesta se fundamenta en conjunto finito de resultados.

- **Aprendizaje no supervisado (Unsupervised Leraning):** para estos algoritmos no es necesario datos etiquetados con anterioridad. Sin embargo, es necesario algún tipo de indicación previa para analizar y comprender la información que se está utilizando para el aprendizaje.
- **Refuerzo por aprendizaje (Reinforcement Learning):** esta rama utiliza los resultados de la práctica, basándose en éxitos o fracasos producidos. Es decir, el sistema aprende solo al ir realizando pruebas.

3.2 Deep Learning o aprendizaje profundo

Esta rama, es una nueva técnica dentro del aprendizaje automático basado en arquitectura de redes neuronales. Trata principalmente en reproducir el proceso de solución de problemas del cerebro. Los seres humanos aplican el conocimiento adquirido a lo largo del tiempo con la experiencia a nuevos problemas o situaciones. Una red neuronal actúa de forma parecida, toma como ejemplo problemas resueltos para construir un sistema que toma decisiones y realiza clasificaciones. Los problemas más apropiados para el desarrollo neuronal son los que no tienen una solución computacional precisa o requieren algoritmos muy extensos como el caso de reconocimiento de imágenes.

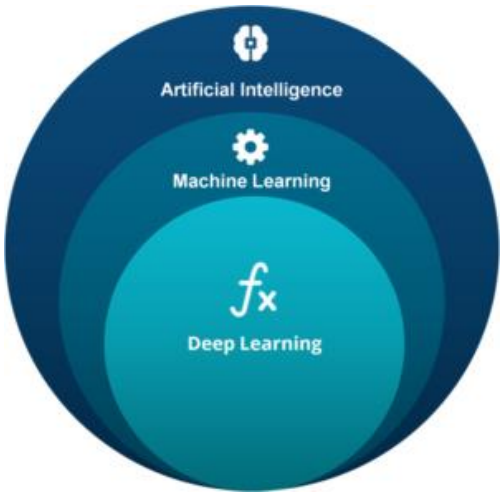


Figura 3: IA, ML y DL

Mientras los modelos tradicionales crean análisis mediante estructuras lineales, los modelos de Deep Learning se caracterizan por su estructura jerárquica permitiendo procesar datos con un enfoque no lineal. Este modelo aprende a realizar diversas tareas de clasificación directamente a partir de imágenes, texto o sonido, sin necesidad de la intervención humana para la selección de características.

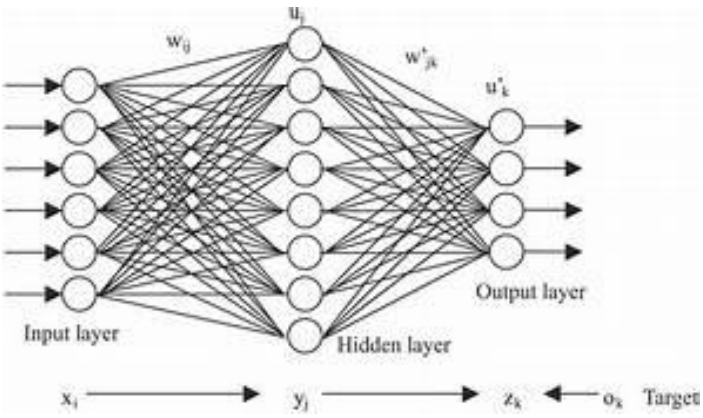


Figura 4: Red neuronal artificial por capas

El modelo se entrena mediante un amplio conjunto de datos etiquetados y arquitecturas de redes neuronales que contiene muchas capas. La primera capa de la red neuronal procesa la entrada de datos brutos como puede ser una imagen, y la pasa a la siguiente capa como salida. Este proceso se va repitiendo continuamente hasta recorrerlas todas. Al pasar por las capas, el programa va recopilando información y agrupándola hasta analizar capa por capa todas las características de la imagen. La principal característica de este método es que estas capas realizan el descubrimiento sin intervención humana.

Algunas ventajas de las redes neuronales artificiales comparadas con otros métodos de procesamiento de imagen pueden ser:

- Sintetizar algoritmos a través de procesos de aprendizaje.
- No son necesarios los conocimientos matemáticos detallados. Solo es necesario estar familiarizado con los datos de trabajo.
- Los problemas no lineales son el punto fuerte de las redes neuronales.
- La red es robusta, puede encontrarse algún fallo durante el procesamiento pero la red continúa funcionando, al contrario que en la programación tradicional.

Algunas de las desventajas que se encuentran son:

- Para afrontar cualquier problema la red debe de ser entrenada con anterioridad para determinar la arquitectura adecuada. El entrenamiento es largo y suele ser necesario una GPU.
- Se necesitan una cantidad considerable de datos para su entrenamiento.
- Para usuarios externos pueden representar un aspecto complejo si se desea realizar cambios. Para implementar nuevos conocimientos se necesita cambiar las iteraciones entre muchas unidades para sintetizar el conocimiento introducido.

Las redes neuronales constan de dos partes:

- **Propagación hacia delante (Forward propagation):** para poder obtener el resultado de salida de una red, previamente se debe calcular por capas el valor de cada una de las neuronas. Se avanza en el proceso hasta alcanzar las neuronas de salida una vez transcurridas todas las capas. Como se ha mencionado, las neuronas son sucesivas, es decir, que cada nueva capa de neuronas depende de la anterior, pero las neuronas de una misma capa son independientes y se puede paralelizar las operaciones.
- **Propagación hacia atrás o retropropagación (Backpropagation):** Una vez hecha la propagación hacia adelante, se calcula el error de la red neuronal con los resultados obtenidos. Posteriormente, se obtienen las derivadas parciales del error con respecto a los pesos que unen la última capa oculta con la de salida, repitiendo el proceso con las anteriores. Finalmente se ajustan los pesos de las neuronas con el objetivo de reducir el error.

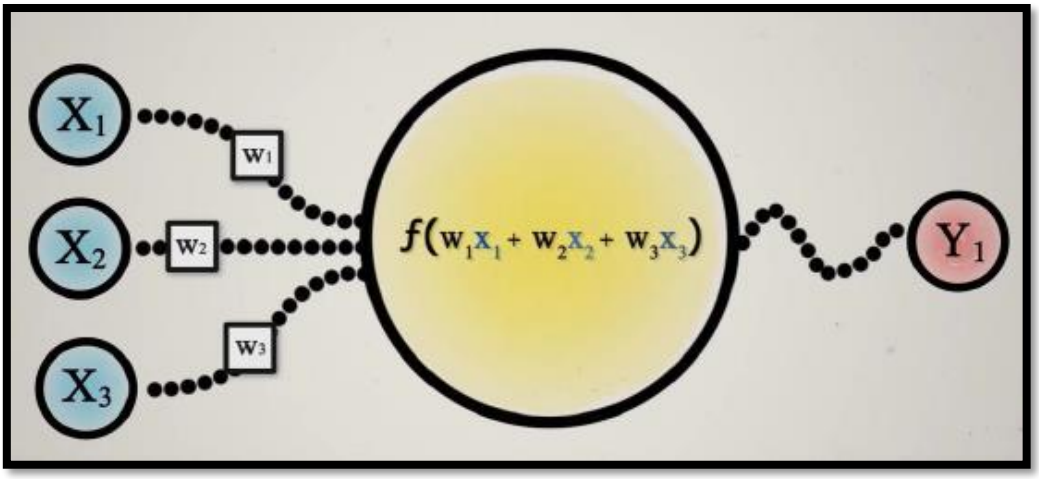


Figura 5: Funcionamiento de las neuronas

Algunas de las arquitecturas de Deep Learning emplean modelizaciones de redes neuronales como:

- Recurrent Neural Networks (RNN) o redes neuronales recurrentes.
- Generative Adversarial Networks (GAN) o redes generativas antagónicas.
- Convolutional Neural Network (CNN) o rede neuronales convolucionales.

3.2.1 Recurrent Neural Networks (RNN) o redes neuronales recurrentes

Este tipo de redes neuronales usan datos secuenciales o datos de series de tiempo. y solucionan problemas ordinales o temporales, como las traducciones, reconocimiento de voz, procesamiento del lenguaje natural y capturas de imágenes. Este estilo de redes se suele usar en tecnologías actuales como Siri o Google translate. Este proceso reconoce la voz de la persona, distinguiendo si es hombre o mujer, el acento, adulto o menor… De esta manera, se consigue analizar la forma de hablar y se puede llegar a su idiolecto.

Las redes neuronales recurrentes se distinguen de las otras redes artificiales en que tienen "memoria". Es decir, la RNN toma información de inputs anteriores para influenciar los inputs y outputs actuales.

3.2.2 Generative Adversarial Networks (GAN) o redes generativas antagónicas

Este tipo de redes consisten en oponer a dos redes neuronales (antagónica) para poder generar nuevo contenido o datos sintéticos que pueden hacerse pasar por reales.

Una de las redes se encarga de generar y la otra funciona como discriminadora. La red discriminatoria o antagónica ha sido entrenada para reconocer contenido real. Actúa como censor para que la red que se encarga de generar contenido, lo haga pareciendo real. Este tipo de redes se utilizan para generar imágenes, videos y voces.

3.2.3 Convolutional Neural Networks (CNN) o redes neuronales convolucionales

Para este tipo de arquitectura se emplean modelización de redes neuronales artificiales donde las neuronas corresponden a campos receptivos, comportamiento similar a las neuronas en la corteza visual de un cerebro humano. Este tipo de redes son muy habituales para diversas tareas, como la detección y categorización de objetos junto a la clasificación y segmentación de imágenes.

El objetivo principal de CNN es el aprendizaje de las características de orden superior utilizando la operación de convolución. Como las redes neuronales convolucionales pueden aprender relaciones de entrada-salida, cada pixel de salida es una combinación lineal de los pixeles de entrada.

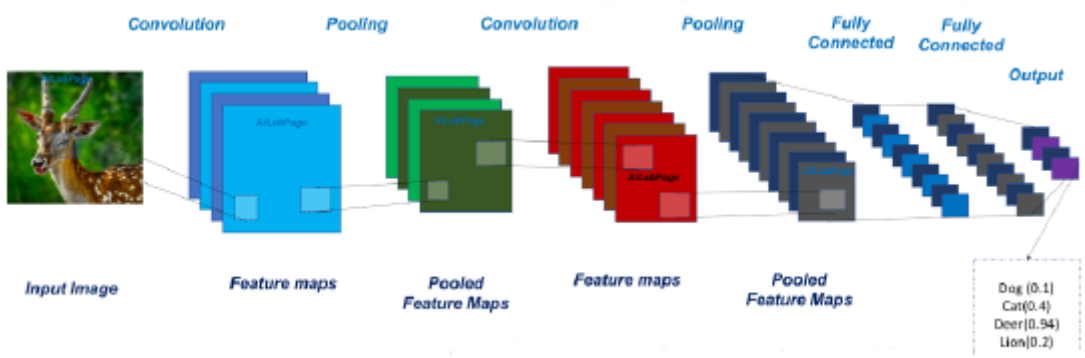


Figura 6: Redes neuronales convolucionales

La convolución trata de filtrar imágenes utilizando máscaras, donde diferentes máscaras producen distintos resultados. Las máscaras utilizadas representan las conexiones entre neuronas de capas anteriores. Estas capas aprenden progresivamente las diferentes características de orden superior de la entrada sin procesar. Este proceso que permite aprender características automáticas, es la principal característica del modelo.

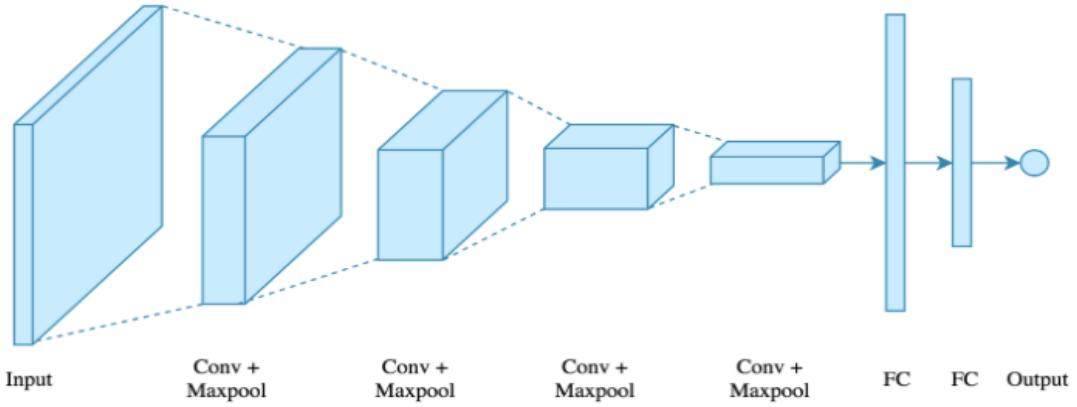


Figura 7: Capas de redes neuronales convolucionales

Las redes convolucionales se componen principalmente en dos tipos de capas: convolucionales y pooling.

La convolución transforma los datos de entrada utilizando la operación matemática convolucional, donde describe como fusionar dos conjuntos de información diferentes.

A continuación, las capas de pooling resumen las respuestas de salidas cercanas. En primer lugar, reduce progresivamente el tamaño espacial de los datos. En segundo lugar, la agrupación ayuda a obtener una representación invariable a una pequeña traslación de entrada. A continuación, se muestra la capacidad de reconocer las imágenes y las diferentes capas de una CNN.

Figura 8: Capa 1 de CNN

Figura 9: Capa 2de CNN

Figura 10: Capa 3 de CNN

Figura 11: Capa 4de CNN

3.2.3.1 Capas convolucionales

Como se ha comentado con anterioridad, la capa de entrada de una red neuronal convolucional tiene como neuronas de entrada los pixeles extraídos de la imagen. Es decir, que para una imagen de color se compone de 32x32x3 que son 3072 neuronas de entrada.

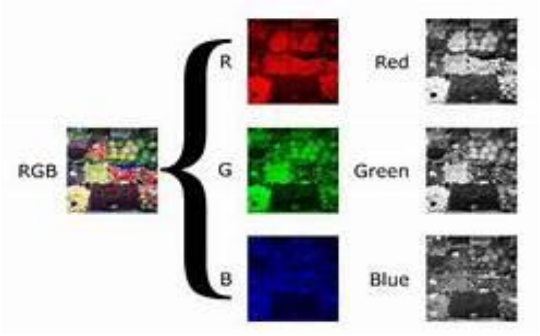


Figura 12: Descomposición en pixeles, imagen RGB

Los valores de los pixeles obtenidos de la imagen son muy parecidos con los adyacentes, por lo tanto, no será necesario usar todos los pixeles disponibles. Utilizando todos, solo se conseguiría aumentar la cantidad de computo necesario. Una vez llegados a este punto entra en escena la convolución.

Este proceso de convolución consiste en tomar grupos de pixeles cercanos de la imagen de entrada e ir haciendo el producto escalar con una pequeña matriz que se llama filtro o Kernel. Este proceso recorre todas las neuronas de entrada, de izquierda a derecha y de arriba abajo, produciendo una nueva matriz de salida, que será la nueva capa de neuronas ocultas.

Este proceso depende principalmente del tamaño proporcionado del kernel y del tamaño de saltos producido. Se puede calcular la dimensión o resolución de la matriz obtenida tras aplicar el proceso de convolución.

$$Resolución = \left(\frac{n + 2p - f}{s} + 1, \frac{n + 2p - f}{s} + 1\right)$$

Donde:

- **n**: número de pixeles en una imagen por filas o columnas.
- **p**: número de pixeles de relleno en la imagen.
- **f**: número de pixeles de un filtro en una fila o columna.
- **s**: número de saltos del filtro.

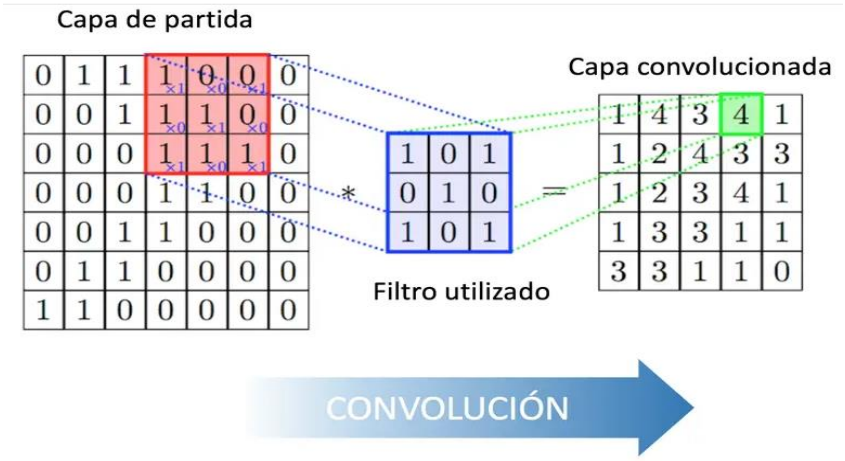


Figura 13: Convolución de una imagen

En primera instancia, los valores del filtro serán aleatorios, como en cualquier red neuronal. Mediante la retropropagación se ajustarán los valores. En una red neuronal convolucional no se aplica solamente un kernel, sino se tendrán un conjunto de filtros de cada capa. Al convolucionar los kernels con la imagen, se obtendrá una nueva imagen con ciertas características de la original. Esto es denominado mapeo de características (FeatureMapping).

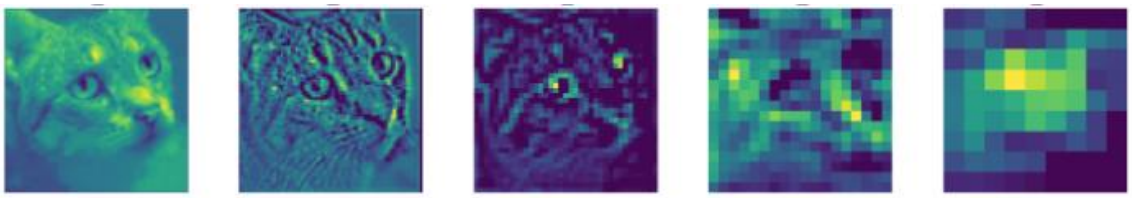


Figura 14: Mapeo de características o FeatureMapping

3.2.3.2 Capas de Pooling

Con las capas convolucionales se han obtenido nuevas matrices que muestran las características destacadas de la imagen original, pero con la misma dimensión. Por lo tanto, interesa reducir la dimensión de dichas características para intentar reducir el número de parámetros de la red. Se obtendrán las características más importantes que detectó cada filtro. Esta operación se llama submuestreo, siendo en imágenes la más común el Max-Pooling, que agrupa las características más destacadas del mapa de características. También se encuentra el Average-Pooling, que agrupa los valores promedio del mapa.

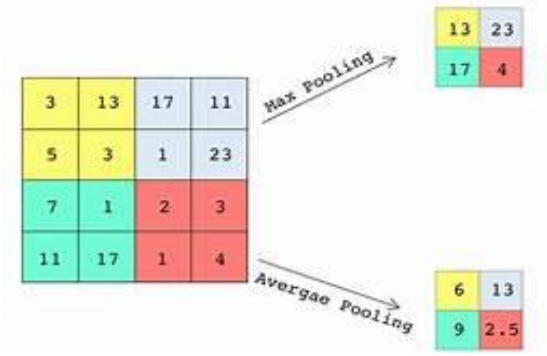


Figura 15: Capas de Pooling: Max Pooling y AveragePooling

Finalmente, uniendo lo explicado anteriormente se consigue obtener la primera capa de convolución.

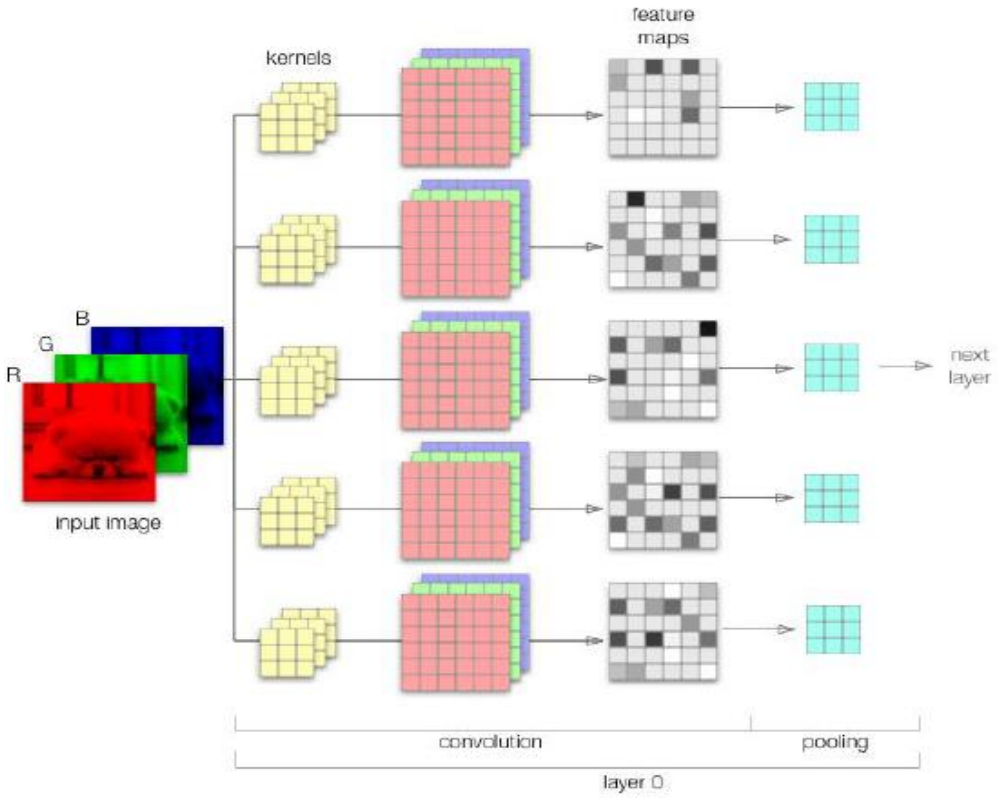


Figura 16: Primera capa de convolución

3.2.3.3 Capas Fully-Connected

Tras realizar el proceso de varias capas convolucionales, se cogería la última capa de Max-Pooling y se aplanaría obteniendo una capa tradicional de redes neuronales. Las redes neuronales multicapa tradicionales se les denomina Fully-Connected. Estas neuronas están completamente conectadas con las capas sucesivas, es decir, las neuronas de una capa están conectadas con todas las neuronas de la siguiente. La función principal de esta capa es obtener los resultados del proceso de convolución y usarlo para clasificar imágenes.

4. Plataforma de desarrollo

En la actualidad se encuentran mucha variedad de productos desarrollados por empresas para la implementación de algoritmos de visión artificial en el campo laboral. Algunas aplicaciones permiten el desarrollo de macros para la sistematización y automatización de procesos industriales. Estas soluciones suelen ser paquetes de software y hardware unificados, es decir, obliga al consumidor a tener que implementar tanto el hardware como el software.

También, existen plataformas de software de ingeniería como *MATLAB* que incluyen librerías de visión artificial muy potentes y de fácil instalación. Estos entornos de trabajo permiten su uso tanto en ordenadores portátiles como fijos, lo que facilita su implementación

Cabe destacar, que ambas opciones necesitan licencias para su utilización y limitan su desarrollo a su entorno de trabajo. Por esto se ha decidido realizar el programa en *Python*.

1.1 Python

Python y la inteligencia artificial están cada vez más presentes en sistemas operativos y aplicaciones de todo tipo. Es de esperar debido a su versatilidad, legibilidad y limpieza, haciendo este lenguaje de programación único, al centrar su código en una sintaxis más comprensible y sencilla.

Ventajas:

- Sencillez y facilidad de aprendizaje.
- Es un código abierto, para introducir mejoras y corregir errores.
- Abarca una gran comunidad dispuesta a ayudar a quien lo requiera.
- Versatilidad y flexibilidad.
- Excelente representación de datos y una gran profundidad en su análisis.

Una de las grandes ventajas es el gran número de librerías gratuitas y de libre distribución que existen.

1.1.1 Librerías utilizadas

- **OpenCV**

  Es la principal librería utilizada para desarrollar el programa. Es una biblioteca de software de visión abierta y aprendizaje automático. OpenCv fue construido para proporcionar una infraestructura común para aplicaciones de visión artificial y para acelerar el uso de la percepción de la

máquina en los productos comerciales. La biblioteca consta con más de 2500 algoritmos optimizados, tanto de visión artificial como de aprendizaje automático. Los algoritmos se pueden utilizar para detectar, reconocer, identificar y clasificar objetos.

- **NumPy**

  Biblioteca de manejo de vectores y matrices de alto nivel.

- **Matplotlib**

  Es la biblioteca más popular de generación de gráficos 2D

- **SciPy**

  Es un conjunto de librerías de optimización, algebra lineal, interpolación, procesamiento de imagen…

- **Neurolab**

  Librería para redes neuronales con algoritmos de entrenamiento integrados.

5. Procesamiento de imagen

Una vez explicado los entornos generales que engloban los campos de detección, clasificación y conteo de vehículos, se pasa a estudiar más en detalle los diferentes métodos que se pueden utilizar para su resolución. Antes de todo esto, para realizar el estudio del tráfico mediante cámaras o videos se debe realizar un procesamiento de imagen para intentar conseguir el menor error posible a la hora de detectar cualquier objeto. En primer lugar, se explicará generalmente el proceso que se suele llevar para el procesado de imagen y posteriormente con los diferentes algoritmos de detección que se explicarán se entrará más en detalle.

En la actualidad, se ha producido un gran avance en diferentes áreas como la ingeniería, medicina… En el campo de visión por computador, es decir, el análisis digital de imágenes por medio de un computador, en la década de los 80 se produjo una gran evolución que permitió que se realizaran las primeras aplicaciones en el área industrial utilizando visión por computador.

La imagen era su principal herramienta. La principal característica que hace referencia se denomina pixel, lo cual representa el nivel de intensidad de luz. Una imagen es representada como una matriz de pixeles.

Para realizar el análisis de una imagen es preciso considerar más específicamente un área dentro de la propia imagen, que se cataloga como la región de interés. Para conseguir esta región se suelen realizar operaciones geométricas que modifican las coordenadas espaciales de la imagen. Las operaciones geométricas tienen como principal objetivo transformar los valores de una imagen, es decir, podrían observarse desde otro punto de vista. De esta forma, las operaciones que se llevan a cabo como magnificar o reducir la imagen no es más que alejar o acercar el punto de vista.

Como se ha comentado, el procesado de imagen digital está compuesto por una serie de etapas en las cuales se intenta mejorar y optimizar la información que contiene una imagen, para posteriormente facilitar la interpretación y el tratamiento de la información contenida en ella. Cabe destacar, que dependiendo del método utilizado para la detección de objetos puede variar el tratamiento implementado a la imagen dependiendo de las necesidades del programa. Las principales etapas son:

- **Adquisición de la imagen:** este proceso consiste en capturar la luz emitida por una escena proveniente de un sensor, pudiendo encontrarse en cámaras de video, cámaras convencionales… y posteriormente almacenarla en un archivo con formato de imagen.

Figura 17: Adquisición de imagen

- **Realce de la imagen:** consiste en mejorar la apariencia visual de una imagen a través de técnicas que permiten la manipulación del contraste, brillo, nitidez, eliminar ruidos…

Figura 18: Realce de la imagen sin nitidez

- **Procesamiento de color:** este proceso consiste en la manipulación del color de la imagen, la cual puede tener más información que la imagen en niveles de gris. Esto permite combinar diferentes espacios como RGB a HSV, niveles de grises…

Figura 19: Procesamiento de color

- **Procesamiento morfológico:** se aplica una serie de operadores que permiten extraer las características útiles de una imagen, como los bordes, operaciones de llenado, filtro…

Las transformaciones morfológicas son algunas operaciones simples basadas en la forma de la imagen, que normalmente se aplica a imágenes binarias. Se necesitan dos entradas, la primera es la imagen original y la segunda se llama elemento estructural o kernel que decide la naturaleza de la operación. Los operadores morfológicos más usuales son erosión y dilatación. Después, se encuentra alguna variante como apertura, cierre, gradiente…

- **Erosión:** es un proceso similar a la convolución 2D, donde el kernel se desliza a través de la imagen. Los píxeles de la imagen original (1 ó 0) solo se consideran 1 si los píxeles caen dentro de la ventana del kernel son 1, sino se erosiona, es decir, se hace 0.

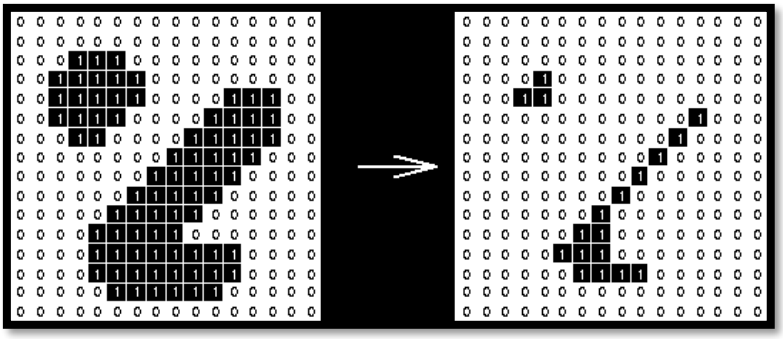


Figura 20: Procesamiento morfológico, Erosión

Entonces, todos los píxeles cerca de los bordes de los objetos de la imagen serán descartados dependiendo del tamaño del kernel. Como consecuencia el grosor y tamaño de los objetos del primer plano disminuye o en otras palabras, la región blanca disminuye.

Figura 21: Procesamiento morfológico, ejemplo de erosión

- **Dilatación:** Este proceso es justo lo opuesto a la erosión. Un elemento de píxel es 1 si al menos un píxel de la imagen de los que están en el interior de la ventana del kernel es 1. La dilatación aumenta el tamaño de los objetos de primer plano, es decir, aumenta la región blanca.

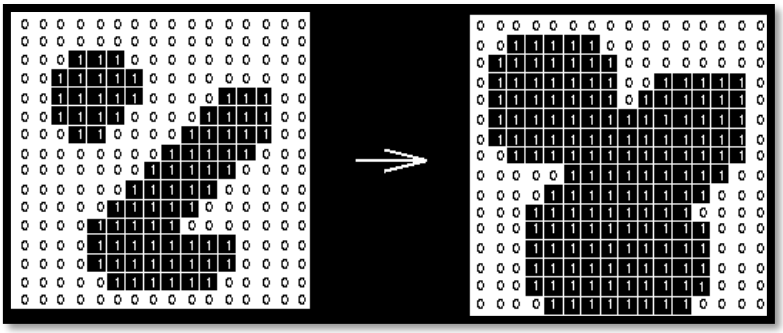


Figura 22: Procesamiento morfológico, Dilatación

En general, en casos donde se elimina el ruido, la erosión va después de la dilatación. Esto es debido a que, aunque la erosión elimina los ruidos blancos también encoge los objetos. Por lo tanto, para poder recuperar el tamaño inicial la imagen se debe de dilatar.

Figura 23: Procesamiento morfológico, ejemplo de dilatación

- **Segmentación de una imagen:** consiste en extraer de una imagen un objeto u objetos que son importantes para el proceso de representación y descripción. Se indica como extraer los elementos de una escena. Este proceso también es conocido como binarización de una imagen. Para este paso existen distintas técnicas básicas para el procesado, entre ellas, la más utilizada es la umbralización. Esto consiste en seleccionar los pixeles de una imagen que corresponde a un valor o rango de niveles de gris.

Figura 24: Segmentación de una imagen

- **Representación y descripción:** este proceso permite la extracción de las características de los objetos que fueron seleccionados en el proceso de segmentación. Entre ellos destacan la representación de bordes, cálculo de áreas, longitudes, direcciones, formas…

Figura 25: Representación y descripción

- **Reconocimiento:** este paso consiste en identificar la información contenida en la imagen mediante un patrón. Este proceso está compuesto por varias etapas. Primero se realiza el etiquetado, pasa por un proceso de condicionamiento para poder filtrar la información de la imagen, donde los pixeles correspondientes a un patrón se clasifican como pertenecientes a una clase u otra. En segundo lugar, el agrupamiento, donde se identifican los pixeles que corresponden a una misma clase. Posteriormente viene la extracción, donde se determinan las propiedades o características de los pixeles agrupados. Por último, se produce la comparación, donde finalmente un grupo de píxeles se asocian a un objeto ya conocido.

Figura 26: Reconocimiento

6. Algoritmos de detección y clasificador de objetos.

Se puede pensar que la detección de objetos es una tarea sencilla, pero realmente es un gran problema a resolver. Los humanos pueden observar una imagen e inmediatamente reconocer cualquier objeto, pero no es tan simple para su diseño digital.

Para poder cualificar como algoritmo de detección, es necesario que:

- El algoritmo debe detectar múltiples objetos.
- Debe proporcionar la posición X e Y del objeto en la imagen o su centro y dibujar un rectángulo a su alrededor.
- Otra alternativa, como se ha visto con anterioridad y se entrará en materia más adelante es la segmentación.
- Detectar "a tiempo" o puede que los resultados no sirvan, por ejemplo, con detección en tiempo real sobre el video

Para la detección de objetos se pueden utilizar diferentes metodologías de programación. En este proyecto se han estudiado diversos métodos para finalmente descartar los menos eficientes y poder crear un programa lo más optimo posible.

6.1 Clasificador en cascada

El primer modelo descrito es el clasificador en cascada, más concretamente los clasificadores de Haar. Fueron los primeros detectores de rostros en tiempo real. Este clasificador es un programa de detección de objetos de aprendizaje automático que identifica objetos en una imagen o video.

Para comenzar, cada clase de objeto tiene unas determinadas características especiales que ayudan a la clasificación de clase. Por ejemplo, todos los círculos son redondos, por lo tanto, la detección de clases del objeto utiliza estas características implementadas. Otro ejemplo, cuando se buscan cuadrados, se necesitan objetos que tengan los lados iguales y que las esquinas sean perpendiculares.

6.1.1 Viola-Jones

El algoritmo de Viola-Jones es un sistema que permite la detección de objetos en imágenes a tiempo real. Este programa tiene un coste computacional bajo, con una probabilidad de verdaderos positivos de alrededor del 99% y de falsos positivos

del 3.5%. Es una característica importante para dotar de fiabilidad a la hora de detectar.

En primer instancia, Viola y Jones crearon el algoritmo para la detección de caras, pero puede utilizarse para detectar cualquier clase de objeto. La detección de objetos es una tarea que requiere de tiempo, especialmente cuando el tamaño de la imagen aumenta. Hasta la actualidad, se han realizado muchos trabajos en la literatura en un intento por acelerar el proceso de detección de objetos. Esto es debido a la demanda actual de aplicaciones como sistemas de video de vigilancia en tiempo real.

A diferencia de otros tipos de algoritmos utilizados en la detección de objetos, este procesa solo la información presente en una imagen en escala de grises. No utiliza directamente la imagen real, sino una imagen en escala de grises denominada imagen integral. Para poder encontrar el objeto buscado en la imagen, el programa divide la imagen integral en subregiones de tamaños diferentes y utiliza una serie de clasificadores en cascada. Cada uno de ellos contiene un conjunto de características visuales. Este método permite ahorrar mucho tiempo al no procesar ciertas subregiones de la imagen, al ser capaz de afirmar con certeza que no contiene el objeto buscado. Solo se analizarán aquellas subregiones que posiblemente sí que lo contengan. A continuación, se procede a analizar paso a paso el funcionamiento del algoritmo.

6.1.1.1 Filtros tipo Haar.

Este tipo de filtros son similares a los filtros de detección de contornos, y se utilizan para detectar la presencia de ciertas características en una imagen.

Se utilizan tres tipos de filtros que acaban proporcionando un único valor como solución. El valor de un rectángulo se define como la diferencia entre la suma de pixeles em la región blanca y la suma de pixeles en la región gris.

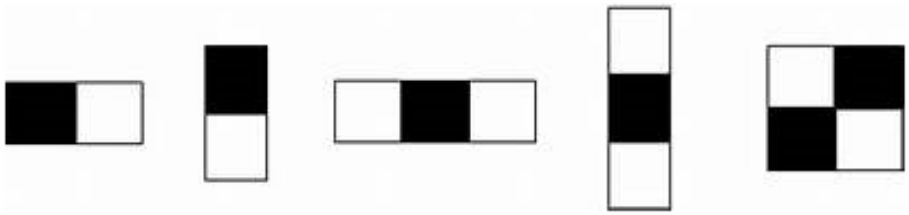


Figura 27: Clasificadores Haar

Como se aprecia, se pueden ver filtros de dos, tres y de cuatro rectángulos con diferentes posiciones. Los filtros se definen sobre una ventana de 24x24 pixeles. El tamaño y la posición de una ventana pueden variar siempre que sus rectángulos en blanco y negro mantengan la misma dimensión, se unan entre sí y continúen con sus posiciones relativas. Debido a estas restricciones, el número de características que se pueden sacar de una imagen son manejables. Pudiendo jugar con el tamaño, posiciones, escala y tipo, se pueden obtener más de 160,000 resultados posibles. Estos resultados se obtienen utilizando la imagen integral.

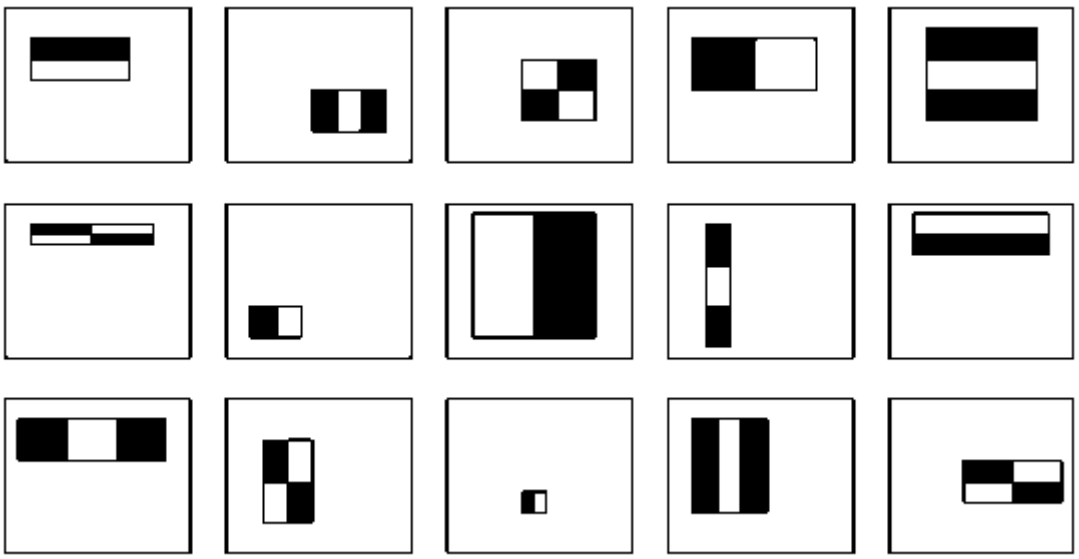


Figura 28:Posibilidades de filtros Haar

6.1.1.2 Imagen integral

Teniendo en cuenta que se tienen que aplicar estos filtros una cantidad considerable de veces por imagen, en vez de sumar todos los pixeles bajo cada uno de los bloques, se ideó un sistema llamado imagen integral. Es un mecanismo para realizar la suma de todos los pixeles dentro del recinto que se extiende por arriba y por la izquierda de cada uno de los pixeles de la imagen.

Input image

4	1	3	7	9
2	5	9	3	7
1	3	7	2	5
7	8	6	1	4
6	1	7	8	7

Integral image

4	5	8	15	24
6	12	24	34	50
7	16	35	47	68
14	31	56	69	94
20	38	70	91	123

Figura 29: Imagen original e imagen integral

Este proceso permite calcular la suma de todos los pixeles dentro de un rectángulo usando los valores de las esquinas pegadas al mismo.

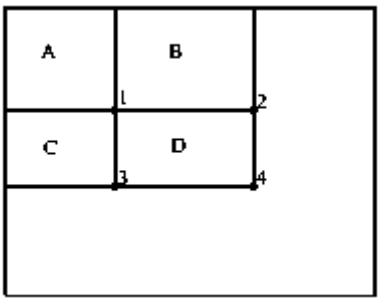


Figura 30: Obtención de una imagen integral

El valor del rectángulo D se puede obtener con cuatro valores de referencia. El valor del punto uno de la imagen es la suma de los pixeles que se encuentran dentro del rectángulo A, el punto dos es la suma de los rectángulos A y B, el punto tres es la suma del rectángulo A y C, por último, el punto cuatro será equivalente a la suma de A, B, C y D. De esta forma todos los pixeles que se encuentren bajo del rectángulo D se pueden calcular como el valor de la posición 4+1-(2+3) de la imagen integral.

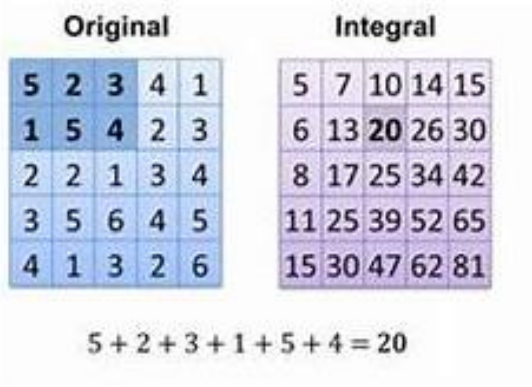


Figura 31: Ejemplo de una imagen integral

6.1.1.3 Aprendizaje en AdaBoost

Antes de proceder a explicar los clasificadores de cascada es necesario realizar un proceso de entrenamiento supervisado. Este entrenamiento se realiza con un algoritmo basado en AdaBoost, es un algoritmo adaptativo de machine learning.

Una vez dado un conjunto de filtros y de imágenes para el entrenamiento (positivas y negativas), se utiliza una variante de AdaBoost para seleccionar un conjunto de filtros y poder entrenar al clasificador. El algoritmo de aprendizaje se usa para potenciar el rendimiento de clasificación de un algoritmo de aprendizaje simple.

Como se ha comentado, se encuentran una cantidad muy grande de funciones de filtro rectangular. Aunque cada filtro se puede obtener de manera eficiente mediante la imagen integral, se requiere un coste computacional muy elevado para calcular el conjunto entero. De esta manera, mediante demostraciones

experimentales, la hipótesis resultante es que existe un número de filtros muy pequeño, que al combinarlos se obtiene un clasificador eficaz. El problema más importante es encontrar esos filtros. Por lo tanto, el algoritmo de aprendizaje se encarga de seleccionar los filtros que encajan mejor con las imágenes, habiendo realizado una selección de imágenes etiquetadas con ejemplos positivos y negativos previamente

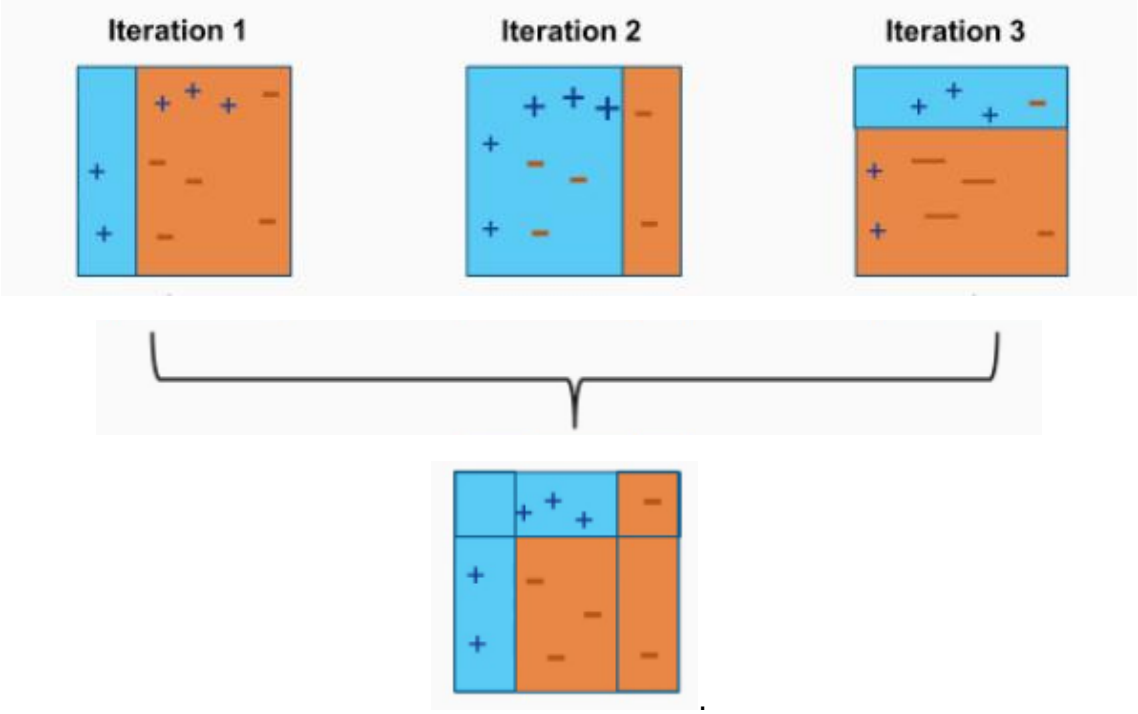


Figura 32: Proceso de selección de filtros

Una vez se han seleccionado los filtros más relevantes, se utiliza una combinación de los diferentes filtros para evaluar y decidir si contiene el objeto o no. Los filtros se incluyen al clasificador cuando en más de la mitad de casos ha podido encontrar el objeto buscado, estos filtros se llaman clasificadores débiles. Para la selección de los filtros, se entrenan clasificadores débiles, limitados a utilizar un solo filtro. El clasificador débil determina el límite que minimiza los ejemplos mal clasificados.

Como se aprecia en la imagen, mediante clasificadores débiles durante varias iteraciones, finalmente se consigue un clasificador fuerte.

6.1.1.4 Clasificadores Haar

El principio básico de implementación del algoritmo es el escaneo de la misma imagen múltiples veces con el mismo clasificador, pero distintos tamaños de imagen para poder identificar el objeto con diferentes tamaños. De esta forma será posible identificar el objeto, incluso si la imagen contiene varias veces el objeto buscado y en diferentes tamaños.

Existe algún inconveniente al escanear la imagen. Aparte del coste computacional que requiere, se producen una cantidad considerable de falsos positivos. Debido a

esto, el algoritmo debe centrarse en las subregiones con posibles positivos y no perder el tiempo en las regiones que no contienen el objeto. Por lo tanto, se utiliza un clasificador en cascada compuesto por varias etapas, donde cada etapa tiene un clasificador fuerte y los filtros se encuentran agrupados en las diferentes etapas, donde se determina si la subregión contiene o no el objeto. Una vez se identifica que no contiene el objeto se descarta inmediatamente, y si lo contiene continua a la siguiente etapa.

De esta forma se realiza una cascada de clasificadores, donde se entrena con AdaBoost y sus valores limite se ajustan para no cometer falsos negativos.

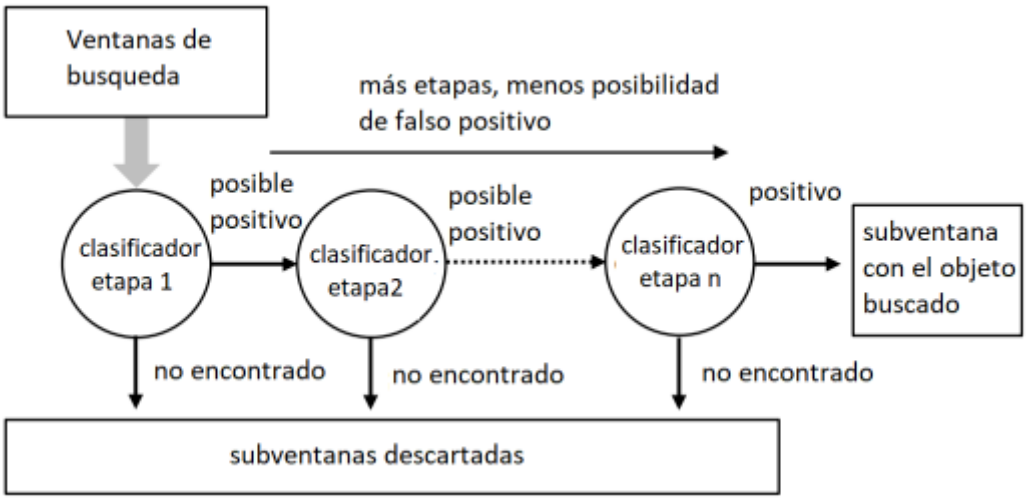


Figura 33: Esquema del clasificador en cascada

6.1.2 Código utilizado para clasificador en cascada para detección, clasificación y conteo de coche

Finalmente, este método representa un algoritmo fiable cuando se trata de detectar objetos, siempre que no se produzcan demasiadas variaciones muy bruscas en la imagen. El algoritmo de Viola-Jones tiene alguna desventaja a la hora de detectar variaciones de escala o rotaciones de objetos. En el presente trabajo, no es el mejor mecanismo de detección y clasificación. Al tratarse de videos que en ocasiones producen diferentes dimensiones del tamaño de la imagen de los coches al alejarse o acercarse de la cámara, no se produce una detección muy precisa. Para cada objeto que se necesite clasificar necesitará un archivo XML pre-entrenado, por lo que es costoso poder predecir varios objetos a la misma vez y en tiempo real. Por estos motivos este mecanismo de detección se ha descartado y se continuará con otros métodos.

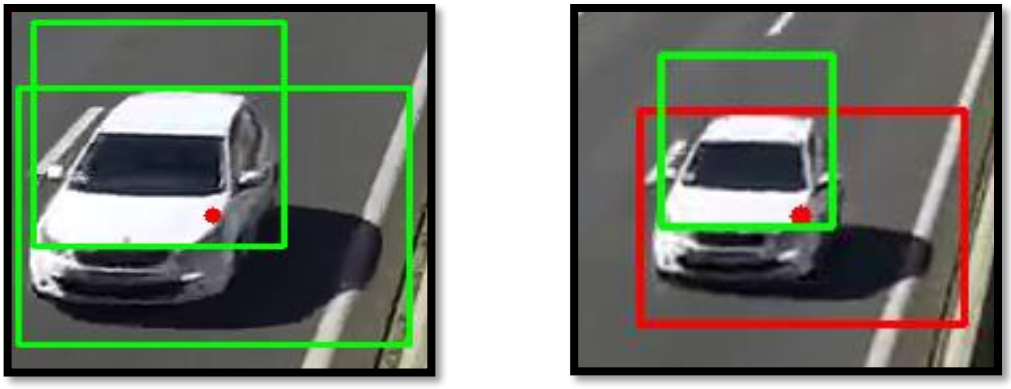


Figura 34 y 35: Detección con clasificador en cascada

A continuación, se procede a explicar un poco más en detalle las partes del código más importantes utilizado para este algoritmo. Cabe destacar que este código se utilizó como primeras pruebas, pero tras comprobar su funcionalidad en la aplicación de detección de coches en tiempo real se ha descartado. Debido a sus desventajas y al no proporcionar una buena eficiencia al compararlo con otros métodos. Por esto. es un ejemplo de código muy simple; posteriormente se explicarán el resto de métodos y finalmente el programa final utilizado.

Código:

- En primer lugar, se importa la biblioteca Python OpenCV.
- A continuación, se utiliza el método *VideoCapture* para poder procesar el video.
- Después, se le introduce la ruta de los modelos ya entrenados del vehículo con la función *CascadeClassifier*, que se componen como ya se ha explicado anteriormente con las características del objeto que se quiere determinar (*cars.xml*).

```
import cv2

cap = cv2.VideoCapture('video.mp4')
cars_cascade = cv2.CascadeClassifier('cars.xml')
```

Código 1: Ejemplo de algoritmo en cascada de detección de vehículos

- La función *cap.read* sirve para leer cada fotograma y da una matriz *frame1.*
- *cvtColor* cambia la imagen a escala de grises para ser procesada.
- Con *detect.MultiScale* se realiza la detección del objeto buscado.

```python
import cv2

cap = cv2.VideoCapture('stable_20211201074125_0005.mp4')
cars_cascade = cv2.CascadeClassifier('cars.xml')

while True:
        ret , frame1 = cap.read()
        grey = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
        cars = cars_cascade.detectMultiScale(grey, 1.15, 4)

        for (x, y, w, h) in cars:
            cv2.rectangle(frame1, (x, y), (x+w,y+h), color=(0, 255, 0), thickness=2)

        cv2.imshow("Video" , frame1)


        if cv2.waitKey(1) == 27:
            cv2.destroyAllWindows()
            cap.release()
            break
cv2.destroyAllWindows()
cap.release()
```

Código 1: Ejemplo de algoritmo en cascada de detección de vehículos

- Una vez obtenido los parámetros (x,y,w,h), que son (x,y) la posición y (w,h) el ancho y la altura. Se dibuja un rectángulo alrededor del objeto detectado mediante *cv2.rectangle*.
- Por último, se muestra por pantalla y con la tecla escape se finaliza el programa.

6.2 Análisis de movimiento y sustracción de fondo.

A continuación, se abordará otro método para realizar la detección, clasificación y conteo de vehículos. En primer lugar, se explicará algún concepto teórico y posteriormente se explicará las partes más importantes del código utilizado.

En este capítulo, se hará referencia a la sustracción de fondos (*background subtraction*) que es un método muy eficiente para la detección de movimiento. La sustracción de fondo es una técnica muy utilizada e importante en aplicaciones basadas en la visión artificial, técnicamente consiste en la extracción del primer plano en movimiento situado sobre el fondo estático. La técnica utilizada para este proceso es generar una máscara binaria para cada cuadro de video, donde los píxeles se dividen en el fondo de la imagen (*background*) y en los que se desplazan sobre el mismo, denominada *foreground.* El fondo de la imagen puede variar dependiendo de qué objetos se deseen detectar, en el marco de tiempo que se quiere realizar y otros criterios.

Existen numerosos algoritmos de detección de fondo, pero todos suelen seguir pasos generales, como son:

- Pre-procesamiento: tareas usuales de procesamiento de imagen para cambiar la entrada de video a un formato que pueda ser procesado en los siguientes pasos.
- Modelamiento de fondos: se conoce como mantenimiento de fondo.
- Detección del primer plano: se conoce como sustracción de fondos.
- Validación de los datos: se refiere al post-procesado utilizado para eliminar los pixeles que corresponden a objetos en movimiento, es decir, falsos positivos que aparecen en la imagen.

En la mayoría de casos se suele confundir el modelamiento de fondo y la sustracción de fondos, pero estos son procesos separados y diferentes. El modelamiento de fondo se refiere a crear y mantener un modelo de apariencia de fondo en el campo visual de la cámara. La sustracción de fondos es el proceso que se lleva a cabo para determinar dónde van a ir ubicados los píxeles, parte del fondo o del frente, comparando un cuadro de imagen con el modelo del fondo para determinar la posición de los pixeles individuales. Cabe destacar, que algunos píxeles que no cambian también pueden clasificarse como fondo si se tratara, por ejemplo, del mismo color. También ocurre en algunos píxeles que cambian, pueden ser clasificados como fondo, como, por ejemplo, las hojas de los árboles o la iluminación. Para poder mitigar esto se debe respetar:

- La cámara y sus parámetros tienen que ser fijos.
- La escena grabada no debe mostrar cambios de iluminación bruscos.
- El fondo debe de ser visible y estático.
- El fondo inicial no contiene objetos del frente estáticos.
- El frente y el fondo pueden ser separados al hacer una comparación entre la imagen del fondo y la imagen actual.

A pesar de estos criterios, en la práctica, resulta muy difícil cumplir todos ellos, por lo tanto, se deberán lidiar con algunos aspectos dependiendo de la situación que se encuentre el video. El modelamiento de fondo y la sustracción son pasos intermedios y no la solución final. Un sistema final y perfecto, debería ser capaz de resolver cualquier tipo de problema como objetos en movimiento, movimiento de los árboles, el movimiento del mar, sombras que cambian de manera gradual… Todos estos problemas son imposibles de resolverlos en paralelo. Cada uno necesita diferentes soluciones y una compresión diferente del movimiento del primer plano y el fondo. Como se ha comentado, a la hora de implementar el código se deben realizar restricciones para poder centrarse en los puntos fuertes y aprovechar al máximo las ventajas que ofrece una técnica frente a otra.

Para aplicar la teoría de sustracción de fondo al programa, existen varios modelos para la sustracción. Se explicará el que se va a utilizar para el algoritmo implementado en este proyecto. Después, se hará una pequeña introducción a los tres algoritmos de *BackgroundSubtractor*que se pueden implementar.

6.2.1 Modelo MOG adaptativo.

6.2.1.1 Descripción del modelo.

Se denomina modelo de mezclas gaussianas (*Gaussian Mixture Model*), que es un modelo probabilístico dado por una función de probabilidad de densidad paramétrica. Se compone de una combinación de varias distribuciones gaussianas lineales. Cada una de ellas tiene un parámetro peso w_k asociado. Estas distribuciones tienen la función de predecir, aproximar y representar la presencia de subpoblaciones dentro de una misma población a partir de muestras adquiridas de la población general.

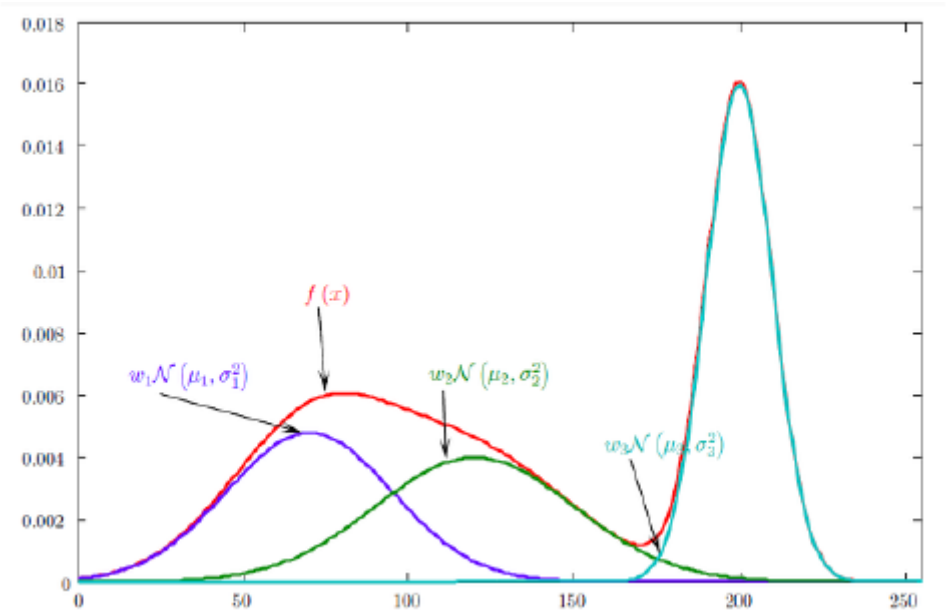


Figura 36: Función de densidad modelada para mezcla gaussiana

En la sustracción de fondo se debe modelar la función de distribución de probabilidad de cada píxel en cada escena con K distribuciones gaussianas. Este número se elige dinámicamente dependiendo la capacidad disponible para el cálculo y la memoria en el momento de ejecutarlo (se suele usar entre 3 y 5). Después, cada objeto se representará con una función gaussiana diferente, que tendrá un determinado peso w_k dependiendo de lo frecuente que aparezca ese objeto en la escena. Este método asocia la imagen de fondo al objeto más probable en la escena. Todo objeto que no esté representado por la gaussiana, será clasificado como imagen de frente.

El modelo permite representar el fondo con más escenarios u objetos. Esto puede ser útil en movimiento repetitivos. Un ejemplo puede ser el movimiento de los árboles debido al viento. Este movimiento podría catalogarse tanto como imagen de fondo como de frente, en función de los parámetros utilizados. Esto permite que la división entre fondo y frente no se base únicamente en el movimiento de

cualquier objeto, sino en la probabilidad de encontrar a cada objeto en esa posición.

6.2.1.2 Modelo matemático.

Se define la probabilidad de que un cierto pixel tenga el valor x_N de intensidad a tiempo.

$$p(x_N) = \sum_{k=1}^{k} w_k \, \eta(x_N; \theta_k)$$

Donde K es el número de gaussianas, w_k es lo que determina el peso de la k-ésima componente gaussiana y $\eta(x_N; \theta_k)$ es la distribución normal del k-ésimo componente, representada por:

$$\eta(x_N; \theta_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_K|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k)}$$

Donde μ_k es la media de la distribución y Σ_K la covarianza del k-ésimo componente. Esto implica que los valores de las componentes R, G y B del pixel son independientes y tienen la misma covarianza.

Cada distribución describe un solo objeto, independientemente de si se trata de un objeto de fondo o de frente y la probabilidad de que el valor de x_N del pixel pertenezca al objeto k modelado por la k-ésima distribución gaussiana es w_k.

6.2.1.3 Estimación de parámetros.

El método utilizado para poder inferir en los parámetros del modelo es el algoritmo de *Expectation-Maximization,* con el cual se encuentran estimadores de máxima verosimilitud. Con un valor x_{N+1} de un píxel, este algoritmo también permite inferir en la clase más probable k. Mediante el teorema de Bayes y las ecuaciones anteriores.

$$\hat{k} = argmaxp(w_k | x_{N+1}) = argmax \, w_k \eta(x_{N+1}; \theta_k)$$

Este método resulta muy costoso computacionalmente, por lo tanto, se reemplaza el criterio por una aproximación. Cada valor de x_{N+1} de un píxel es comparado con cada k-ésima componente gaussiana hasta encontrar una coincidencia.

$$\mu_k - 2.5\sigma_k < x_{N+1} < \mu_k + 2.5\sigma_k$$

Donde 2.5 veces la desviación estándar es un valor empírico demostrado para una implementación robusta frente a cambios de iluminación.

$$\hat{p}(w_k | x_{N+1}) = \begin{cases} 1 \; si \; x_{N+1} \; coincide \; con \; w_k \\ 0 \quad C.C \end{cases}$$

Si no se consigue coincidir ambos valores, la distribución menos probable será reemplazada por una nueva distribución de media x_{N+1}.

6.2.1.4 Actualización de distribuciones.

Una vez estén asociadas las componentes gaussianas al valor del pixel, se deberá actualizar sus parámetros en cada ciclo del algoritmo. Esto se realiza mediante las ecuaciones de actualización del método *Expectation-Maximization*. El peso se actualiza con la siguiente regla.

$$\hat{w}_k^{N+1} = (1-\alpha)\hat{w}_k^N + \alpha\hat{p}(w_k|x_{N+1})$$

Siendo 1/α la constante de tiempo que determina cuando serán actualizados los pesos de la distribución gaussiana o tasa de aprendizaje (*learning rate*). Esta ecuación viene de un filtro paso bajo discreto que se aplica a la media de la probabilidad. Los parámetros de la distribución gaussiana no serán analizados si no corresponden con el valor del píxel. Mientras la gaussiana que si pertenece se modifica con la siguiente regla.

$$\hat{\mu}_k^{N+1} = (1-\alpha)\hat{\mu}_k^N + \rho x_{N+1}$$

$$\hat{\Sigma}_k^{N+1} = (1-\alpha)\hat{\Sigma}_k^N + \rho(x_{N+1} - \hat{\mu}_k^{N+1})(x_{N+1} - \hat{\mu}_k^{N+1})^T$$

Siendo

$$\rho = \eta\left(x_{N+1}; \hat{\mu}_k^N; \hat{\Sigma}_k^N\right)$$

Esta regla también es un filtro paso bajo del mismo tiempo. La ventaja de este método es permitir cambiar de modelo de fondo sin destruir la información del anterior. Esto permite alternar entre modelos de fondo.

6.2.1.5 Estimación del modelo de fondo.

En esta sección se pretende seleccionar la distribución gaussiana que describe el modelo de fondo. Para esto se ordenan las K distribuciones en función de la aptitud w_k/σ_k y las primeras B son usadas por el modelo de donde de la escena.

$$B = argmin\left(\sum_{j=1}^{b} w_j > T\right)$$

Donde T es el valor umbral que determina la probabilidad de que el píxel analizado pertenezca al fondo. Mediante este valor se fija la cantidad de gaussianas que modelan la imagen.

6.2.2 Algoritmos de sustracción de fondos.

6.2.2.1 BackgroundsubtractorMOG.

El modelo que sigue este algoritmo se ha explicado anteriormente. Es un algoritmo de segmentación de fondos basado en la combinación gaussiana. P. KadewTraKuPong y R. Bowden lo presentaron en el documento "Un modelo de mezcla de fondo adaptativo mejorado para el seguimiento en tiempo real con detección de sombras" en 2001. Se necesita crear un objeto de fondo usando *cv2.createBackgroundSubtractorMOG().* Luego, dentro del bucle donde se lee el video se utiliza el *backgroundsubtractor.apply()* para obtener la máscara de primer plano.

```
import cv2
import numpy as np
cap = cv2.VideoCapture('stable_20211201074125_0005.mp4')
subtracao =cv2.bgsegm.createBackgroundSubtractorMOG()
while True:
                    ret , frame1 = cap.read()
                    img_sub = subtracao.apply(frame1)
                    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
                    dilatada = cv2.morphologyEx (img_sub, cv2. MORPH_CLOSE , kernel)

                    cv2.imshow("Video Original" , dilatada)
                    if cv2.waitKey(1) == 27:
                        cv2.destroyAllWindows()
                        cap.release()
                        break
cv2.destroyAllWindows()
cap.release()
```

Código 2: Ejemplo de algoritmo de sustracción de fondos MOG para detección de vehículos

Estos algoritmos son un simple ejemplo del funcionamiento de la sustracción de fondos para comparar el resultado de los diferentes algoritmos. Posteriormente, se explicará el programa utilizado para la clasificación de vehículos con más detalle.

Figura 37, 38 y 39: Algoritmo MOG para moto, coche y furgoneta

Figura 40: Algoritmo MOG

6.2.2.2 BackgroundsubtractorMOG2

Este algoritmo también es de segmentación de fondo basado en mezclas gaussiana. Está basado en dos artículos de Z.Zivkovic, "Modelo de mezcla gausiana adaptable mejorado para la sustracción de fondo" en 2004 y "Estimación de densidad adaptativa eficiente por píxel de imagen para la tarea de sustracción de fondo" en 2006. Una característica importante es la selección de un número apropiado de distribuciones gaussianas por cada píxel. En el anterior, se elegía K distribuciones gaussianas a lo largo del algoritmo. Por lo tanto, se consigue una mejor adaptabilidad para diferentes escenas debido a cambios de iluminación.

Como con MOG, se crea un objeto sustractor de fondo. En MOG2 se puede elegir si detectar sombra o no (*detectShadow= True or False*).

```
import cv2
import numpy as np
cap = cv2.VideoCapture('stable_20211201074125_0005.mp4')
subtracao =cv2.bgsegm.createBackgroundSubtractorMOG2()
while True:
                    ret , frame1 = cap.read()
                    img_sub = subtracao.apply(frame1)
                    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
                    dilatada = cv2.morphologyEx (img_sub, cv2. MORPH_CLOSE , kernel)

                    cv2.imshow("Video Original" , dilatada)
                    if cv2.waitKey(1) == 27:
                        cv2.destroyAllWindows()
                        cap.release()
                        break
cv2.destroyAllWindows()
cap.release()
```

Código 3: Ejemplo de algoritmo de sustracción de fondos MOG2 para detección de vehículos

A continuación, se muestran los mismos ejemplos sacados de un video para poder comparar la detección de movimiento en cada vehículo.

Figura 41, 42 y 43: Algoritmo MOG2 para moto, coche y furgoneta

Figura 44: Algoritmo MOG2

6.2.2.3 BackgroundsubtractorGMG

Este algoritmo está modelado con una combinación de la estimación de la imagen de fondo y la segmentación bayesiana por píxel. Fue presentado por Andrew B. Godbehere, AkihiroMatsukawa, Ken Goldberg en su artículo "Seguimiento visual de visitantes humanos en condiciones de iluminación variable para una instalación de arte de audio sensible" en 2012.

Este algoritmo utiliza los primeros fotogramas (120) para realizar el modelado del fondo. Para identificar posibles objetos de primer plano mediante inferencia bayesiana, utiliza un algoritmo de segmentación de primer plano probabilístico. Las observaciones más recientes son más importantes que las antiguas por adaptarse a la iluminación variables, es decir, las estimaciones son adaptativas.

```
import cv2
import numpy as np
cap = cv2.VideoCapture('stable_20211201074125_0005.mp4')
subtracao =cv2.bgsegm.createBackgroundSubtractorGMG()
while True:
                    ret , frame1 = cap.read()
                    img_sub = subtracao.apply(frame1)
                    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
                    dilatada = cv2.morphologyEx (img_sub, cv2. MORPH_CLOSE , kernel)

                    cv2.imshow("Video Original" , dilatada)
                    if cv2.waitKey(1) == 27:
                        cv2.destroyAllWindows()
                        cap.release()
                        break
cv2.destroyAllWindows()
cap.release()
```

Código 4: Ejemplo de algoritmo de sustracción de fondos GMG para detección de vehículos

Figura 45, 46 y 47: Algoritmo GMG para moto, coche y furgoneta

Figura 48: Algoritmo GMG

6.2.3 Código utilizado en análisis de movimiento y sustracción de fondo para detección, clasificación y conteo de coches.

El siguiente paso para la detección, clasificación y conteo de vehículos tras haber explicado el algoritmo de clasificación en cascada ha sido la detección de movimiento mediante la sustracción de fondo. A continuación, se va a explicar un poco más en detalle las partes más importantes del algoritmo utilizado para el conteo de vehículos. No se ilustrará el código entero porque muchas partes del mismo son repeticiones para poder clasificar los vehículos, y no tener que repetir lo mismo. Se explicará el código catalogando solo las motos, pero sería igual para el resto de vehículos.

Es importante mencionar que la detección y clasificación de vehículos ha sido más óptima y eficiente que en la clasificación en cascada. Parte del programa final utilizado se basa en la sustracción de fondos, pero se comentará más adelante.

Este algoritmo no es el más óptimo para la clasificación y conteo de vehículos. Debido a que clasifica los vehículos por tamaño, es decir, se podrá distinguir entre motos, coches y camiones debido a su gran diferencia de tamaño, pero para vehículos de tamaño similar no se podrán catalogar. Algunos ejemplos, como autobuses y camiones o bicicletas y motos.

Código:

- En primer lugar, se importan las bibliotecas necesarias y se especifica el video que va a ser leído.
- Se indica que algoritmo de sustracción de fondos se quiere utilizar.

```
import cv2
import numpy as np
cap = cv2.VideoCapture('stable_20211201074125_0005.mp4')
subtracao =cv2.bgsegm.createBackgroundSubtractorMOG()
# subtracao =cv2.bgsegm.createBackgroundSubtractorGMG()
# subtracao =cv2.createBackgroundSubtractorMOG2()
```

Código 5: Ejemplo de algoritmo de sustracción de fondos para clasificación de vehículos

- Se establecen los distintos parámetros que serán necesarios posteriormente.
- En primer lugar, está el error permitido entre pixeles. Se explicará con más detalle posteriormente.
- Los contadores, iniciados en cero, para el conteo, para evitar el solapamiento y que no cuente varias veces la misma moto.
- La largura y altura son las limitaciones tanto mínimas como máximas del tamaño de la moto.

```
offset = 9 #Error permitido entre pixel
#Contador de motos por ruta
motos_1_4 = 0
#Contador de frames para evitar solapamiento y que cuente varias veces COCHES
frames_cont = 0
frame_rut14 = 0
#Tamaño aproximado de una moto
largura_min_moto = 50
altura_min_moto =50
largura_max_moto=80
altura_max_moto=80
```

Código 5: Ejemplo de algoritmo de sustracción de fondos para clasificación de vehículos

- Este punto se ha explicado con los anteriores algoritmos. Para la lectura del video, el contador de frames y después se realiza una serie de procesamiento de imagen para eliminar ruidos, junto a una transformación morfológica para mejorar la imagen binaria. Estos procesos pueden variar dependiendo del video, la posición de la cámara, la iluminación…
- Finalmente, se utiliza el *cv2.findContours()* para encontrar los contornos del objeto.

```
while True:
        ret , frame1 = cap.read()
        frames_cont+=1
        frame2 = np.copy(frame1)
        grey = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(grey,(21,21),0)
        img_sub = subtracao.apply(blur)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
        dilatada = cv2.morphologyEx (img_sub, cv2. MORPH_CLOSE , kernel)
        contorno,h=cv2.findContours(dilatada,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

Código 5: Ejemplo de algoritmo de sustracción de fondos para clasificación de vehículos

- Se dibuja la línea por donde pasarán los vehículos para poder realizar el conteo. Se pueden poner tantas líneas como carreteras tenga el video para poder contar por separado o junto.

```
x_linea_ini14 = 0
x_linea_fin14 = 550
y_linea_ini14 = 620
y_linea_fin14 = 620
cv2.line(frame2,(x_linea_ini14,y_linea_ini14),(x_linea_fin14,y_linea_fin14),(255,100,0),3)
```

Código 5: Ejemplo de algoritmo de sustracción de fondos para clasificación de vehículos

- Una vez se han obtenido los contornos, se va a tratar de analizar cada uno de ellos para asegurarse que se trata de motos, coches o camiones. Aquí se explicará solo para motos.
- Se procede a analizar cada uno de los contornos contenidos en *contorno.*
- Con *cv2.boundingRect()* se obtienen los parámetros (x, y, w, h). (x,y) será el punto del rectángulo creado en la esquina superior izquierda y (w,h) es la anchura y altura.
- Posteriormente, se ponen condiciones de tamaño tanto máximas como mínimas para poder clasificar dependiendo del tamaño del vehículo si es un coche, moto o camión.
- Una vez reconocidos y clasificados, se dibuja el rectángulo de alrededor con su centro, de diferente color dependiendo que vehículo sea, y se añade a la lista dependiendo su clasificación.

```
detec_moto=[]
for(i,c) in enumerate(contorno):

    (x,y,w,h) = cv2.boundingRect(c)
    validar_contorno_moto = (w >= largura_min_moto) and (h >= altura_min_moto)
    validar_contorno1_moto = (w <= largura_max_moto) and (h <= altura_max_moto)

    if not validar_contorno_moto or validar_contorno1_moto:
        continue

    font = cv2.FONT_HERSHEY_DUPLEX
    cv2.putText(frame2, 'Moto', (x + 6, y - 6), font, 0.5, (0, 0,255), 1)
    cv2.rectangle(frame2,(x,y),(x+w,y+h),(0,0,255),2)
    centro = pega_centro(x, y, w, h)
    detec_moto.append(centro)
    cv2.circle(frame2, centro, 4, (0, 0,255), -1)
```

Código 5: Ejemplo de algoritmo de sustracción de fondos para clasificación de vehículos

- Al estar clasificados, solo queda realizar el conteo. Se suma un vehículo a su respectivo contador cada vez que los parámetros (x,y) pasen por las líneas dibujadas.

```
 # DETECCION DE MOTOS SEGUN LINEAS
for (x,y) in detec_moto:
    #CONTEO RUTA 1-4
if y<(y_linea_fin14+offset)and y>(y_linea_ini14-offset)and x>(x_linea_ini14)and x<(x_linea_fin14)
    motos_1_4+=1
    detec_moto.remove((x,y))
    frame_rut14 = frames_cont
    break
```

Código 5: Ejemplo de algoritmo de sustracción de fondos para clasificación de vehículos

- Finalmente, se representan los marcadores en el video y se finaliza el programa.

```
cv2.putText(frame2,"Coches:"+str(carros_1_4),(50,300),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
cv2.putText(frame2,"Motos:"+str(motos_1_4),(50,350),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
cv2.putText(frame2,"Camiones:"+str(camion_1_4),(50,400),cv2.FONT_HERSHEY_SIMPLEX,1,(0, 0, 255),2)
cv2.imshow("Video Original" , frame2)

if cv2.waitKey(1) == 27:
    cv2.destroyAllWindows()
    cap.release()
    break
cv2.destroyAllWindows()
cap.release()
```

Código 5: Ejemplo de algoritmo de sustracción de fondos para clasificación de vehículos

Para vehículos de diferente tamaño funciona muy bien, pero el principal problema de este algoritmo es que cuando se tratan de tamaños similares no se puede diferenciar entre ellos. Por ejemplo, los autobuses y los camiones, los coches y camionetas, las bicicletas y las motos…

Otro problema podría ser que, en algunos videos, al inicio, el vehículo parece muy pequeño y podría ser catalogado como moto, pero al ir recorriendo la carretera se acerca a la cámara y crece su tamaño. Debido a esto se tiene que cuadrar los tamaños y la posición de la línea de conteo dependiendo del propio video y de las características que tenga.

Como se aprecia justo antes de cruzar la línea de conteo cambia de reconocerse como moto a coche.

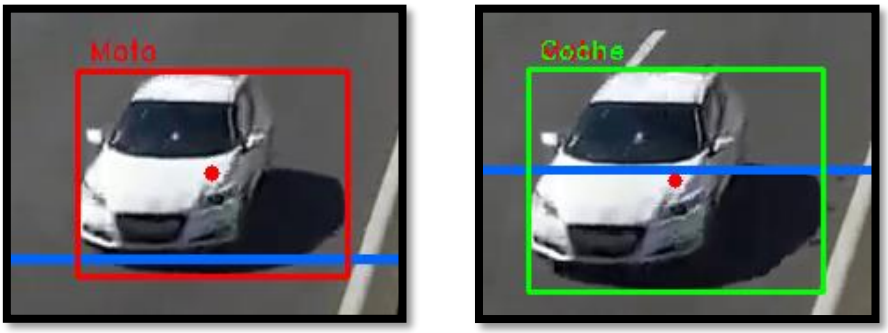


Figura 49: Clasificación de vehículos según la lejanía

Por lo tanto, se concluye que para una clasificación más general como es la de moto, coche y camión funciona de manera eficiente, pero para distinciones más detalladas no será el algoritmo más óptimo.

También realiza el conteo de manera rápida, sin necesidad de una gran memoria o GPU. Esto es importante porque el algoritmo que se verá a continuación necesitará de GPU.

Algunos ejemplos de conteo realizados se aprecian a continuación.

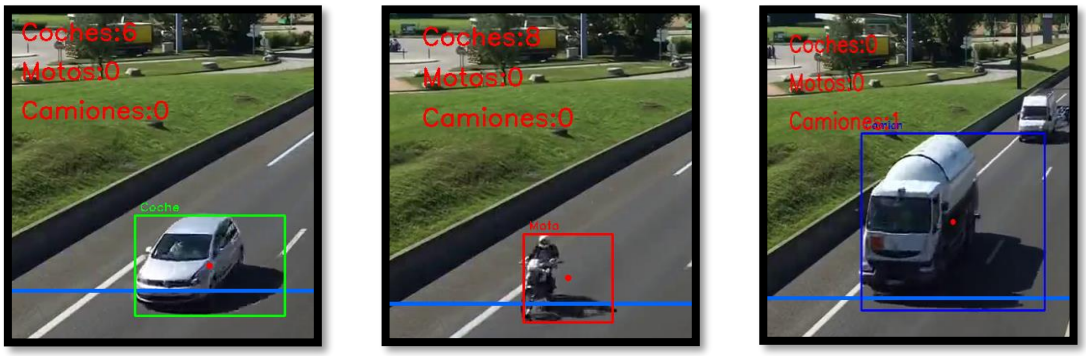


Figura 50: Clasificación de vehículos con algoritmo de sustracción

6.3 YOLO (YouOnly Look Once).

Para este apartado se explicará el último algoritmo requerido para completar el programa de detección, clasificación y conteo de vehículos de este proyecto.

YOLO apareció para dar un nuevo enfoque a los tradicionales algoritmos de detección de objetos. Es una red convolucional CNN, ya explicada anteriormente, para la detección de objetos en tiempo real. Anteriormente, esta red reutilizaba los clasificadores para conocer la forma de los objetos, pero ahora se basa en un problema de regresión con cuadros delimitadores y con propiedades de clase asociadas. De esta forma, una sola red neuronal es capaz de predecir cuadrados delimitadores y adjudicar probabilidades de clases directamente a imágenes completas a través de una sola evaluación (lo que permite ser el más rápido entre sus competidores, aunque pierde un poco de exactitud). Al tratarse de una sola red, permite su máxima optimización en cuanto a rendimiento.

La detección de YOLO es rápida, procesando alrededor de 45 frames por segundo en su primera versión. Dependiendo de cual se use tendrá de una velocidad u otra. Cabe destacar, que comete más errores en comparación a otros algoritmos a la hora de detectar la localización, pero es menos propenso a predecir falsos positivos (detectar objetos que no deberían ser detectados).

6.3.1 Funcionamiento de YOLO.

El procesamiento de imagen es simple y directo:

1. Cambia el tamaño de la imagen (dependiendo de la versión)
2. Ejecuta la red convolucional
3. Canaliza las detecciones resultantes a través de la confianza del modelo.

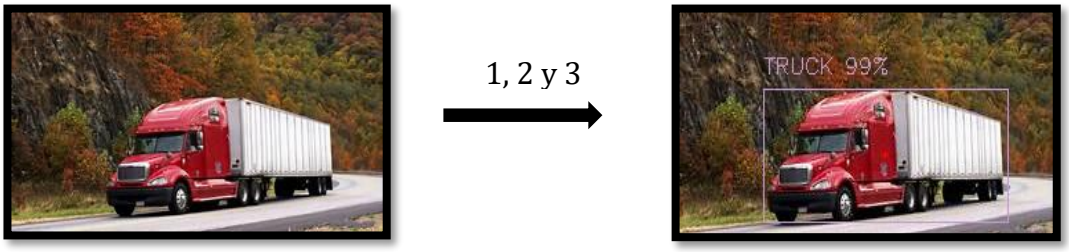


Figura 51 y 52: Camión sin clasificar y camión clasificado por YOLO.

La idea de este método es segmentar una imagen en cuadros más pequeños. La imagen se divide en una cuadrícula de SxS.

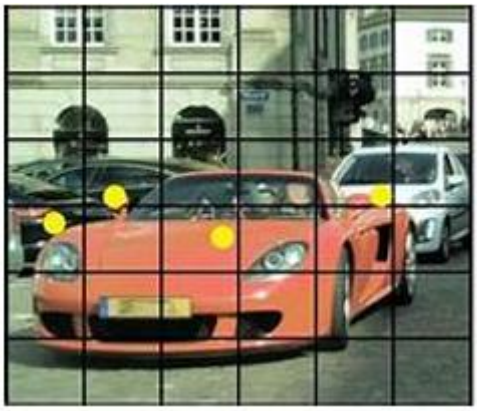


Figura 53: Cuadricula SxS

Una vez dividida la imagen, la celda que contenga el centro del objeto en cuestión es la responsable de detectar ese objeto. Cada una de las celdas predice X cuadrados delimitadores y su puntuación de confianza. La idea de esta arquitectura es que prediga dos cuadros delimitadores. El rango de clasificación irá desde 0 hasta 1 (siendo 0 el nivel de confianza más bajo y 1 el más alto). Una vez analizado, si no existe ningún objeto en la celda, la confianza será de 0 y si está completamente seguro de la predicción será 1. Estos niveles de confianza representan la certeza del modelo, asegurando que existe un objeto en esa celda y que el cuadrado delimitador es preciso. Los cuadros delimitadores constan de 5 parámetros. (x,y) son las coordenadas que representan el centro del cuadrado delimitador. (w,h) son el ancho y la altura del cuadrado, siendo fracciones relativas del tamaño total de la imagen. Por último, la confianza que representa el área cubierta por el cuadrado delimitador previsto y el cuadrado delimitador real, denominado el cuadro de verdad de suelo.

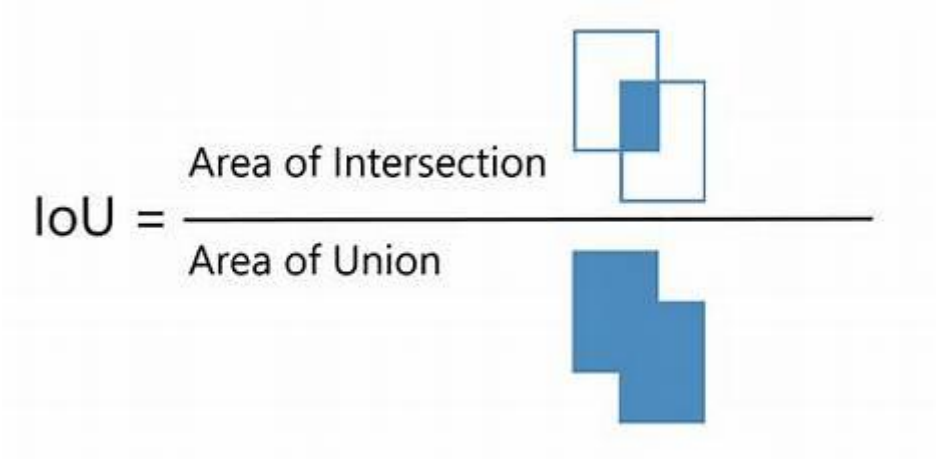


Figura 54: Cuadro de verdad del suelo.

Es el área de intersección de las cajas predichas y base, divididas por el área de unión de las mismas cajas. Es decir, será 1 si ambos cuadrados se superponen el uno al otro.

También hay que tener en cuenta que no solo genera cuadros delimitadores y asigna puntuaciones de confianza, sino que cada celda predice la clase de objeto que está detectando. Esta predicción de clase está representada por una cierta longitud de un vector, el número de clases es el conjunto de datos. Cabe destacar, que cada celda es capaz de producir infinitos cuadros delimitadores y puntuaciones de confianza, pero solo predice una clase. Esto es una limitación del algoritmo de YOLO, si se encuentran varios objetos diferentes en la misma celda, el algoritmo no clasificará correctamente la clase. Por lo tanto, cada predicción constará de la forma C+Bx5, donde C es el número de clases y B es el número de cuadrados delimitadores previstos. El 5 es por el número de parámetros de los que se forma cada casilla *(x,y,w,h,confianza).* Cada celda se forma por SxS (resolución del mapa de segmentación) cuadrículas, la predicción del modelo es un tensor de forma SxSx(C+Bx5).

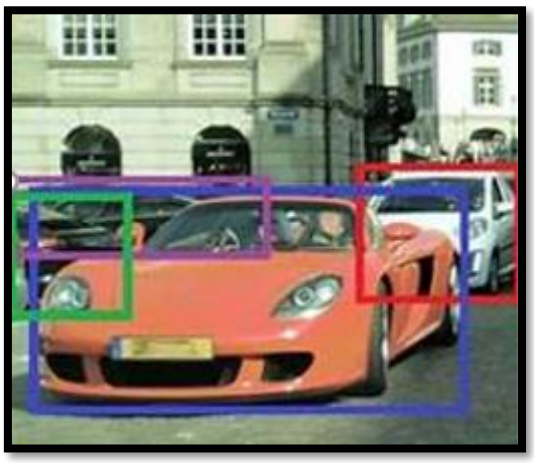


Figura 55: Ejemplo de salida del modelo

Para dar un ejemplo, el primer trabajo del algoritmo YOLO se fijó en una dimensión de 7x7, dos predictores por casilla y un número de unas veinte clases. Por lo tanto, a cada casilla le correspondieron treinta neuronas en la capa de salida.

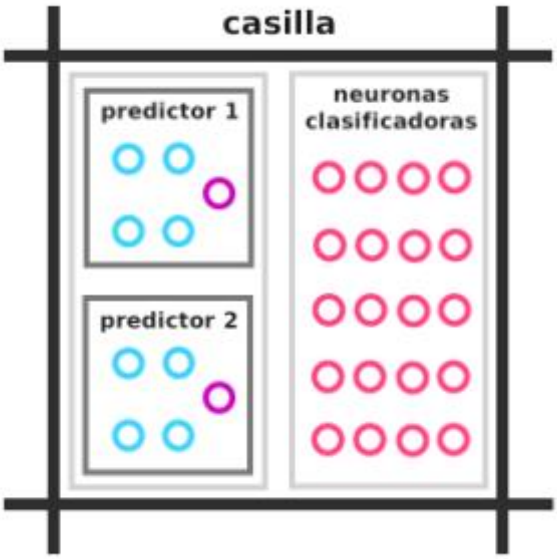


Figura 56: Ejemplo de estructura de una casilla

Como se observa en la imagen, a la izquierda se encuentran los predictores encargados de generar una ventana y a la derecha las neuronas clasificadoras. En la primera fase de entrenamiento estas ventanas tendrán un tamaño y posición aleatoria. El circulo morado que es la fiabilidad, es la probabilidad que dicha ventana realice una predicción exitosa de un objeto de la clase detectada por las neuronas selectoras. Es un parámetro elaborado por la propia red, para medir la calidad de la ventana propuesta. Conforme va aprendiendo la red, tendrá un aumento de fiabilidad en las ventanas generadas.

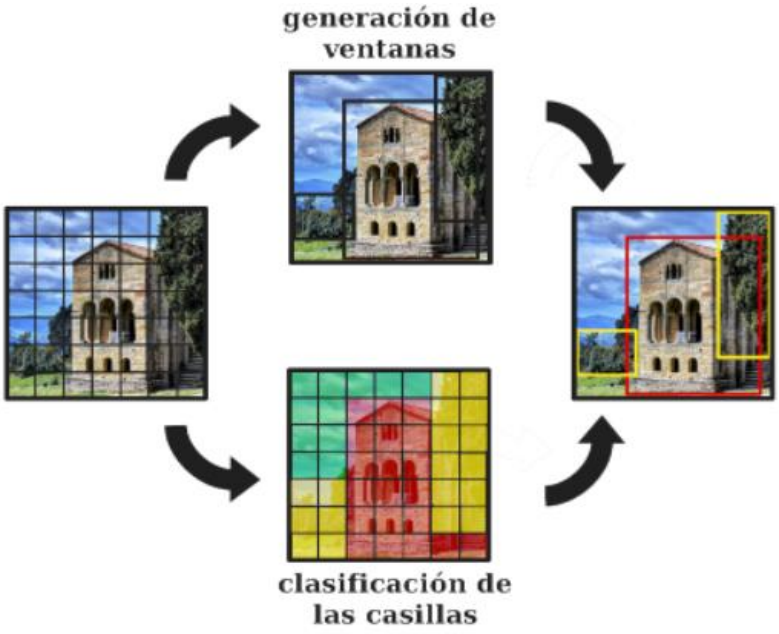


Figura 57: Generación y clasificación de casillas.

6.3.2 Arquitectura de la red.

En este apartado se va a pasar por la arquitectura por la que está compuesta YOLO. Anteriormente, se explica con detalle todo el proceso de las capas convolucionales. Consiste en 24 capas de convoluciones, intercaladas por capas de agrupación (max-pool). Todas las capas son convolucionales o de pooling, excepto las dos últimas que están conectadas. La última capa del proceso viene conformada por las neuronas selectoras y las clasificadoras.

Hay que tener en cuenta, que todo el proceso se está explicando de la primera versión de YOLO, y como se compuso. Posteriormente, se hará una breve explicación de YOLOv3.

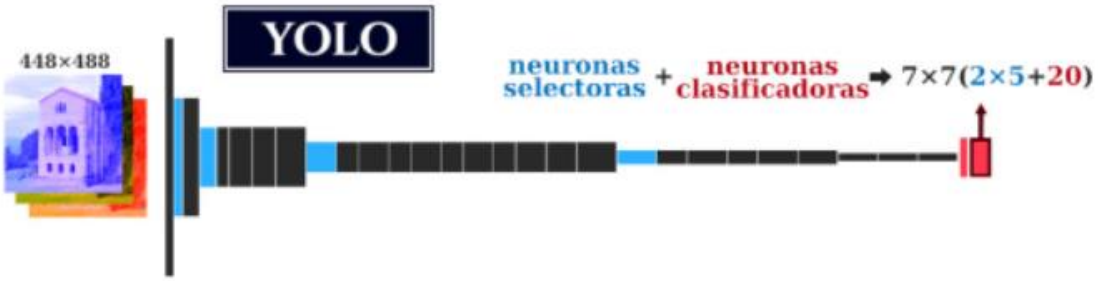


Figura 58: Arquitectura de la red

Todas las capas como se puede apreciar son de convolución (capas negras) o de pooling (capas azules). Excepto las dos últimas que están densamente conectadas (capas rojas). Como se aprecia en la imagen, se parte de una imagen de dimensiones 448x448 y tres canales de color. La salida viene definida por un tensor de 7x7x30. De estas treinta neuronas se dividen en dos. Diez de ellas corresponden a las coordenadas de las dos ventanas (neuronas selectoras) y veinte a las puntuaciones de cada una de las clases (neuronas clasificadoras).

6.3.3 YOLOv3.

Como se ha comentado, se ha explicado un poco más en detalle el método YOLO. Para el presente trabajo se va a utilizar una versión algo más moderna, YOLOv3.

En esta tercera versión, se presenta una arquitectura nueva y más profunda de extractores, llamada Darknet-53. Es una versión híbrida entre la versión anterior conocida como Darknet-19 y una red residual. Su nombre viene dado por sus capas convolucionales que serán 53 en vez de 19 como su anterior versión. Es una red totalmente convolucional (FCN), es decir, no tiene agrupación máxima. Para cada convolución sigue una normalización por lotes. Es un método para que las redes neuronales artificiales sean más rápidas y estables a través de la normalización de las entradas de las capas al volver a centrar y escalar. En otras versiones no se tenía la normalización por lotes y se usaban las capas de max-pooling.

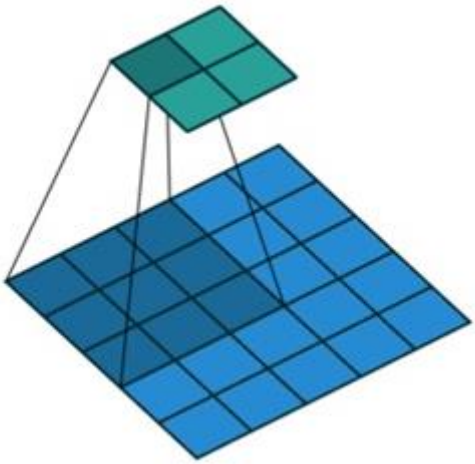


Figura 59: Circunvoluciones estriadas

Como la agrupación máxima no funciona especialmente bien y se necesita reducir la muestra de nuestros mapas de filtro, se utilizan circunvoluciones estriadas que reducen la entrada a la mitad. En este caso, si se observa la arquitectura, las capas convolucionales son seguidas por una conexión o red residual que ayudan a evitar el sobreajuste.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 60: Estructura de la red Darkent-53

Esta nueva versión es mucho más potente que la versión anterior, y sigue siendo más eficiente que las rede troncales (ResNet-101 o ResNet-152).

Backbone	Top-1	Top-5	Ops	BFLOP/s	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Figura 60: Comparación entre diferentes redes.

- **Top-1 y Top-5:** es la precisión en los clasificadores top-1 o top-5.
- **Ops:** son miles de millones de operaciones requeridas en el momento previsto.
- **BFLOP/s:** son miles de millones de operaciones de punto flotantes requeridas por segundos.
- **FPS:** fotogramas por segundo.

Para resumir, las dos primeras columnas denotan precisión y el resto velocidad. Lo que se busca es lógico, más precisión y velocidad. Cuanto menor sea el número de miles de millones de operaciones (Ops), más rápido podrá ejecutar. Cuantos más BFLOP/s y FPS, mejor.

La estructura DarkNet-19 tiene una precisión muy elevada y con los Ops pequeños, y por eso resulto unos FPS fueran muy elevados. La ResNet-152 tiene un buen rendimiento, pero tuvo más Ops y menos FPS. El objetivo final de esta nueva estructura (DrarkNet-53) era que funcionase tan bien como ResNet-152, pero con una velocidad mayor. Como se puede observar, hicieron un trabajo excelente al tener un rendimiento muy elevado con menos Ops y más FPS. Es más lento que YOLOv2 pero funciona mucho mejor.

A continuación, se explica la representación del cuadro delimitador o el BoundingBox, usando dimensiones como cuadros de anclaje para hacer predicciones de cuadros delimitadores. La red predice cuatro coordenadas para cada BoundingBox (tx, ty, tw y th). Si la celda se encuentra desplazada desde la esquina superior izquierda por (cx, cy) y el Bounding Box tiene unas dimensiones de (pw, ph) resulta una predicción como:

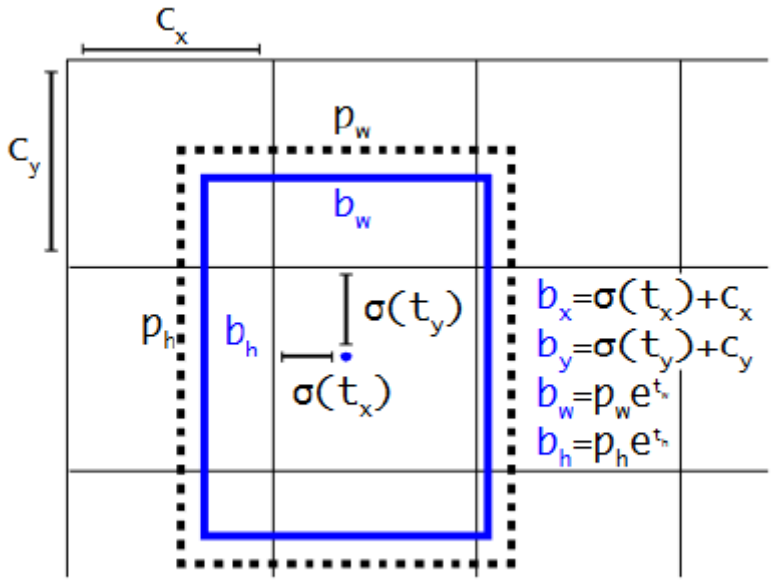


Figura 59: Predicción Bounding Box para YOLOv3

YOLOv3 realiza una predicción objetiva para cada Bounding Box usando regresión logística. Es decir, el cuadro delimitador es una pequeña modificación en el cuadro ancla. En principio se probó la predicción directa de las coordenadas del cuadro, pero no funcionó de manera correcta.

Para representar estas cajas de anclaje se usa el conjunto de datos COCO, donde se observan todos los cuadros delimitadores en los datos de entrenamiento y se empezaron a agrupar usando el agrupamiento K-means. K=5, quiere decir que se tendrán cinco cajas de anclaje, donde se tendrá una probabilidad muy alta de encontrar el objeto buscado. Se puede pensar en ellos como antes de sus cuadros delimitadores. Le indica la región suficientemente buena para poder mirar, al encontrarse la mayoría de los cuadros delimitadores en sus datos de entrenamiento. Al usar esta información para codificar todos los cuadros se produce menos posibilidad de error, mejorando el rendimiento en un 3-4%.

Cabe mencionar, que COCO es un conjunto de datos de subtítulos, segmentación y detección de objetos a gran escala. Los objetos se etiquetan usando segmentación por instancias para ayudar a la localización precisa de objetos. Tiene un total de 2.5 millones de instancias etiquetadas en 328.000 imágenes.

Una vez que se ha visto el funcionamiento de las cajas, se pasa a la puntuación de objetividad que comprueba si hay algún objeto en el cuadro representado. Intenta encontrar una probabilidad de que el objeto esté en el interior de la caja. Esto se realiza con clasificación binaria simple. Como se ha comentado, para cada cuadro y probabilidad se tendrá un modelo de regresión logístico. Si se desea encontrar la probabilidad de que esta caja contenga un objeto, se obtiene multiplicando la puntuación de probabilidad de objetividad y la probabilidad de clase.

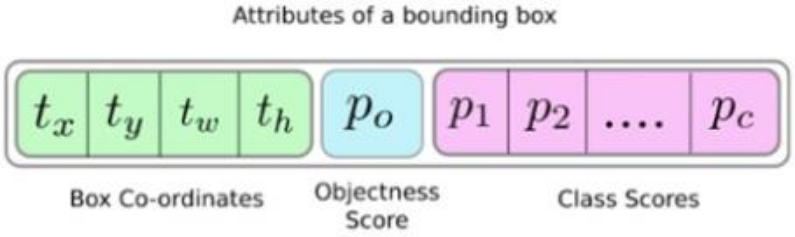


Figura 60: Atributos del cuadro delimitador

Por lo tanto, para la versión tres se construyó una regresión logística para cada una de las clases. En resumen, para un cuadro delimitador, tiene cuatro coordenadas de cuadro, una regresión logística para el puntaje de objetividad y 80 regresiones logísticas para cada una de las clases. Debido a esto surgen los sigmoides por clase. Para poder etiquetar con varias clases algunos cuadrados.

6.3.4 Código utilizado en YOLO para detección, clasificación y conteo de coche.

Una vez explicado el funcionamiento de YOLO se pasa a explicar los parámetros más importantes para su programación sin incluir el código final total, es decir, el funcionamiento principal del algoritmo como se ha hecho hasta ahora.

Para la detección de objetos en YOLO, se puede programar para hacerlo en tiempo real o solamente procesando una imagen. En este apartado se va a explicar lo más relevante del código en general, ya que, el proceso que lleva a cabo para la clasificación es el mismo, tanto para una sola imagen o en tiempo real. Al mismo tiempo, se explicará el programa final utilizado, que es un algoritmo híbrido entre análisis de movimiento y YOLO, para que procese los videos en el menor tiempo posible. Debido a esto, se pondrá parte del programa sin explicar profundamente el análisis de movimiento al haberlo tratado con anterioridad.

A pesar de la rápida detección y reconocimiento de YOLO, al procesar un video utilizando el reconocimiento de objetos en todo momento hace que la lectura del video sea muy lenta. Es por esto que se han unido ambos programas para poder analizar el objeto solo en un determinado momento. De esta forma se ahorra mucho tiempo, porque el programa no procesa el reconocimiento de objetos hasta que el propio vehículo pasa por un determinado lugar establecido, obteniendo una imagen concreta en ese momento para poder clasificarla sin necesidad de hacerlo continuamente.

Código:

- En primer lugar, se debe descargar tres archivos para el funcionamiento del programa.
  - **Archivo de peso:** es el modelo entrenado, algoritmo para detección de objetos.

- **Archivo cfg:** archivo de configuración, todas las configuraciones del algoritmo YOLO.
- **Coco.names:** conjunto de 80 nombres de objetos.

❖ Hay varias versiones del modelo YOLOv3. En este apartado se usará YOLOv3-320 para la detección de objetos. 320 se refiere al tamaño de las imágenes en las que se entrena el modelo YOLO. Si se utiliza una resolución más baja los frames por segundo serán más rápidos.

Tamaño de imagen	Nombre del modelo	Frames/segundo (FPS)
(320,320)	YOLOv3-320	45
(416,416)	YOLOv3-416	35
(608,608)	YOLOv3-608	20
N/A	YOLOv3-pequeño	220

Tabla 2: Versiones de YOLOv3

Como se observa, en el modelo YOLOv3-tiny la velocidad de fotogramas es muy elevada, pero penaliza su precisión, es decir, tendrá muchas menos detecciones.

❖ Al principio del código se ponen las bibliotecas y las diferentes funciones que se explicarán cuando sean nombradas en el programa.

```
import cv2
import numpy as np
import collections
```

Código 6: Ejemplo de algoritmo de sustracción de fondos combinado con YOLO

❖ Se ponen los contadores a 0 de las diferentes rutas que se tenga, aunque en este código solo se explicará para una.
❖ *confThreshold* y *nmsThreshold* son el límite mínimo de puntuación de la confianza al detectar un objeto y el umbral de Non-max supresion. Los siguientes comandos son el color, tamaño y espesor de letra.

```
ruta=[0,0,0,0,0]
ruta14_conteo=[0,0,0,0,0]

confThreshold =0.1
nmsThreshold= 0.1

font_color = (0, 0, 255)
font_size = 0.5
font_thickness = 2
```

Código 6: Ejemplo de algoritmo de sustracción de fondos combinado con YOLO

- Se lee el archivo de datos COCO que contiene todos los nombres de clases y se almacenan en una lista. *Required_class_index* contiene el índice de las clases que son moto, bicicleta, camión, autobús y coche.
- Se cargan el resto de archivos del modelo YOLO y se configura la red mediante *cv2.dnn.readNetFromDarknet*().
- Usando la función *np.random.randint*() se genera un color aleatorio para los rectángulos de cada clase.
- Finalmente, se da el tamaño de entrada. Se pone 320 porque estamos usando el modelo YOLOv3-320 y será con el mejor tamaño que funcione.

```
# Store Coco Names in a list
classesFile = "coco.names"
classNames = open(classesFile).read().strip().split('\n')
# class index for our required detection classes
required_class_index = [1, 2, 3, 5, 7]
detected_classNames = []
frequency = []
# Model Files
modelConfiguration = 'yolov3-320.cfg'
modelWeigheights = 'yolov3-320.weights'
# configure the network model
net = cv2.dnn.readNetFromDarknet(modelConfiguration, modelWeigheights)

# Configure the network backend

net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

# Define random colour for each class
np.random.seed(42)
colors = np.random.randint(0, 255, size=(len(classNames), 3), dtype='uint8')
input_size = 320
```

Código 6: Ejemplo de algoritmo de sustracción de fondos combinado con YOLO

- Esto es idéntico al programa de análisis de movimiento. Se realiza la lectura del video, al procesado de imagen y se dibujan las líneas.

```
while True:
ret , frame1 = cap.read()
frames_cont+=1
frame2 = np.copy(frame1)
grey = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(grey,(21,21),0)
img_sub = subtracao.apply(blur)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
dilatada = cv2.morphologyEx (img_sub, cv2. MORPH_CLOSE , kernel)
contorno,h=cv2.findContours(dilatada,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
x_linea_ini14 = 0
x_linea_fin14 = 350
y_linea_ini14 = 620
y_linea_fin14 = 620
cv2.line(frame2,(x_linea_ini14,y_linea_ini14),(x_linea_fin14,y_linea_fin14),(255,100,0),3)
```

Código 6: Ejemplo de algoritmo de sustracción de fondos combinado con YOLO

- Después se realiza un proceso casi idéntico que en el análisis de movimiento, pero solo con una lista, sin diferenciar los tipos de vehículos que pasan.

```
detec=[]
for(i,c) in enumerate(contorno):
    (x,y,w,h) = cv2.boundingRect(c)

    font = cv2.FONT_HERSHEY_DUPLEX
    cv2.putText(frame2, 'vehiculo', (x + 6, y - 6), font, 0.5, (0, 0,255), 1)
    cv2.rectangle(frame2,(x,y),(x+w,y+h),(0,0,255),2)
    centro = pega_centro(x, y, w, h)
    detec.append(centro)
    cv2.circle(frame2, centro, 4, (0, 0,255), -1)
```

Código 6: Ejemplo de algoritmo de sustracción de fondos combinado con YOLO

- A continuación, es donde viene la gran diferencia. En principio el programa es casi idéntico al explicado con anterioridad, pero aún más simple, es decir, sin ni siquiera diferenciar entre tamaños. Esto es debido a que solo se busca detectar el vehículo moviéndose. Cuando pasa por la línea establecida se llama a la función *from_statec_image*() que es donde entra en juego YOLO. Cuando pasa el vehículo por la línea, se manda justo esa imagen recortada del vehículo a la función, donde detecta y clasifica el vehículo que ha pasado.
- La función *clasificación_yolo*() es para realizar el conteo por diferentes rutas.

```
for (x,y) in detec:
    #CONTEO RUTA 1-4
if y<(y_linea_fin14+offset)and y>(y_linea_ini14-offset)and x>(x_linea_ini14)and x<(x_linea_fin14):
        detec.remove((x,y))
        frame_rut14 = frames_cont
        frame3=frame1[y-300:y+300, x-170:x+120]
        tipo,frame3,frequency = from_static_image(frame3)
        clasificación_yolo(ruta14_conteo,tipo)
        break
```

Código 6: Ejemplo de algoritmo de sustracción de fondos combinado con YOLO

- Se van a explicar las funciones que utiliza YOLO para reconocer el objeto.
- En esta función se introduce una imagen a la red de detección. Ahora no se puede proporcionar una imagen simple directamente a la red. La red de detección de objetos solo procesa un tipo particular de formato denominado blob. Se tiene que convertir la imagen de entrada en blob, solo de esta manera se puede introducir a la red con la función *cv2.dnn.blobFromImage*(). Con *net.setInput*() se envían datos de imagen de blob transformados a red YOLO.
- Se producen tres capas de salida. Esto quiere decir que se tienen tres valores diferentes que salen de la red YOLO. Para conocer los resultados de estas capas, se necesita saber los nombres de las capas. Ahora, con *Net.forward*() se envían datos de imágenes de blob a esas tres capas de salida.
- Es un vector con 85 columnas donde las 5 primeras son:
    1. Valor x del punto central del cuadro delimitador de un objeto.
    2. Valor y del punto central del cuadro delimitador de un objeto.
    3. Ancho del cuadro delimitador.
    4. Altura del mismo.
    5. Valor de confianza.
- Por último, se llama a la función *postProcess*() que se explicará a continuación. *Frequency* es una manera de contar cuantos vehículos de cada clase se han procesado. Crea una lista con las detecciones realizadas y cuenta el número de vehículos que se han repetido en la lista de cada clase.

```
def from_static_image(image):
img=image
blob = cv2.dnn.blobFromImage(img, 1 / 255, (input_size, input_size), [0, 0, 0], 1, crop=False)
# Set the input of the network
net.setInput(blob) #se envian datos de la imagen convertidos en yolo
layersNames = net.getLayerNames() #extrae todas las capas
outputNames = [(layersNames[i - 1]) for i in net.getUnconnectedOutLayers()]
outputs = net.forward(outputNames) # enviamos datos de la imagen blob a las tres capas
postProcess(outputs,img) # se llama a la función
# count the frequency of detected classes
frequency = collections.Counter(detected_classNames)

return detected_classNames[-1],img,frequency
```

Código 6: Ejemplo de algoritmo de sustracción de fondos combinado con YOLO

- En primer lugar, se define una lista vacía donde se almacenan las clases detectadas en un marco.
- Se extrae el valor de la puntuación, el ID de cada objeto y la confianza para cada ID.
- El bucle de la confianza es porque cada cuadro delimitador tiene una puntuación de confianza, entonces se seleccionan los cuadros delimitadores que superen el valor límite asignado con *confThreshold.*
- Después, se extraen valores de cuadros delimitadores seleccionados para objetos con una precisión establecida.
- Una vez se tienen muchas opciones de cuadros, algunas de ellas se superpondrán. Aparecerán varios cuadros para el mismo objeto por lo que se debe reducir la cantidad, tomando el mejor cuadro de detección para cada clase. Esto es la supresión no max con la función *cv2.dnn.NMSBoxes*().
- Para concluir con la función, una vez elegido el cuadro delimitador se representa.

```
def postProcess(outputs,img):

    height, width = img.shape[:2]
    boxes = []
    classIds = []
    confidence_scores = []
    detection = []
    for output in outputs:
        for det in output:
            scores = det[5:]
            classId = np.argmax(scores)
            confidence = scores[classId]
            if classId in required_class_index:
                if confidence > confThreshold:
                    w,h = int(det[2]*width) , int(det[3]*height)
                    x,y = int((det[0]*width)-w/2) , int((det[1]*height)-h/2)
                    boxes.append([x,y,w,h])
                    classIds.append(classId)
                    confidence_scores.append(float(confidence))
    # Apply Non-Max Suppression
    indices = cv2.dnn.NMSBoxes(boxes, confidence_scores, confThreshold, nmsThreshold)
    if len(indices) > 0:
        for i in indices.flatten():
            x, y, w, h = boxes[i][0], boxes[i][1], boxes[i][2], boxes[i][3]
            color = [int(c) for c in colors[classIds[i]]]
            name = classNames[classIds[i]]
            detected_classNames.append(name)
            # Draw classname and confidence score
            cv2.putText(img,f'{name.upper()} {int(confidence_scores[i]*100)}%',
                        (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
            # print (int(confidence_scores[i]*100))
            # Draw bounding rectangle
            prob=int(confidence_scores[i]*100)
            cv2.rectangle(img, (x, y), (x + w, y + h), color, 1)
            detection.append([x, y, w, h, required_class_index.index(classIds[i])])
```

Código 6: Ejemplo de algoritmo de sustracción de fondos combinado con YOLO

6.3.5 Resultados

Una vez que se ejecuta el programa, se reproduce el video analizando el movimiento con la sustracción de imagen. Se procesa el video para reducir los ruidos y analizar los vehículos los más claro posible. A diferencia del anterior algoritmo en cualquier momento se detecta como vehículo sin necesidad de calificar por tamaños.

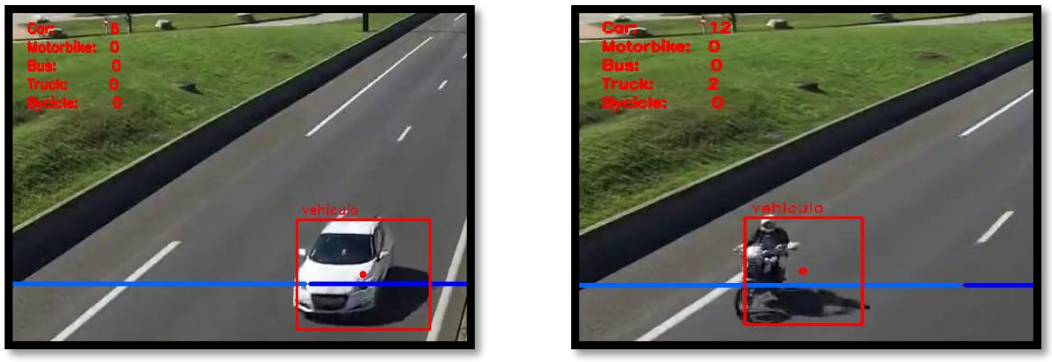


Figura 61 y 62: Detección de vehículos por sustracción de imagen

Una vez que pasa el centro del rectángulo por la línea se manda esa imagen a las funciones explicadas con anterioridad. Una vez sea procesado, se devuelve la imagen recortada con su recuadro, la clasificación y su confianza. Una vez se clasifica se añade a la lista y se suma al contador.

Figura 63 y 64: Detección y clasificación con YOLO

El principal problema de este método es limpiar los ruidos, ya que, al recorta la imagen si encuentra algún vehículo dentro de ese rango lo detectará y lo contará. De este modo se tendrá que ajustar los recortes para cada posición de cámara diferente.

A continuación, se muestran varios ejemplos en diferentes intersecciones para ver su validación.

- **Primera intersección**

En primer lugar, se muestra una intersección con cinco direcciones diferentes. En la primera imagen se muestra la posición de las líneas para la detección de vehículos cuando pasan sobre ellas. En las imágenes posteriores se ilustra cuando un coche pasa por una de las líneas e instantáneamente se realiza una captura de pantalla, que se manda al proceso de YOLO para su clasificación. Se muestra una ventana en la parte superior con una ampliación del coche que cruza la línea con su respectivo recuadro y clasificación hecha. En la segunda imagen aparece un coche y en la segunda una moto clasificadas.

Figura 65: Resultados primera intersección YOLO

Figura 66: Resultados primera intersección YOLO

Figura 67: Resultados primera intersección YOLO

- **Segunda intersección**

Para continuar con la validación, se realizan diferentes pruebas con otra intersección diferente. Se posicionan las líneas de detección en sus respectivos carriles, los cuales serán tres en este caso. Como en las imágenes anteriores se muestra una ventana en la parte superior con el aumento de imagen y su clasificación correspondiente. En la primera aparece un camión y en la segunda un coche clasificados correctamente.

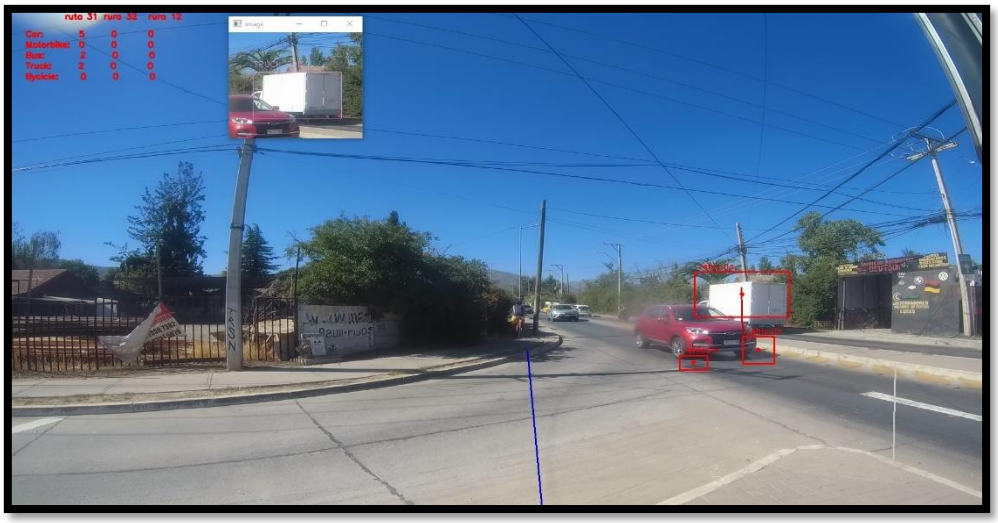


Figura 68: Resultados segunda intersección YOLO

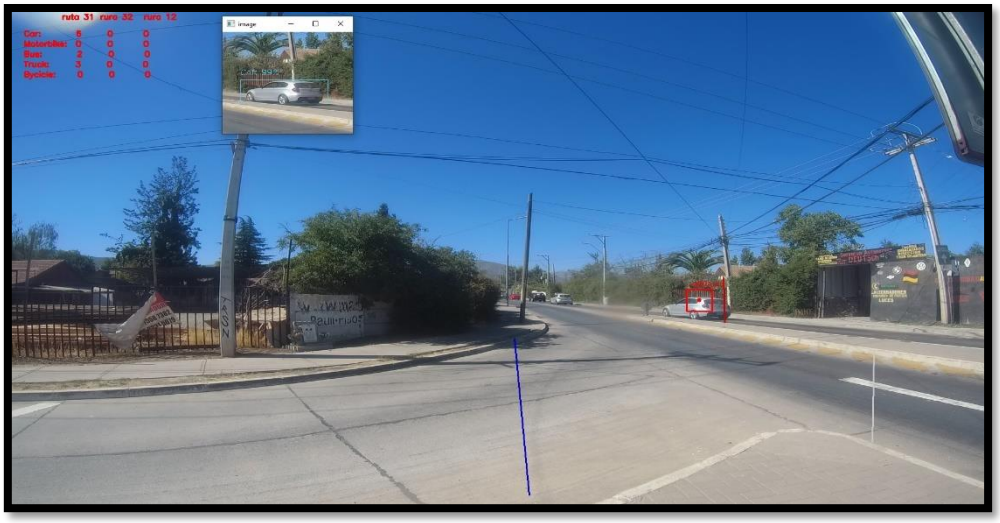


Figura 69: Resultados segunda intersección YOLO

- **Tercera intersección**

Por último, se muestra una tercera intersección que también dispone de cinco carriles. En la primera imagen se muestra un recorte del coche mucho más cercano que el resto. Esto ocurre cuando no se mide adecuadamente la anchura y altura del vehículo, pero aun estando recortada con demasiada dimensión se aprecia como el funcionamiento sigue siendo correcto. Detecta en ambas imágenes dos coches como clasificación, añadiéndolos al marcador en su respectivo carril.

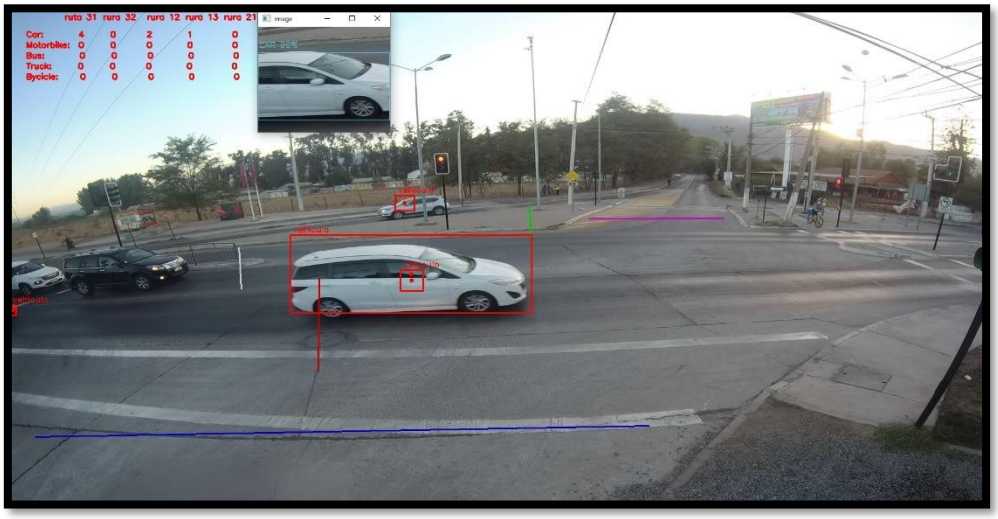


Figura 70: Resultados tercera intersección YOLO

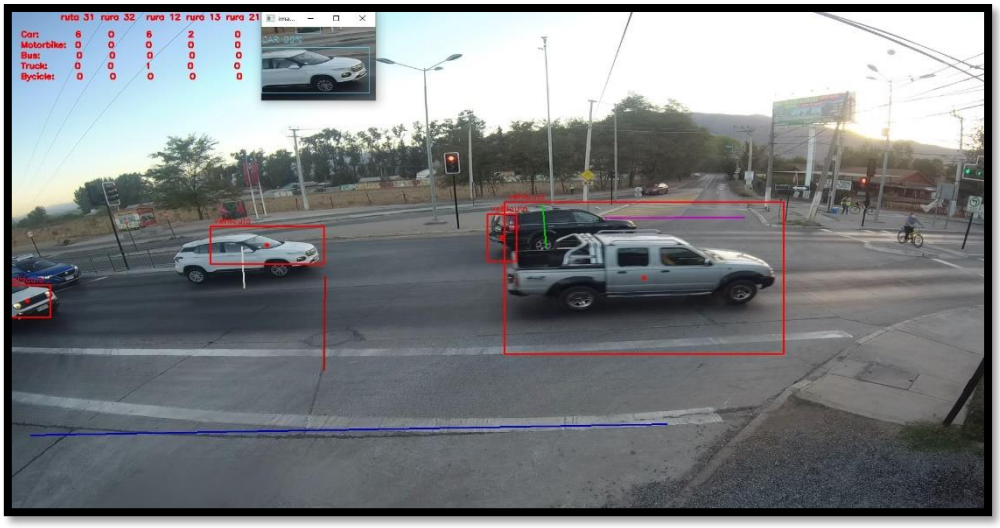


Figura 71: Resultados tercera intersección YOLO

7. Entrenamiento y detección de vehículos aeroportuarios.

Una vez ya se ha dispuesto del algoritmo para la clasificación y conteo de vehículos, se va a proceder a entrenar la red neuronal para el reconocimiento de vehículos aeroportuarios. Se va a explicar los pasos que se van a llevar a cabo para conseguir detectar los vehículos, para conocer su ubicación en tiempo real y la demanda necesaria para los aeropuertos. Concretamente se va a realizar el entrenamiento de la red para específicamente dos vehículos aeroportuarios.

- **Tractor push-back:** los aviones no disponen de marcha atrás. Estos vehículos se encargan de remolcar los aviones. Se encuentran varios tamaños del vehículo para los diferentes tamaños de aviones. Se componen de sistemas hidráulicos para levantar el tren de aterrizaje principal para conseguir mayor maniobrabilidad y rapidez.

Figura 72: Tractor push-back

- **Cabeza tractora:** estos vehículos se utilizan para el transporte de cargas en las pistas de los aeropuertos. Son capaces de tirar de remolques empleados para transportar el equipaje de los pasajeros en un entorno complicado como en las pistas de un aeropuerto.

Figura 73: Cabeza tractora

En un aeropuerto se encuentran mucha variedad de vehículos, como camiones que proporcionan electricidad a los aviones cuando se encuentran lejos de la terminal, camiones de repostaje, camiones con escaleras para bajar del avión… En este trabajo solo se entrenará la red para los vehículos explicados anteriormente. No se realizará para más vehículos porque supone una gran cantidad de imágenes, un gran procesador y mucho tiempo de entrenamiento.

7.2 Conjunto de imágenes.

Para poder detectar los vehículos, se necesita un conjunto de datos de imágenes donde se encuentran el objeto personalizado que se desea detectar. El proceso seguido se explicará para un solo vehículo, ya que, el proceso es el mismo para diferentes tipos.

Por lo tanto, se tienen que conseguir un número elevado de imágenes. En este caso se va a realizar un conjunto de unas cien imágenes por automóvil. Cuantas más imágenes se tengan, más probabilidad de detectar y reconocer los objetos. Para hacerse una idea de cuantas imágenes se necesitan para crear una probabilidad muy elevada y que se pueda distinguir el objeto incluso en malas condiciones de imágenes o videos, la clasificación de vehículos utilizada por YOLOv3-320 tiene alrededor de 180.000 imágenes introducidas y recortadas para enfocar el objeto y poder obtener sus datos.

Como ya se ha comentado, no solo es necesario obtener las imágenes, sino que también se debe encuadrar donde está el objeto en la imagen. Para realizar esta tarea se ha utilizado un software externo denominado Labeling.

7.2.1 Configuración de imágenes en Labeling.

En primer lugar, se descarga la última versión de software para Windows. Una vez se ejecute, se hace presionar el botón de “Open Dir”.

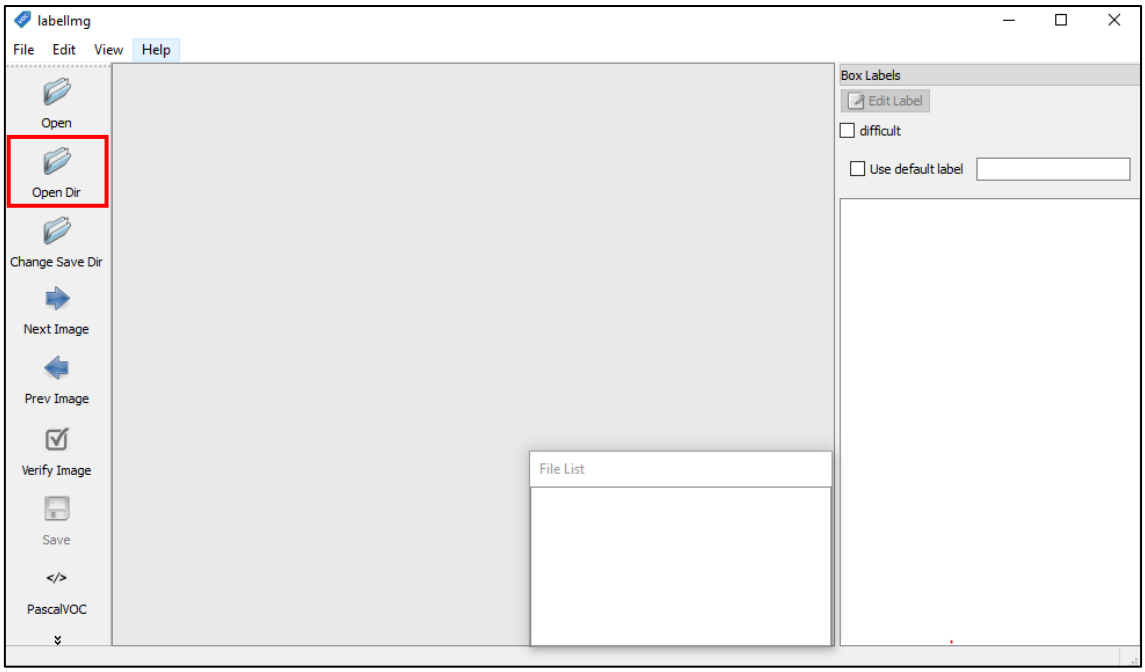


Figura 74: Página inicial Labeling

Se escoge la carpeta donde se han guardado todas las fotos y se selecciona. También se debe cambiar la configuración para poder procesarlo en YOLO. Se cambia "PascalVOC" por "YOLO".

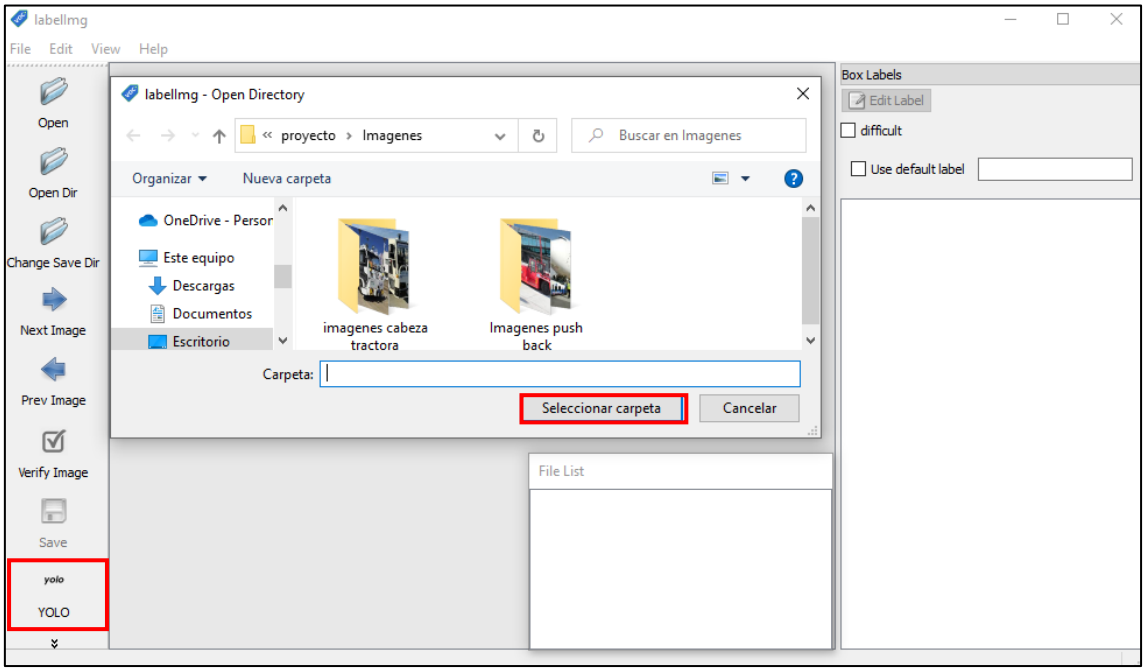


Figura 75: Selección de carpetas

Una vez seleccionadas se incluyen todas en el programa y aparecen por pantalla.

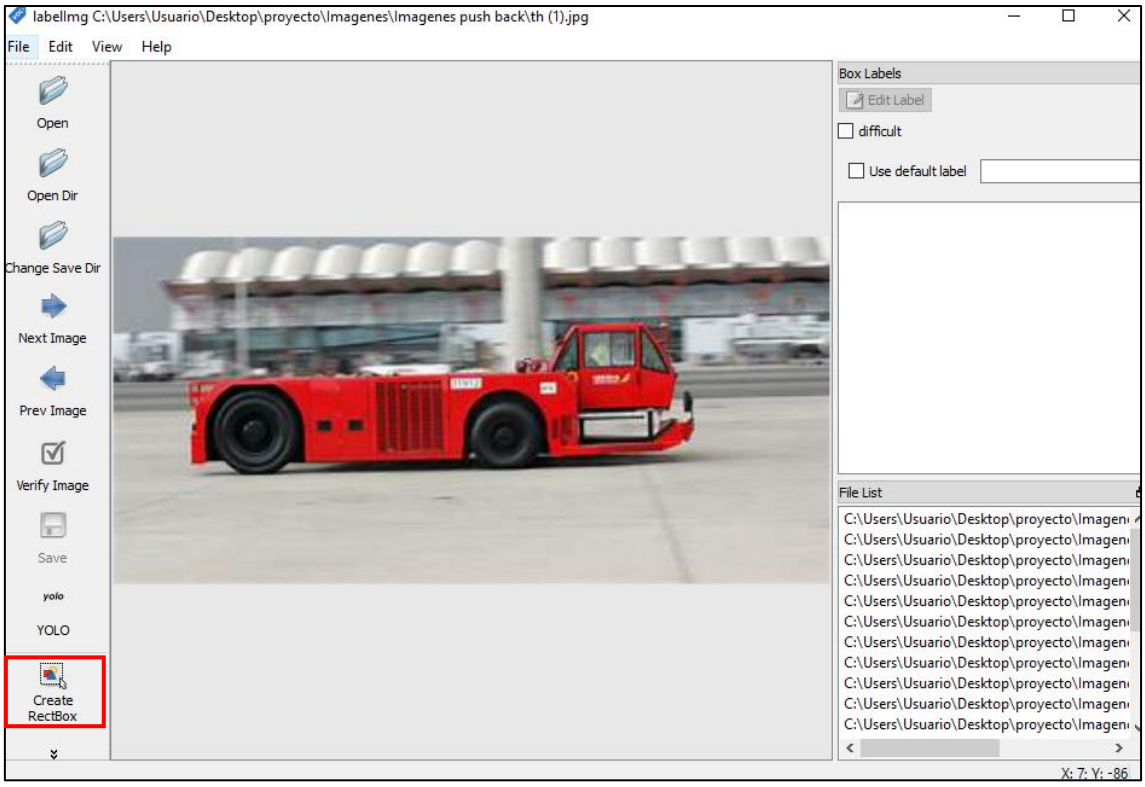


Figura 76: Imágenes seleccionadas

Ahora se pretende especificar en la imagen donde está situado el objeto. Para ello, se da al botón "Crear RectBox". Se selecciona el área donde se encuentre específicamente el vehículo, se añade la etiqueta con el nombre deseado y se guarda. Esta operación se debe hacer para cada imagen del conjunto de datos.

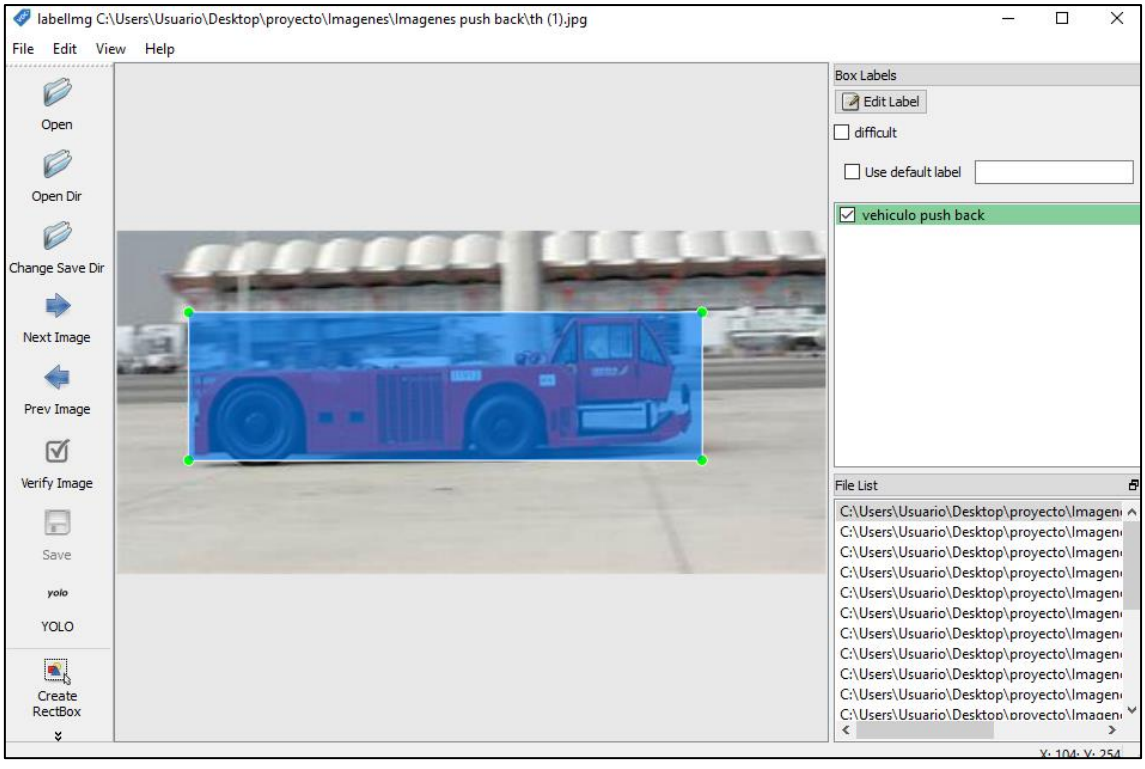


Figura 77: Vehículo recuadrado

Este proceso es largo y pesado si se quiere hacer para muchas imágenes. Para este vehículo en concreto se debe de hacer así, pero hay páginas que se encuentran muchos objetos ya recuadrados, donde se puede descargar un paquete de imágenes con los parámetros necesarios para poder introducirlo a su entrenamiento.

Finalmente, cada imagen se guarda con un archivo .txt donde se encuentran los parámetros acotados de la imagen. Se guardan todas las imágenes y archivos en un archivo .zip y se sube a Google drive para continuar con el entrenamiento. Es necesario subirlo porque se usará Google Colab para entrenar el conjunto.

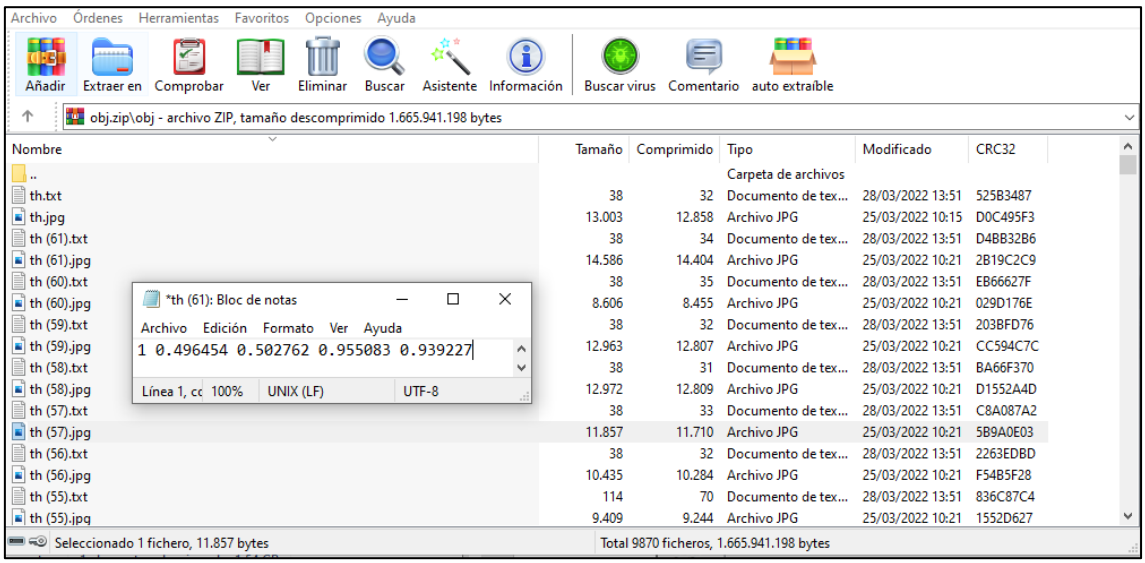


Figura 78: Imágenes y archivos txt

Google Colab es un servicio que ofrece Google para poder ejecutar scripts de Python y poder usar bibliotecas de aprendizaje automático aprovechando un hardware muy potente. Se utiliza este servicio porque el proceso de entrenamiento es muy largo y necesita de un gran procesador. Aun usando este servicio la iteración para el entrenamiento puede llevar horas, dependiendo de la cantidad de fotos introducidas en el conjunto. Google Drive se utiliza porque Colab deja de funcionar al cabo de 12 horas y borra todo el contenido. Si el entrenamiento dura más se guardarían los avances y de esta forma no se perdería.

7.2.2 Entrenamiento

Como se ha comentado, se pasa a entrenar la red en Colab que permite ejecutar y programar en Python con las siguientes ventajas:

- No requiere configuración.
- Da acceso gratuito a GPUs.
- Permite compartir contenido fácilmente.

A continuación, se va a explicar paso a paso como utilizar Colab para realizar el entrenamiento deseado. Para poder llevar a cabo el entrenamiento se usará

YOLOv3. Esta red se realiza con "YOLO Pipeline", el cual es un código de programación para entrenarlo, que se detallará paso a paso posteriormente. Para empezar a trabajar en Colab se debe crear un cuaderno nuevo.

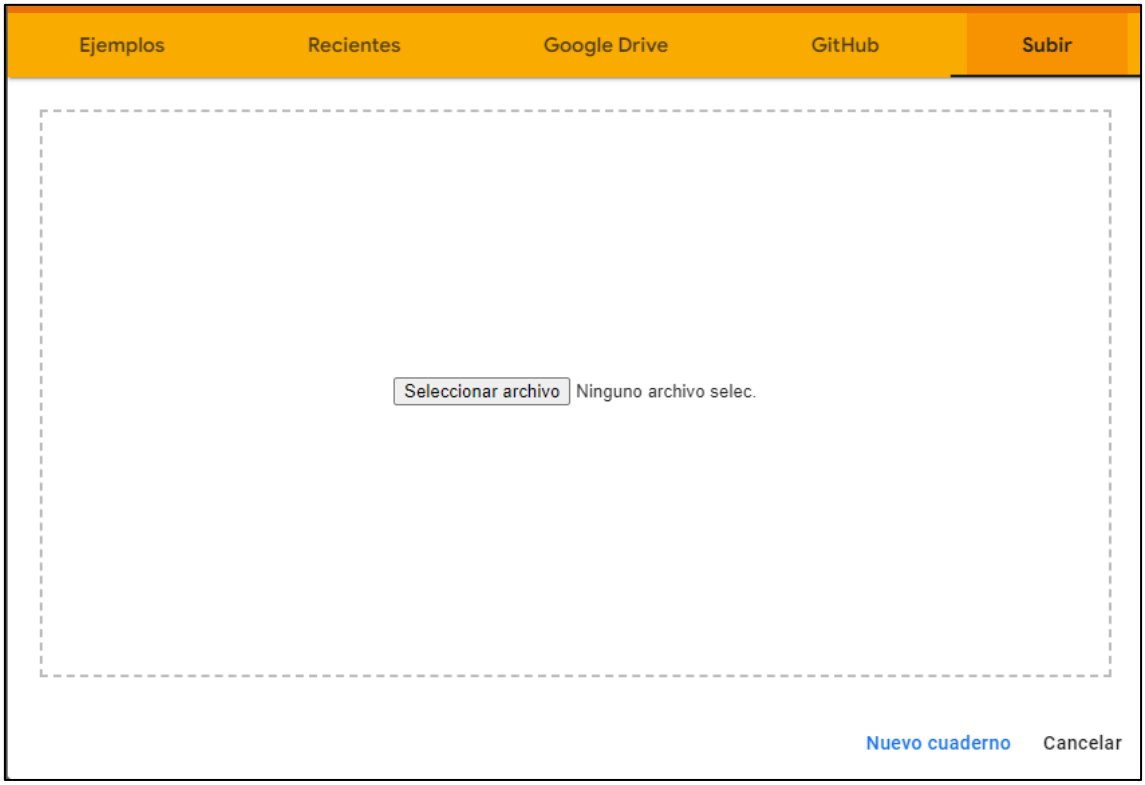


Figura 79: Nuevo cuaderno

En este cuaderno se introducirá el Pipeline de YOLOv3, el cual se detallará paso a paso.

- En primer lugar, en el menú de edición se cambia la configuración del portátil de hardware a GPU.

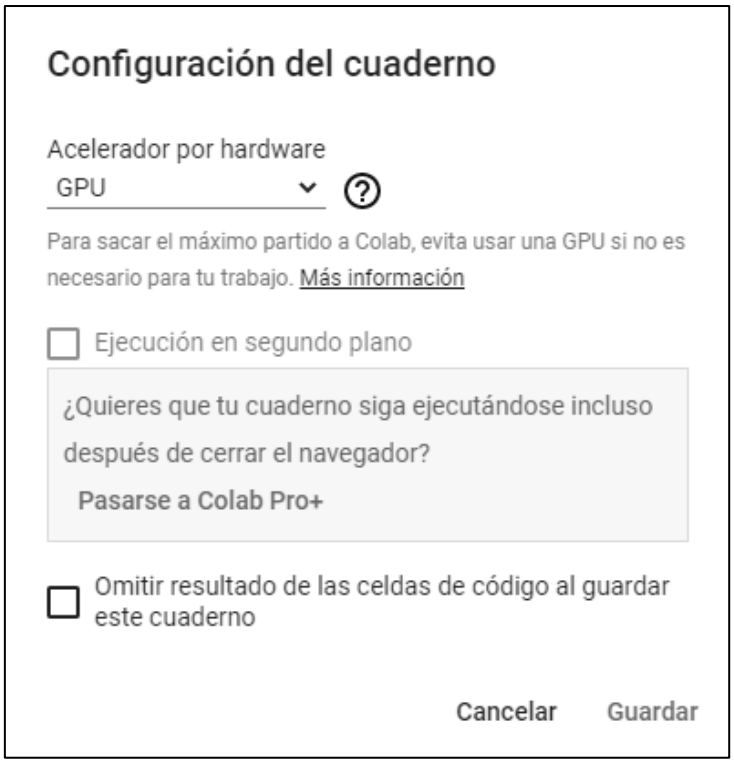


Figura 80: Configuración del hardware

- Ahora se clona el repositorio en el cual se encuentra la arquitectura de la red.

```
!git clone https://github.com/AlexeyAB/darknet

Cloning into 'darknet'...
remote: Enumerating objects: 15412, done.
remote: Total 15412 (delta 0), reused 0 (delta 0), pack-reused 15412
Receiving objects: 100% (15412/15412), 14.02 MiB | 17.86 MiB/s, done.
Resolving deltas: 100% (10356/10356), done.
```

Código 6: Código entrenamiento, clonar la arquitectura de la red

Una vez que se ha clonado la arquitectura, aparecerá la carpeta con el nombre de Darknetjunto a todos sus archivos que componen la red neuronal.

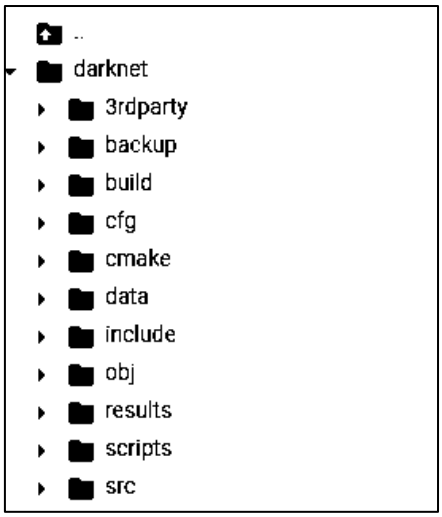


Figura 81: Carpetas de arquitectura de la red

- Dentro de esta red se deben cambiar varios parámetros para el entrenamiento funcione correctamente. Dentro de la carpeta darknet se debe cambiar el uso de la GPU. Para activar las opciones de GPU Y CUDNN se asigna el número uno, al igual que para la opción de openCV, que es una biblioteca de visión artificial.

```
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
```

Código 7: Código entrenamiento, clonar la arquitectura de la red

- A continuación, se verifica el CUDA para el entrenamiento. CUDA es el cómputo acelerado por GPU, es decir, un soporte para la programación en paralelo para mejorar el rendimiento en sus aplicaciones.

```
!/usr/local/cuda/bin/nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
```

Código 7: Código entrenamiento, CUDA

- Ahora, con el comando *make* se construye la red darknet y todas sus dependencias con el comando principal para ser capaz de correr los cambios asignados anteriormente.

```
!make

mkdir -p ./obj/
mkdir -p backup
chmod +x *.sh
```

Código 7:Código entrenamiento, Darknet

- Como se ha comentado Google Colab solo dejará trabajar en el software durante un periodo de doce horas. Para impedir que el trabajo que lleva hasta el momento, se conecta Colab con Google Drive y se crea una carpeta especifica como yolov3, para poder guardar el progreso que lleve hasta el momento. Con el siguiente comando conectaremos ambos.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Código 7: Código entrenamiento, Colab y Drive

- Una vez conectados, donde se ha creado la carpeta en Drive de yolov3 se introduce el .zip que se ha creado anteriormente con el software Labeling. Donde contiene las imágenes y las características necesarias para el entrenamiento.
- Se descarga el archivo *yolov3.cfg,* que se cambiará el nombre a *yolov3_custom.cfg*al modificar varios parámetros dentro del archivo. Este archivo puede se puede preparar para realizar un test de detección o para entrenar un nuevo objeto.

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 4000
policy=steps
steps=3200,3600
scales=.1,.1
```

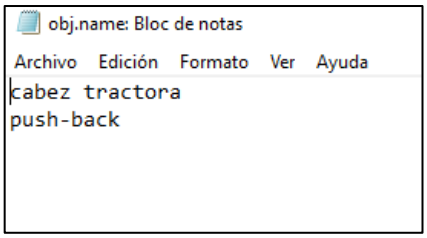
Código 7: Código entrenamiento, modificación archivo .cfg

- Se debe cambiar el batch a 64 y subdivisión a 16 para que este en modo entrenamiento y no testeo.
- Se establece el tamaño de la red de ancho y alto en 416.
- El max_batches se obtiene multiplicando 2000 por el número de clases. En este caso se van a entrenar dos objetos por lo que será 4000.
- Los pasos serán irán del 80% al 90% del max_batches, es decir, de 3200 a 3600.

```
[yolo]
mask = 3,4,5
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=2
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1



[route]
layers = -4

[convolutional]
batch_normalize=1
filters=21
size=1
stride=1
pad=1
activation=leaky
```

Código 7: Código entrenamiento, modificación archivo .cfg

- En las clases se pone el número de objetos que se vayan a entrenar.
- El filtro será: (clases+5)x3. En este caso 21. Estos pasos se deben realizar tres veces para las tres capas convolucionales que tiene YOLO.
- Posteriormente en la carpeta creada en Drive se tienen que incluir dos archivos. El primero es *obj.names* el cual contiene los nombres de las clases que se quieren detectar y *obj.data* para saber la ruta dónde están localizados los archivos, así como el directorio donde se guardan los modelos generados en el entrenamiento y número de clases que se va a entrenar. El backup es una carpeta donde se guardará el progreso del entrenamiento cada 1000 iteraciones por si Colab se cerrará antes de finalizar.



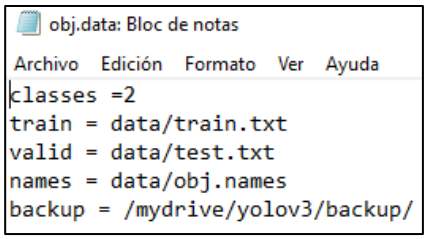


Figura 82 y 83: Archivos obj.name y obj.data

- Una vez creado ambos archivos, se pasa a ejecutar el código *generate_train.py.* Este código se encarga de crear un archivo *train.txt*donde incluye todas las imágenes presentes en la carpeta que habíamos subido, necesario para que a la hora de entrenar el programa sea capaz de hallar las mismas imágenes.

```
import os

image_files = []
os.chdir(os.path.join("data", "obj"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        image_files.append("data/obj/" + filename)
os.chdir("..")
with open("train.txt", "w") as outfile:
    for image in image_files:
        outfile.write(image)
        outfile.write("\n")
    outfile.close()
os.chdir("..")
```

Código 7: Código entrenamiento, archivo generate_train.py

- El archivo *train.txt* creado, debería aparecer de la siguiente forma con todas las fotos que ha procesado.

```
!unzip ../obj.zip -d data/

Archive:  ../obj.zip
   creating: data/Imagenes push back/
  inflating: data/Imagenes push back/classes.txt
  inflating: data/Imagenes push back/th (1).jpg
  inflating: data/Imagenes push back/th (1).txt
  inflating: data/Imagenes push back/th (10).jpg
  inflating: data/Imagenes push back/th (10).txt
  inflating: data/Imagenes push back/th (11).jpg
  inflating: data/Imagenes push back/th (11).txt
  inflating: data/Imagenes push back/th (12).jpg
  inflating: data/Imagenes push back/th (12).txt
  inflating: data/Imagenes push back/th (13).jpg
  inflating: data/Imagenes push back/th (13).txt
  inflating: data/Imagenes push back/th (14).jpg
```

Código 7: Código entrenamiento, fotos procesadas

- Antes de comenzar con el entrenamiento el último paso será descargar los pesos para las capas convolucionales de la red YOLOv3. Al usar estos pesos ayudará a una detección mejor y que el proceso de entrenamiento no sea demasiado largo. Este archivo se llama *darknet53.conv.74.*

```
!wget http://pjreddie.com/media/files/darknet53.conv.74

URL transformed to HTTPS due to an HSTS policy
--2022-04-22 10:40:32--  https://pjreddie.com/media/files/darknet53.conv.74
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 162482580 (155M) [application/octet-stream]
Saving to: 'darknet53.conv.74'

darknet53.conv.74   100%[===================>] 154.96M  58.1MB/s    in 2.7s

2022-04-22 10:40:35 (58.1 MB/s) - 'darknet53.conv.74' saved [162482580/162482580]
```

Código 7: Código entrenamiento, descarga archivo darknet.conv.74

- Finalmente, se ejecuta el comando para entrenar la red y detectar los objetos deseados. Como se ha comentado se irán guardando los pesos de los archivos cada 1000 iteraciones. Si el entrenamiento se detuviera se podría seguir desde el último archivo creado para no perder la evolución, hasta el último archivo de pesos.

    - *Yolov3_custom_1000.weights*
    - *Yolov3_custom_2000.weights*
    - *Yolov3_custom_last.weights*

```
!./darknet detector train data/obj.data cfg/yolov3_custom.cfg darknet53.conv.74 -dont_show

 CUDA-version: 11010 (11020), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 3.2.0
yolov3_custom
 0 : compute_capability = 370, cudnn_half = 0, GPU: Tesla K80
net.optimized_memory = 0
mini_batch = 4, batch = 64, time_steps = 1, train = 1
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  14 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
```

Código 7: Código de entrenamiento de la red

También se representa la evolución que lleva a cabo el entrenamiento con una gráfica que se muestra a continuación. En ella se representa como va disminuyendo el error a lo largo de las iteraciones.

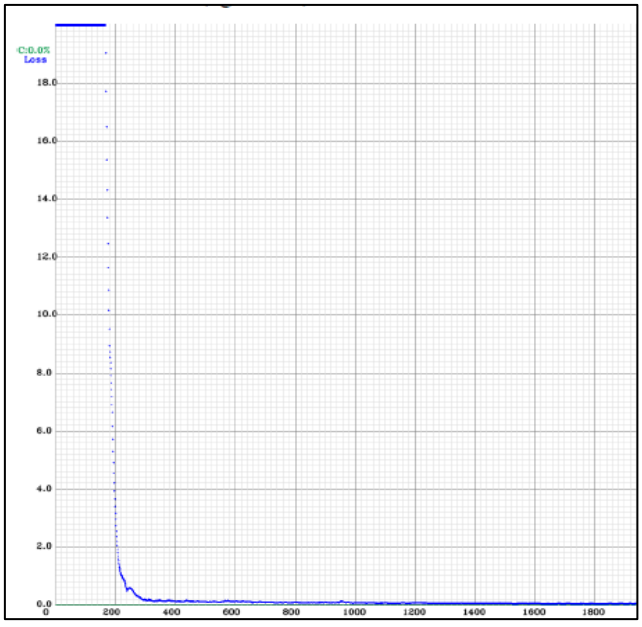


Figura 84: Error vs iteraciones

Como se observa, al inicio el error es muy grande como es lógico. Al avanzar en las iteraciones el error va disminuyendo. Es recomendable que para finalizar el proceso de configuración de los pesos, el error resultante sea menor que dos, pero cuanto más bajo sea más precisión tendrá. En este caso no tardará mucho en conseguir un error aceptable por la cantidad de fotos

que se han introducido. Si se coge un conjunto de datos mayor el proceso será más lento.

Este es el proceso que se debe seguir para poder entrenar cualquier clase de objeto. Es un proceso un poco costoso si se debe de especificar donde está el objeto en cada una de las imágenes. Como se ha comentado, cuantas más imágenes se introduzcan en el entrenamiento más fiables será la detección, pero mayor tiempo de proceso tendrá.

Por último, se va a testear el entrenamiento final. Se introducirán varias imágenes en el programa para visualizar la probabilidad de detección.

Vehículo push-back:

Figura 85 y 86: Testeo de vehículos push-back

Figura 87 y 88: Testeo de vehículos push-back

Vehículo cabeza tractora:

Figura 89 y 90: Testeo de vehículos cabeza tractora

Figura 91, 92 y 93: Testeo de vehículos cabeza tractora

Por último, se han representado varias imágenes con los diferentes vehículos entrenados. Como se puede observar, se realiza una predicción fiable superando una probabilidad de 50%. Dependiendo de la imagen tendrá un valor u otro. La probabilidad de detección y acierto puede incrementar notablemente si durante el entrenamiento se introdujeran muchas más imágenes para obtener las características de cada una de ellas y hacer que la red tenga un aprendizaje más extenso. El reconocimiento de imagen se puede aplicar a videos en tiempo real para determinar los vehículos que se encuentran en pista, la cantidad de vehículos que se encuentran estacionados o en movimiento, la necesidad de cada una de ellos dependiendo del tránsito ocupacional que tengan…

CONCLUSIONES.

Finalmente, tras un largo trabajo se ha podido llegar a varias conclusiones. Al principio, se ha pasado por una parte teórica para entender y desarrollar un poco en detalle el funcionamiento de la inteligencia artificial y todas las secciones de las cuales está compuesta. Pasando por el aprendizaje automático y posteriormente, el aprendizaje profundo. Los objetivos principales de este proyecto han sido el algoritmo para la detección, clasificación y conteo de vehículos en las entradas de los aeropuertos, junto con la clasificación y reconocimiento de ciertos vehículos aeroportuarios.

En primer lugar, para realizar el programa final de clasificación de vehículos se ha pasado por varias opciones de diferentes algoritmos, probando la funcionalidad y efectividad de cada uno. Tras el estudio y desarrollo de cada uno de ellos se ha llegado a la conclusión que el Deep Learning o aprendizaje profundo es la mejor opción para los objetivos establecidos, más concretamente YOLO (You Only Look Once).

El primer algoritmo estudiado ha sido la clasificación en cascada de Haar que se descartó principalmente porque tiene problemas con las variaciones de escala. En determinados videos se producen cambios de dimensiones en la imagen los cuales no procesa de manera correcta este algoritmo. También consta de archivos XML pre-entrenados, resultando difícil poder predecir varios objetos al mismo tiempo.

Posteriormente, se ha desarrollado la sustracción de fondo. En primera aproximación, es un mecanismo útil para detectar objetos en movimiento, pero no para la clasificación. Puede llegar a clasificar entre tres tipos de vehículos, pero guiados por el tamaño de cada uno de ellos no por el propio vehículo. Esto puede dar errores si se trata de video en tiempo real con cambios de dimensión.

Por todo lo anterior, se llega al programa final utilizando una mezcla de sustracción de fondo para la detección de movimiento junto con la clasificación de vehículos producida por YOLO. Este algoritmo tiene un funcionamiento óptimo en comparación al resto, pudiendo detectar entre varios vehículos con una gran probabilidad de acierto.

Gracias a este algoritmo se puede realizar un conteo de vehículos en las cercanías de los aeropuertos para estudiar su tránsito y poder realizar diversos estudios para conocer la ocupación de los aeropuertos y el estado de los mismos.

Como apartado final del proyecto, se ha realizado un entrenamiento de la red neuronal para YOLOv3. Este proceso se ha detallado paso a paso para conseguir la distinción de dos vehículos aeroportuarios. Finalmente, se ha conseguido poder clasificar los dos tipos de vehículos, destinado a conocer el funcionamiento y necesidades de su uso. Ha resultado un camino largo de recorrer, pero observando los resultados finales será una buena opción para posteriores detecciones.

BIBLIOGRAFÍA.

[1] M. A. Antúnez. Algoritmos de detección de objetos para la detección y seguimiento de ojos. Universidad politécnica de Catalunya, 2019.

[2] J. D. Echeverry. Detección de rostros en imágenes digitales usando clasificación en cascada. Universidad Tecnológica de Pereira, 2008.

[3] A. García. Deep Learning: Neural Networks for Object Detection. Universidad autónoma de Barcelona.

[4] A. Silva. Estudio comparativo de modelos de clasificación automática de señales de tráfico. Universidad Politécnica de Navarra, 2020

[5] L. F. Escobar. Evaluación de algoritmos de sustracción de fondo para conteo de personas. Universidad del Valle, 2016.

[6] S.Kshatrian. Simplest way Train a YOLO model for Custom object Detection, 2018

[7] L. A. Fernández. Extensión de algoritmos de sustracción de fondo para cámaras PTZ. Universidad Nacional del Centro de la Provincia de Buenos Aires, 2017.

[8] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. IJCV, 2013.

[9] M. Amarilla, G. Devincenzi, R. Ramos, V. Schvastzman. Conteo vehicular utilizando visión artificial, 2014.

[10] A. Krizhevsky, I. Sutskever, and G. Hinton. Image Net classification with Deep convolutional neural networks. In NIPS, 2012

[11] A. Yabo. Video-detección vehicular para sistemas inteligentes de transporte (SIT). Universidad Nacional de Quilmes, 2016.

[12] F.R. Sisalima. Sistema para detección y conteo vehicular aplicando técnicas de visión artificial. Facultad de la Energía, las Industrias y los Recursos Naturales No Renovables, 2018

[13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In ICLR, 2014

[14] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid methods in image processing. RCA engineer, 1984

[15] D. Horcajadas. Metodología para la detección de objetos en imágenes basada en la librería YOLO con aplicación a la detección de carros. Universidad de Sevilla, 2021.

[16] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017

[17] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv, 2018

[18] J. Redmon, A. Farhadi, S.Divvala and R.Girshick. You Only Look Once: Undified, Real-Time Object Detection, 2016

[19] J. Redmon and A.Farhadi. YOLOv3: An Incremental Improvement, 2018

[20] O. Barnichand M. Van Droogenbroeck. ViBe: A universal background subtraction algorithm for video sequences. Vol. 20,2011.

[21] M. Piccardi. Background subtraction techniques: a review, 2004.

[22] T. Bouwmans& F. Porikli, B. Hoferlin, A. Vacavant. Background modeling and foreground detection for video surveillance, 2015

[23] A. Rosebrock. Deep Learning, how OpenCV blobFromimage Works, November 2017.

[24] Intelligent Information Technologies. Deep learning and Convolutional Neuronal Networks, Janauary 2018

[25] T. Shireen, M. Khaled, S. El, H. Sumaya. Moving Object Detection in Spatial Domainusing Background Removal Techniques – State-of-Art, 2008

[26] A.Naskar. YOLO object detection using Deep learning OpencCV, 2021.

[27] TechVidvan. Vehicle Counting, Classification and Detection using OpencCV. 2021

[28] L. Maddalena& A. Petrosino. A self-organizing approach to background subtraction for visual surveillance applications. 2008.