UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN

BIG DATA ANALYTICS - MBI

TRABAJO FIN DE MÁSTER

SISTEMA DE DETECCIÓN AUTOMÁTICO DE PLACAS DE MATRÍCULA EN TIEMPO REAL

NOMBRE:

JESÚS CUESTA TENORIO

CURSO 2020-2021

TÍTULO: SISTEMA DE DETECCIÓN AUTOMÁTICO DE PLACAS DE MATRÍCULA EN TIEMPO REAL

AUTOR: JESÚS CUESTA TENORIO

TITULACIÓN: Máster Universitario en Análisis de Grandes Cantidades de Datos - MBI / Big Data Analytics

DIRECTOR DEL PROYECTO: JOSÉ JAVIER RUIZ

FECHA: OCTUBRE de 2021

RESUMEN

Uno de los campos de aplicación más extendidos de las **Redes Neuronales Convolucionales**, es el de la **Visión Artificial** (**VA**). Los avances en este espacio se están sucediendo de forma acelerada debido a la demanda, entre otros, de su aplicación a sistemas de aumento de seguridad y vigilancia, industria del entretenimiento y numerosas necesidades en el ámbito industrial.

El presente trabajo aborda un ejemplo práctico de aplicación de **Redes Neuronales Convolucionales** en **Visión Artificial**, al reconocimiento de placas de matrícula de vehículos. Aunque son muchos los avances en reconocimiento y clasificación de objetos, siguen existiendo dificultades en conseguir procesamiento en tiempo real, cuando los recursos hardware son limitados y, con este trabajo, se demuestra un caso de uso de reconocimiento de objetos en tiempo real.

El trabajo se basa en la implementación de un sistema de **Visión Artificial** de una sóla pasada, basado en **Redes Neuronales Convolucionales**, conocido como **YOLO** (**You Only Look Once**) (Redmon & Farhadi, 2018) y, en concreto, el proyecto *Open Source* **Darknet** (Bochkovskiy, Wang, & Mark Liao, 2020).

Utilizando imágenes reales, tanto para entrenamiento como para prueba, se demuestra que el sistema escogido, es capaz de reconocer placas de matrícula en tiempo real (superior a 24 *frames* por segundo), mostrando una elevada tasa de acierto, tanto en imágenes estáticas como en vídeo.

Palabras clave: Redes Neuronales Convolucionales (RNC), Visión Artificial (VA), You Only Look Once (YOLO), Reconocimiento de Objetos en Imagen, Reconocimiento de Objetos en Video, Darknet

ABSTRACT

One of the most widespread fields of application of **Convolutional Neural Networks** is that of **Computer Vision (CV)**. Advances in this space are happening at an accelerated rate due to the demand, among others, of its application to systems for increasing security and surveillance, the entertainment industry, and numerous needs in the industrial field.

The present work addresses a practical example of the application of Convolutional Neural Networks in Artificial Vision, to the recognition of vehicle registration plates. Although there are many advances in object recognition and classification, there are still difficulties in achieving real-time processing when hardware resources are limited and, with this work, a real-time object recognition use case is demonstrated.

The work is based on the implementation of a one-shot **Computer Vision** system, based on **Convolutional Neural Networks**, known as **YOLO** (**You Only Look Once**) (Redmon & Farhadi, 2018) and, specifically, the Open Source **Darknet** project (Bochkovskiy, Wang, & Mark Liao, 2020).

Using real images, both for training and testing, it is shown that the chosen system is capable of recognizing license plates in real time (greater than 24 frames per second), showing a high success rate, both in static images and in video.

Keywords: Convolutional Neural Networks (RNC), Machine Vision (VA), You Only Look Once (YOLO), Image Object Recognition, Video Object Recognition, Darknet

AGRADECIMIENTOS

Agradezco a la Universidad Europea de Madrid las facilidades dispuestas para la presentación del trabajo actual y, a mi tutor, José Javier Ruiz, por el apoyo y la ayuda en todo momento.

Índice

Índice de Figuras

CAPÍTULO 1. INTRODUCCIÓN

La **Visión Artificial** es una de las áreas más importantes en las que tienen aplicación las tecnologías que denominamos **Aprendizaje Automático** o ***Machine Learning***. Existen diferentes aproximaciones al trabajo en visión computacional utilizando dichas tecnologías y, una de ellas, es la que acumula más investigación en los últimos años, que es la basada en **Redes Neuronales Multinivel**, un paradigma que ha evolucionado exponencialmente gracias a la evolución de sistemas *software* y *hardware*.

Uno de los trabajos claves en este campo es *Imagenet Classification with Deep Convolutional Neural Networks* (Krizhevsky, Sutskever, & Hinton, 2012), donde se sientan las bases del procesamiento neuronal convolucional aplicado de forma práctica al procesado de imagen.

Una de las mayores ventajas de los sistemas computacionales basados en **Redes Neuronales**, es la capacidad de poder ser entrenados para tareas específicas, sin tener en cuenta procedimientos heurísticos. Esto se consigue, mediante la alimentación del sistema con las salidas deseadas ante entradas conocidas.

En el caso de uso de reconocimiento de objetos en imágenes, el uso de **Redes Neuronales Multinivel**, cuyo caso particular podrían ser las Redes Neuronales Convolucionales, permite evitar la selección manual de las características a detectar. En el ejemplo de detección de la figura de un coche en una imagen, por ejemplo, la diferencia entre uno y otro método podría esquematizarse de la siguiente forma:

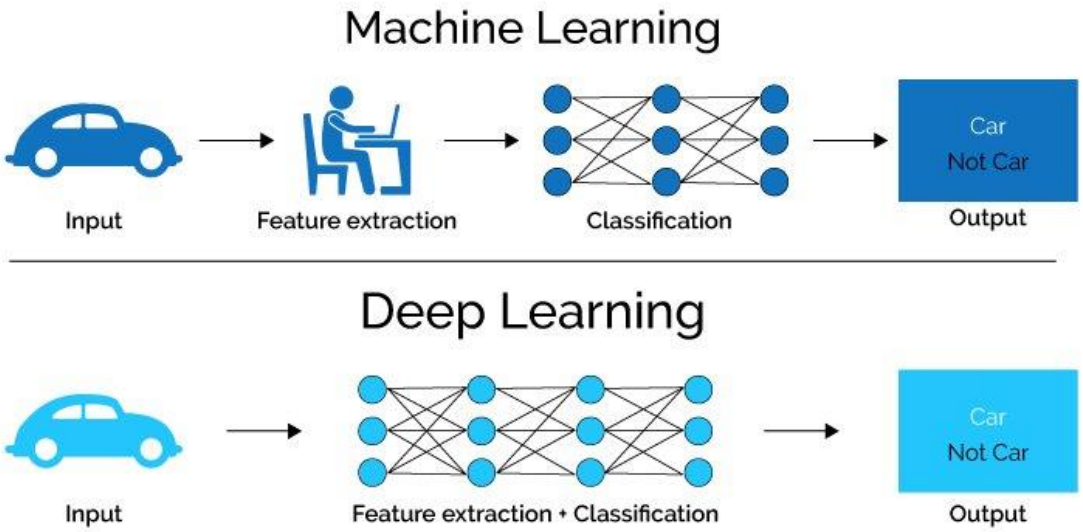


*Ilustración 1. **Machine Learning** vs **Deep Learning** (Pai, 2020)*

Gracias a los avances en este terreno, iniciativas como la conducción automática de **Tesla**, son posibles. Se trata de un sistema real que permite la conducción autónoma, aunque no desatendida, de vehículos a motor. El sistema de **Tesla** basa su actuación en el reconocimiento de objetos en tiempo real a través de las cámaras implantadas en diferentes localizaciones del exterior del vehículo.

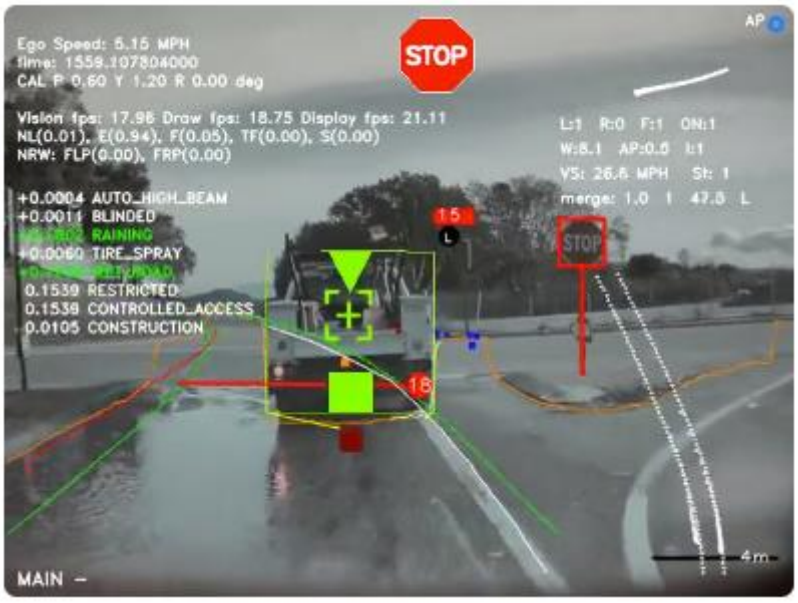


Ilustración 2. Ejemplo de sistema de detección para guiado de Tesla (Tesla Web Page, 2021)

No obstante, hay que tener en cuenta que, para la consecución de objetivos en implantaciones de la tecnología a nivel profesional, o como pueda ocurrir en el caso mencionado, frecuentemente los sistemas requerirán un *hardware* de muy elevadas capacidades o, incluso, del desarrollo microprocesadores específicos.

En el caso de **Tesla**, la compañía ha desarrollado sus propios microprocesadores orientados al procesamiento neuronal y, recientemente, han anunciado la evolución de dicho hardware. En la siguiente ilustración puede observarse la placa de control de un vehículo **Tesla**, con dos microprocesadores específicos.

Los ejemplos de uso de tecnología de reconocimiento de imagen son innumerables, desde aplicaciones industriales para guiado o actuación, hasta la seguridad, para la detección de amenazas, identificación de personas, etc

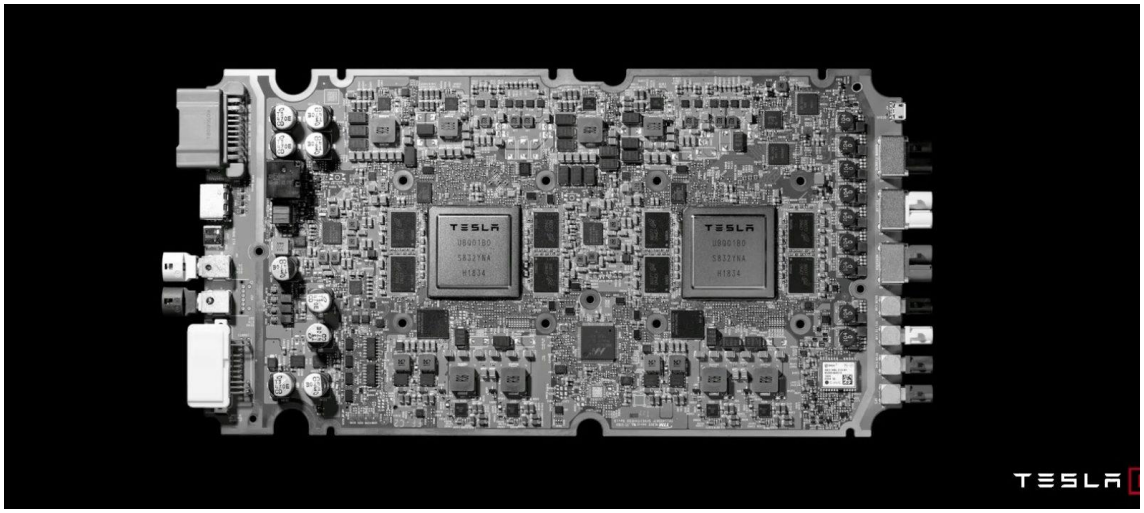


*Ilustración 3. Recientemente Tesla ha anunciado la evolución de sus procesadores para aplicaciones **Deep Learning** (Tesla Web Page, 2021)*

Actualmente, todos podemos llevar eventualmente en nuestro bolsillo dispositivos que utilizan sistemas de red neuronal a escala para operaciones específicas. Son ejemplos de esto la capacidad de difuminar automáticamente las fotos en los modos "retrato" de algunos teléfonos móviles o la capacidad de realizar reconocimiento facial y permitir la autenticación del usuario para dar acceso al dispositivo, aunque este último ejemplo se apoya en hardware específico, ya sean cámaras con reconocimiento en el espectro infrarrojo o, incluso, cámaras 3D (Triggs, 2019).

1.1 Planteamiento del problema objeto del trabajo

El interés para empezar a desarrollar este trabajo, vino motivado por una entrevista con uno de mis clientes en la compañía de *software* para la que trabajo actualmente como responsable de ventas para el sector financiero. Este cliente, una importante compañía de seguros, reconocía la importancia de la utilización de tecnologías asistidas por visión artificial en los procesos de negocios del sector asegurador.

Los posibles campos de aplicación en el sector seguros son múltiples: asistencia a la peritación de siniestros, valoración de estado de los vehículos previo a la creación de pólizas, confirmación visual de partes de siniestro, confirmación de vehículos involucrados en siniestros, identificación de asegurados, etc.

Esta compañía, igualmente reconocía la dificultad de la implementación de este tipo de tecnologías en sus procesos y, cómo algunas de sus aplicaciones podrían mejorar mediante la incorporación de la capacidad de reconocimiento de placas de matrícula o de números de bastidor, que pudiesen ser obtenidos de imágenes o vídeos.

Dicha conversación, fue la idea que originó el presente trabajo y, el objetivo, de encontrar un procedimiento que permitiese reconocer objetos, específicamente placas de matrícula, en imágenes. Adicionalmente, añadí el interés en la capacidad de

procesamiento en tiempo real, entendida como la detección de placas de matrícula en objetos en movimiento a 24 o más, *frames* por segundo (*fps*).

El trabajo en este proyecto comenzó en 2019 y, desde ese momento se han producido interesantes avances en las aplicaciones de la tecnología al sector asegurador.

Recientemente, la compañía Mapfre anunciaba el uso de técnicas de peritación basadas en ***Deep Learning*** (Leonor, 2021). Tecnología con la que esperar aumentar la efectividad de las peritaciones y automatizar procesos, con el consiguiente ahorro de costes.

Ilustración 4. Aplicación de Deep Learning al negocio de seguros

1.2 Objetivos del proyecto

El trabajo se plantea con los siguientes objetivos primordiales a cumplir durante la ejecución del del proyecto:

Objetivos primarios:

- Desplegar un sistema computacional basado en **Redes Neuronales Convolucionales**, capaz de realizar análisis de imagen.
- Entrenar el sistema para el reconocimiento de patrones en forma del objeto deseado especificado durante el entrenamiento.
- Capacidad de clasificación del objeto, cuyo reconocimiento es deseado, por su grado de probabilidad de identificación correcta.
- Asignación correcta de la tipología de objeto entrenado tras el reconocimiento del mismo.
- Capacidad de detección en tiempo real, entendida esta como la posibilidad de detección del objeto entrenado en imágnes en movimiendo a una tasa igual o superior a 24 frames por segundo (*fps*).

Objetivos secundarios:

- Identificar los requisitos mínimos *hardware* para la consecución de los objetivos primarios.

- Establecer un procedimiento replicable, que pueda facilitar el uso del proyecto en otras plataformas estándar, consiguiendo los objetivos primarios.
- Desplegar una plataforma que pueda ser mejorada mediante el desarrollo de aplicaciones que utilicen sus capacidades, en leguajes Python o C++.

1.3 Realización del trabajo

El trabajo ha contado con múltiples retos. El primero de ellos fue la selección de la tecnología a emplear para el desarrollo. Inicialmente se investigaron las posibilidades existentes que utilizasen procedimientos heurísticos, es decir, la capacidad de acometer la detección de objetos mediante el uso de reglas y transformaciones prefijadas.

Para ello se utilizan transformaciones sobre las imágenes de partida, como puedan ser las reducciones o filtros de "ruido", los reescalados, conversión a espacio de color de tonalidades de gris o blanco y negro y técnicas de aumento de contraste. El objetivo es amplificar las diferencias cromáticas que puedan facilitar la diferenciación del objeto deseado. Este tipo de técnicas han sido usadas ampliamente por la industria en el reconocimiento óptico de caracteres u OCR.

Tras las transformaciones adecuadas, el proyecto se reduce a un problema de regresión, donde se desea estimar una variable dependiente (presencia de un objeto), en base a variables independientes (*píxeles* o conjuntos de *píxeles*). La clave será eliminar la información no necesaria o redundante, mediante el empleo de los algoritmos adecuados.

Tengamos en cuenta que, una placa de matrícula puede representar una zona pequeña dentro de una imagen de gran tamaño, por lo que una gran parte del problema es eliminar información sobrante, algo que, computacionalmente, puede ser bastante costoso.

El trabajo *License Plate Number Recognition* (García, Díez-Pastor, Rodríguez, & Maudes, 2008), hace un gran trabajo en explicar el proceso seguido con este enfoque sobre placas de matrícula europeas. Los autores se basan principalmente, en el uso de un filtro de tipo Sobel sobre las zonas de baja saturación de color (las matrículas son principalmente caracteres negros sobre fondo blanco).

Ilustración 5. Transformaciones para detección de placas de matrícula por procesamiento heurístico basadas en filtro Sobel

El enfoque heurístico tiene diversos problemas:

- La calidad de la detección está limitada a la calidad y efectividad de las reglas implementadas. Las reglas imponen ciertas limitaciones de partida, como puedan ser en el trabajo comentado, el descarte del 30% de la zona superior de la imagen y el 10% de la zona inferior. Las placas situadas en dichas zonas no serán detectadas.
- La necesidad computacional es elevada y pone en riesgo el objetivo de detección en tiempo real.
- Las limitaciones en el proceso de detección exigen unas condiciones de uso muy determinadas: ángulos de visualización de las placas, principalmente frontales.

Estas limitaciones, hacen indicado el uso de este tipo de técnicas en casos de uso delimitados, como pueda ser la detección de placa de matrícula en accesos a aparcamientos, donde existen unas condiciones específicas de iluminación, colocación y ángulo desde el que se captura la imagen.

Por todo lo anterior, se descarta el planteamiento heurístico, lo que nos lleva a la selección de un sistema de procesamiento basado en **Redes Neuronales**, del que hablaremos en detalle en el siguiente capítulo.

1.4 Aproximación de implementación del proyecto

Dentro de las tecnologías de **Redes Neuronales**, existen múltiples aproximaciones posibles para el uso en el caso de detección y clasificación de objetos en imagen. La mayor parte de ellas comienzan con la selección de un marco de desarrollo especializado en Redes Neuronales (***AI Development Framework***), sobre el que pueden desarrollarse o implantarse módulos tengan aplicación a nuestro caso.

Las plataformas más habituales son:

- TensorFlow.
- Pytorch.
- Keras.
- Caffe

Estas plataformas son de tipo generalista. Requieren la definición de un modelo específico para el tratamiento de las imágenes y de las zonas, que puedan llevar a la detección de los objetos.

Otro de los enfoques posibles es el uso de una plataforma específica para la detección de objetos, que tenga embebida una arquitectura neuronal y que pueda ser entrenada para los objetos de detección necesarios. Ejemplos de estas plataformas son **Darknet** o **Darkflow**. Como nota, hay que destacar que **Darkflow** está construido sobre el *framework* **TensorFlow**.

La selección del modelo algorítmico de implementación, como veremos más adelante, define la mayor parte del proyecto y el éxito o fracaso del mismo, sobre todo teniendo en cuenta el requisito de procesamiento en tiempo real sobre plataformas de hardware limitadas.

Para el trabajo, se ha optado por el modelo **YOLO** (*You Only Look Once*), que es el que requiere menores capacidades de hardware para conseguir el objetivo de procesamiento en tiempo real. Y en concreto se ha escogido la implementación **Darknet**. En los siguientes apartados de la presente memoria, se analiza en detalle esta plataforma y se detallan los pasos seguidos para la ejecución del proyecto de forma completa.

1.5 Estructura del trabajo

El trabajo se ha estructurado en los siguientes apartados principales:

- Explicación de la plataforma tecnológica escogida basada en **Red Neuronal**: **YOLO** - **Darknet**
- Despliegue de la plataforma tecnológica escogida
- Entrenamiento del sistema con imágenes reales
- Evaluación del sistema sobre imágenes estáticas y vídeo

El trabajo incluye la explicación de los marcos teóricos y técnicos pertinentes para el caso y detalla los pasos seguidos y reproducibles para la consecución de una plataforma funcional.

CAPÍTULO 2. PLATAFORMA TECNOLÓGICA BASADA EN CNN: YOLO

Como se ha introducido, para el proyecto se ha escogido trabajar con la plataforma **YOLO** construida sobre **Darknet**. A continuación, se explican los motivos que han basado dicha selección y se resumen el marco teórico que aplica a esta tecnología.

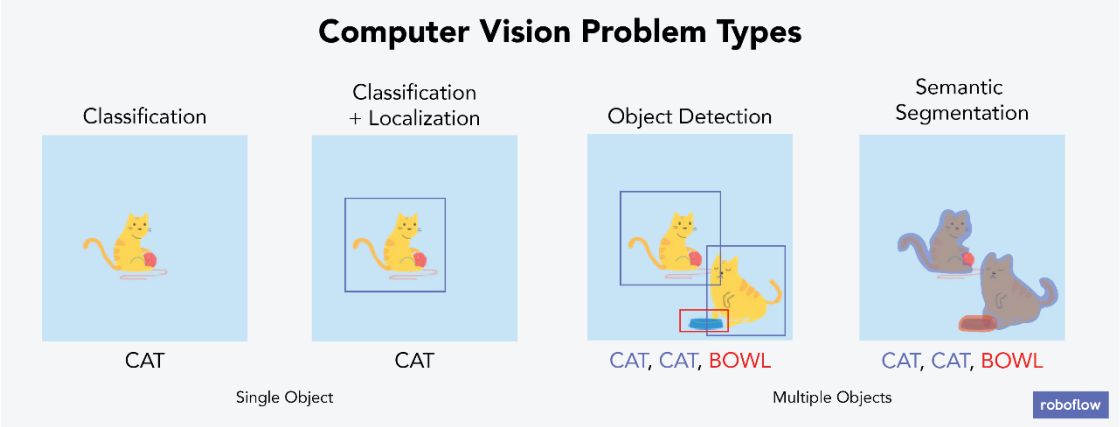


Ilustración 6. Tipos de problemas en visión por computador (Nelson, Roboflow, 2020)

El problema que nos ocupa, la detección de placas de matrícula en imagen, estaría encuadrado en la Detección de objetos. Ya se ha mencionado que, eliminados los enfoques heurísticos al problema de detección y clasificación de objetos, lo que nos queda son los modelos neuronales.

En el modelo neuronal, un sistema multinivel interconectado, que toma como referencia el procesamiento individual de una neurona, se encarga de trabajar con un conjunto de entradas para proporcionar determinadas salidas. La belleza del modelo neuronal, es que es entrenable, siendo durante el proceso de entreno cuando se generan las relaciones entre las neuronas y niveles de procesamiento.

El siguiente esquema facilita la comprensión de lo comentado. En las capas que denominamos “ocultas”, las entradas son ponderadas y atraviesan una función matemática de activación, cuyo resultado alimentará los subsiguientes niveles neuronales.

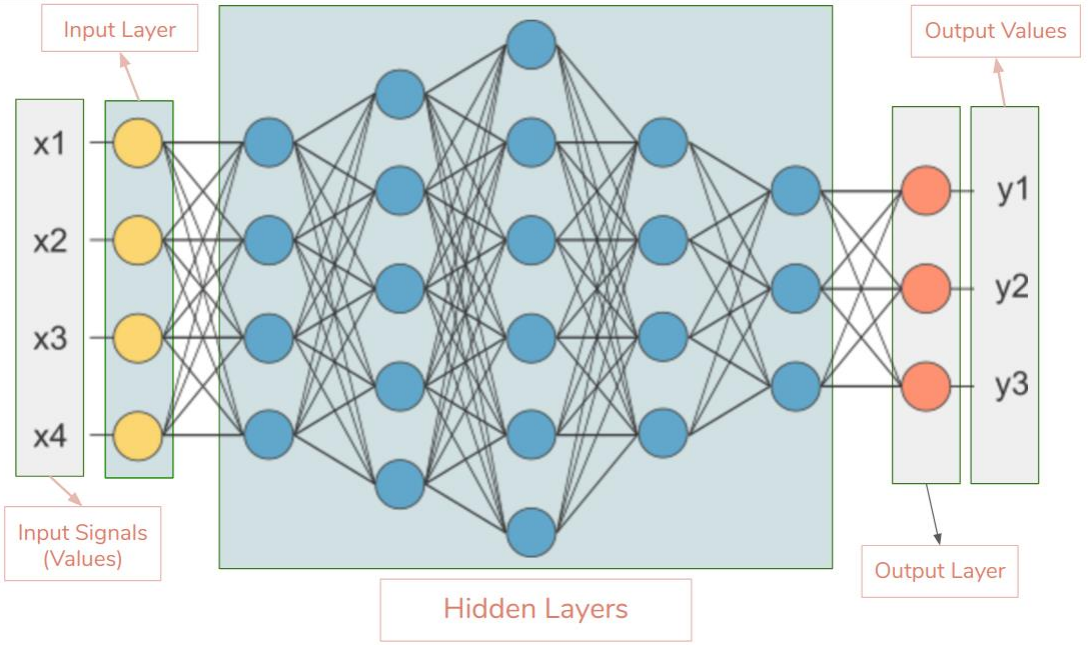


Ilustración 7. Esquema general de una red neuronal (Ahirwar, 2020)

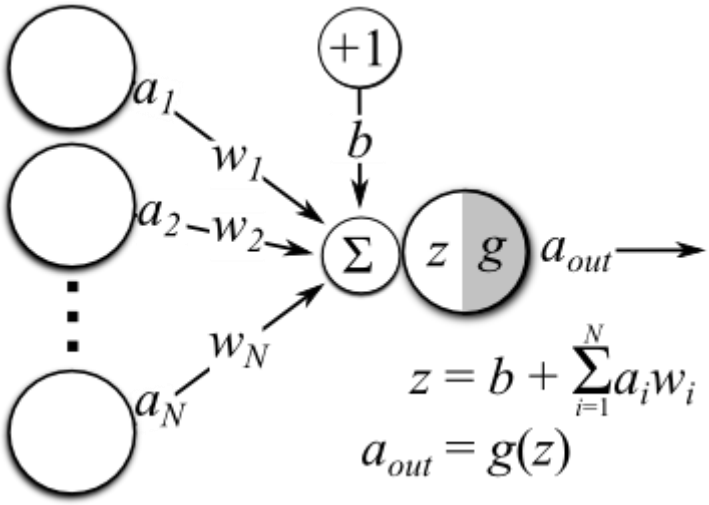


Ilustración 8. Pesos en red neuronal (Ahirwar, 2020)

La aplicación básica del procesamiento neuronal a imagen, consiste en la sectorización masiva de imágenes y estudio de cada una de dichas regiones. El problema de usar redes neuronales típicas en el análisis de imagen es que la capacidad de procesamiento sería ingente y tendiente a *overfitting*. Por ello, típicamente las redes neuronales aplicadas a imagen serán de tipo **Convolucional**. Las redes neuronales de este tipo, no necesariamente obligan que todas las neuronas de un nivel estén conectadas con todas las del siguiente nivel, dado que utilizan técnicas de filtrado y agrupación que simplifican la conectividad entre neuronas y niveles.

Una aproximación específina a las **Redes Neuronales Convolucionales**, con las **R-CNN** (***Region*** based **CNN**). En estas redes, se realiza un planteamiento de estudio en regiones

(típicamente 2000) específico que alivia bastante la carga de procesamiento del modelo. Para ello, algoritmos específicos estudian de forma recursiva las regiones de la imagen y proponen agrupaciones de estas que simplifiquen el proceso.

El problema de esta aproximación, es que sigue siendo intensivo computacionalmente. Al incluir algoritmos específicos para el filtrado neuronal, estamos forzando una parte heurística dentro del modelo, por así decirlo, lo que puede llevar ineficiencias o errores de selección frente al procesamiento neuronal completo.

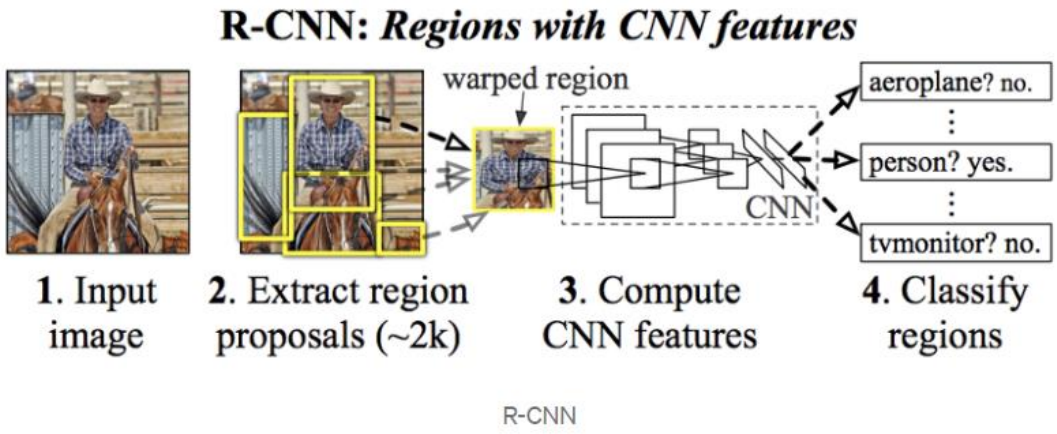


Ilustración 9. Modelo de CNN basado en Regiones (R-CNN) (Gandhi, 2018)

La evolución de **R-CNN** se denomina ***Fast* R-CNN**. Se trata de un procesamiento neuronal de dos fases. Primero se hace un procesamiento convolucional de la estructura general de la imagen para extraer las características principales y, posteriormente, ese resultado convolucional se alimenta a una red neuronal completamente conectada. La diferencia con **CNN** es que, no se realiza una convolución por cada región, sino una convolución inicial sobre la imagen global que extrae el mapa de características, reduciendo el número de zonas a tratar por la red neuronal completamente conectada.

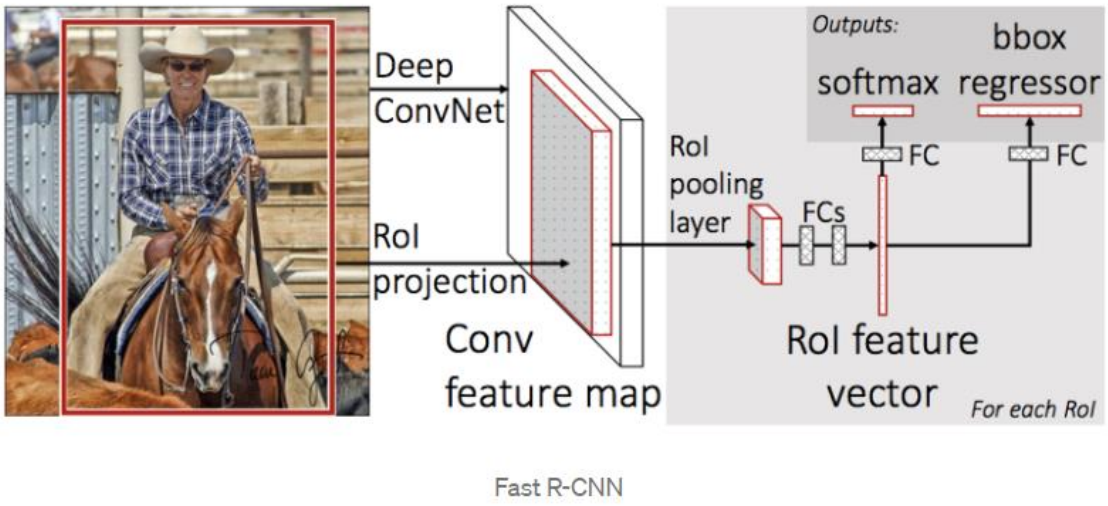


*Ilustración 10. **R-CNN** (Gandhi, 2018)*

Frente a estas típicas aproximaciones, encontramos el enfoque diferencial de **YOLO**. **YOLO** es un acrónimo que en lengua inglesa representa habitualmente la expresión "*You Only Live Once"* o "sólo se vive una vez". Los autores Redmon, Girshick, Divvala y Farhadi (2016), hicieron un divertido juego de palabras para reutilizar el acrónimo en forma de "*You Only Look Once*".

Ilustración 11Portada del trabajo Yolo (Redmon et al., 2016)

Habitualmente el problema de la detección está basado en dos fases:

- Primero identificando una "caja", lo que representa un problema de regresión
- Y posteriormente, identificando la clase a la que pertenece el objeto "cajeado", lo que se configura como un problema de clasificación

Pero el aspecto diferencial de **YOLO**, es que sólo se procesa la imagen una única vez. En una sola pasada, una red convolucional analiza la imagen y predice no sólo las posibles

regiones que puedan contener los objetos de interés, sino que también identifica la clasificación de los objetos. Todo en un único paso.

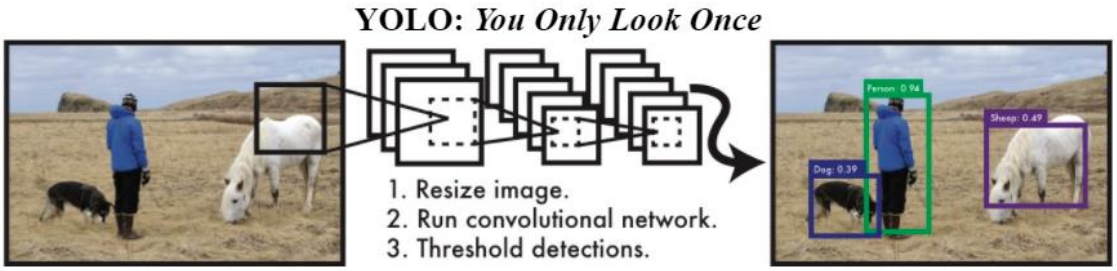


Ilustración 12. YOLO (Redmon, Divala, Girshick, & Farhadi, 2016)

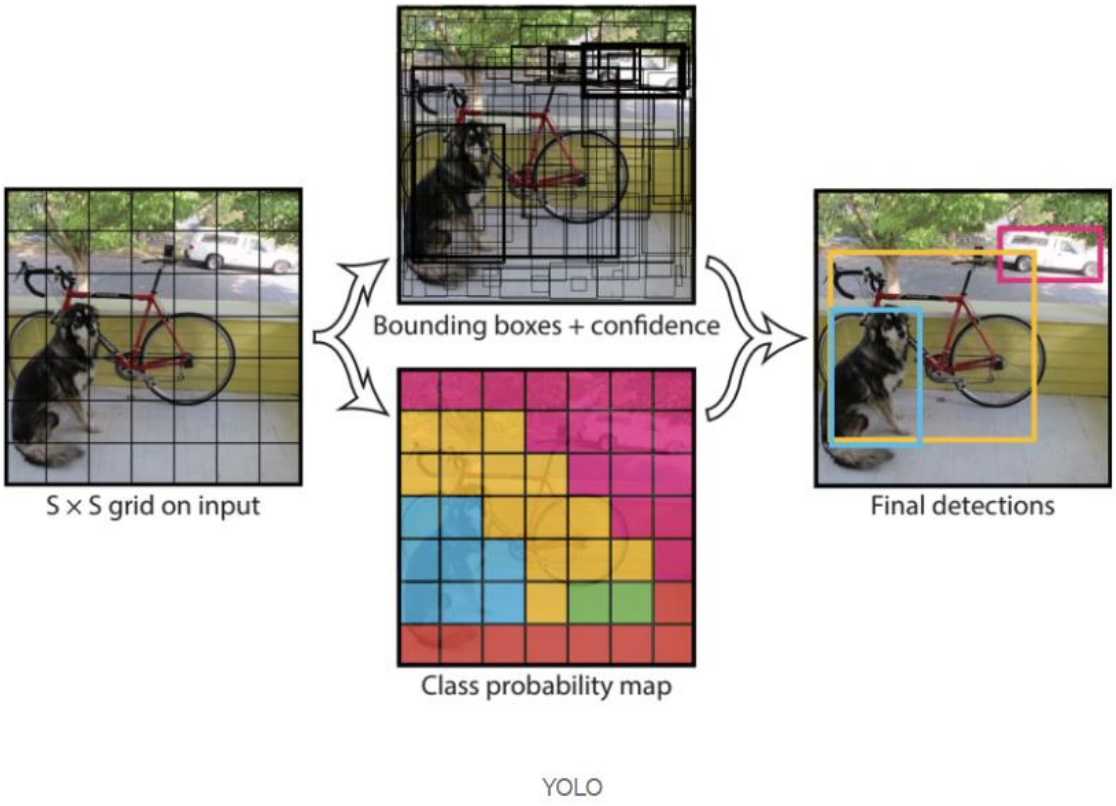


Ilustración 13. YOLO (Redmon, Divala, Girshick, & Farhadi, 2016)

Por lo comentado, **YOLO** se configura como uno de los procedimientos de referencia para el procesado de imágenes en tiempo real, que es justamente el objetivo principal del proyecto.

2.1 Evolución de YOLO

El sistema YOLO y su implementación a nivel programático, **Darknet**, ha tenido varias evoluciones en el tiempo:

- La versión original (Redmon, Divala, Girshick, & Farhadi, 2016)
- La version 2 (Redmon & Farhadi, YOLO9000: Better, Faster, Stronger, 2017)
- La tercera versión, YOLO v3 es del 2018 (Redmon & Farhadi, 2018)
- La versión 4, que es la utilizada en este trabajo (Bochkovskiy, Wang, & Mark Liao, 2020)

La versión era capaz de procesar imágenes a un ratio de 40 a 90 *fps*, pero la versión 3 y posteriores, permiten balancear precisión frente a velocidad (a mayor número de niveles convolucionales, la precisión aumenta a costa de la velocidad).

Existe una versión 5 que no se ha considerado, al estar envuelta en una cierta polémica. Está mostrando resultados de rendimiento contradictorios y aparentemente, sólo mejora en flexibilidad al estar construida sobre **Pytorch**.

El rendimiento de **YOLO**, frente a otros sistemas se ha probado en múltiples ocasiones. Además de enfoque diferencial, su implementación Darknet está desarrollada en C/C++, lo que facilita las compilaciones orientadas a rendimiento. Por otra parte, **Darknet** soporta la capacidad de procesamiento paralelo de las tarjetas gráficas **NVIDIA** y, adicionalmente las librerías **cuDNN** de aceleraciónd de procesamiento neuronal basado en tensores.

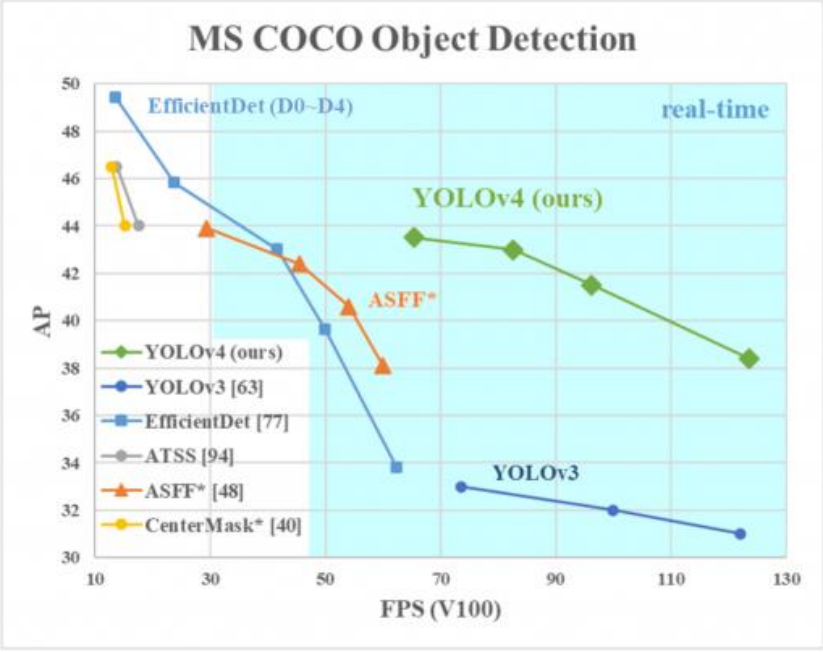


*Ilustración 14. Rendimiento **YOLO v4** (Bochkovskiy, Wang, & Mark Liao, 2020)*

Como curiosidad, comentar que existe una plataforma de **YOLO** desarrollada en TensorFlow. Se denomina **Darkflow**. Se trata de una interesante implementación, pero en pruebas publicadas (Kromann, 2018), **YOLO** Darknet multiplica por tres el rendimiento de YOLO **Darkflow**.

Por todo lo anterior, se ha decidido utilizar **YOLO** y su implementación **Darknet**, como la base del presente proyecto. El siguiente capítulo explica los pasos y problemas encontrados en la implementación de la plataforma tecnológica.

DESPLIEGUE DE LA PLATAFORMA TECNOLÓGICA

2.2 Medios utilizados

El trabajo se ha realizado sobre un ordenador portátil con Windows 10 versión 64 bits, procesador i7-9750H a 2,60 GHz con 6 *cores* y 12 *threads*, 32 GB de memoria RAM y una tarjeta gráfica de tipo *Mobile*, con 6 GB de memoria dedicada y chip 2060 de NVIDIA.

Item	Value
OS Name	Microsoft Windows 10 Pro
Version	10.0.19043 Build 19043
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	DESKTOP-2KE5MGO
System Manufacturer	LENOVO
System Model	81SX
System Type	x64-based PC
System SKU	LENOVO_MT_81SX_BU_idea_FM_Legion Y540-15IRH
Processor	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s)
BIOS Version/Date	LENOVO BHCN41WW, 30/11/2020
SMBIOS Version	3.0
Embedded Controller Version	1.41
BIOS Mode	UEFI
BaseBoard Manufacturer	LENOVO
BaseBoard Product	LNVNB161216
BaseBoard Version	NO DPK
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume5
Locale	Spain
Hardware Abstraction Layer	Version = "10.0.19041.1110"
User Name	DESKTOP-2KE5MGO\squir
Time Zone	Romance Daylight Time
Installed Physical Memory (RAM)	32,0 GB
Total Physical Memory	31,9 GB
Available Physical Memory	7,08 GB
Total Virtual Memory	72,5 GB
Available Virtual Memory	8,97 GB
Page File Space	40,7 GB
Page File	C:\pagefile.sys

Ilustración 15. Características equipo de trabajo

El *hardware*, fue seleccionado como ejemplo de una plataforma hardware con capacidad de procesamiento a nivel computacional y gráfico, de tipo medio. Hubiera sido más sencillo realizar el trabajo en un *hardware* menos limitado, pero como se comentado, el trabajo incluye entre sus objetivos buscar la capacidad mínima de máquina que garantice el éxito de los objetivos del proyecto.

Como confirmaremos más adelante, este equipo representa el mínimo necesario para conseguir la detección de objetos en tiempo real con la plataforma escogida.

Se escogió una plataforma basada en sistema operativo Windows de 64 bits como base del trabajo, por diferentes motivos:

- Amplia disponibilidad de la plataforma sobre *hardware* estándar
- Existencia de múltiples proyectos de reconocimiento de objetos construidos sobre Linux, pero escasez de casos similares para Windows

En el caso de desearse generar un proyecto similar para sistemas embarcados y embebidos, se aconsejaría utilizar Linux por su mayor disponibilidad para este tipo de entornos

2.3 Proceso e instrucciones de despliegue

El proceso de despliegue es muy largo y complejo. Requiere de múltiples piezas de software que, al estar basadas en código abierto (*Open Source*), en algunos casos carecen de instaladores estándar.

Las dos piezas principales de software de este proyecto, **OpenCV** y **Darknet**, han requerido compilaciones específicas para poder cumplir con los objetivos planteados al inicio del trabajo de detección en tiempo real, siendo la de **OpenCV** especialmente compleja como detallaremos a continuación.

El proceso de instalación de la plataforma incluye el despliegue del siguiente software, que se aconseja desplegar en el orden propuesto, debido a dependencias entre los paquetes necesarios.

2.3.1 Visual Studio 2017 Community Edition

Visual Studio es el compilador multilenguaje de Microsoft. El software se descarga de la dirección https://visualstudio.microsoft.com/es/vs/

El proyecto ha sido realizado con la versión 2017 Community Edition, pero puede funcionar igualmente con la versión 2019. La última revisión de las librerías **CUDA**, de las que hablaremos posteriormente, aconseja la versión 2019 pero, por un cierto tiempo seguirá admitiendo la 2017. Se ha escogido la versión 2017 porque garantiza una mejor compatibilidad en la compilación del software **OpenCV** que veremos posteriormente.

Durante el proceso de instalación de Visual Studio, hay que habilitar la opción para “Desarrollo para el escritorio con C++” y, adicionalmente, instalar el paquete de idioma inglés adicional. Sin este último paso, será imposible la compilación del software **Darknet** que se explica más adelante.

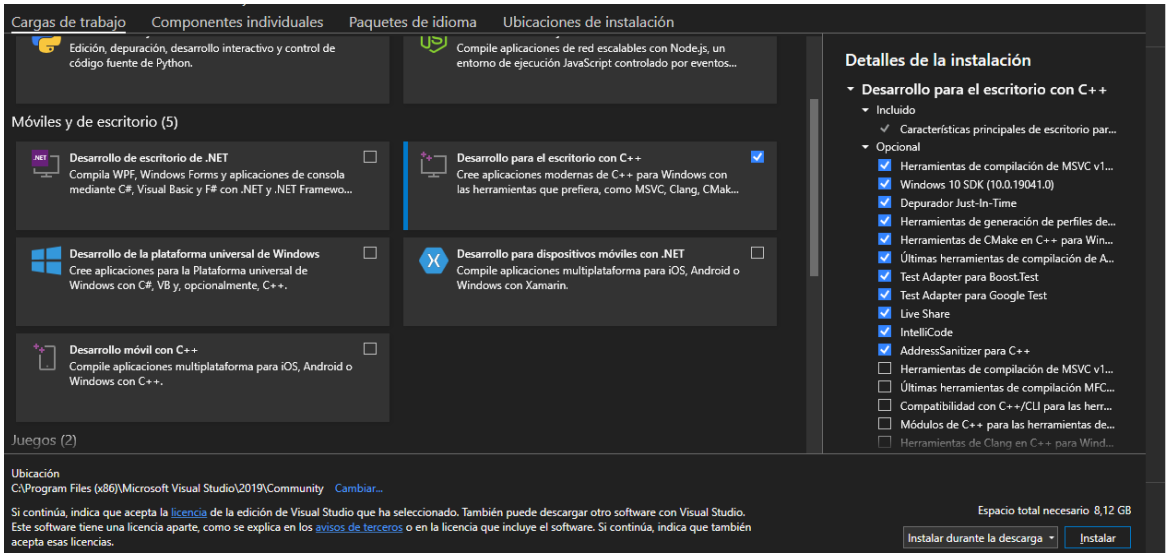


Ilustración 16. Instalacion Visual Studio 2019 Community Edition

Adicionalmente, es obligatorio instalar el paquete de lenguaje inglés para Visual Studio, de lo contrario las compilaciones de algunos productos fallarán.

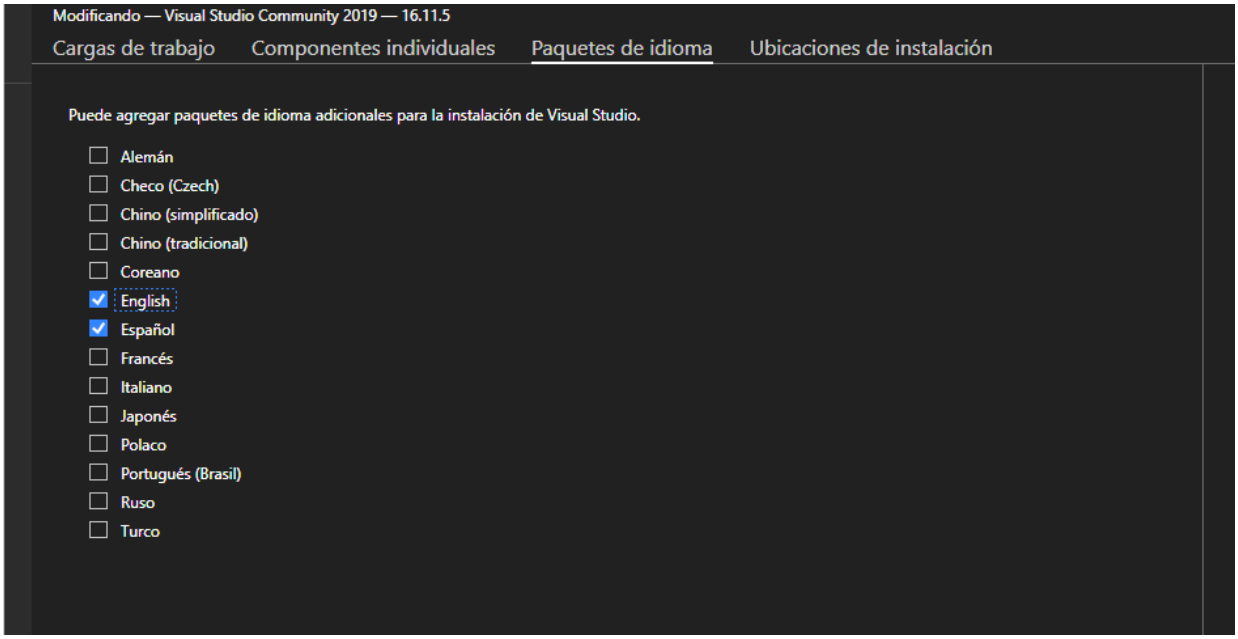


Ilustración 17. Instalación paquete de lenguaje inglés para Visual Studio

2.3.2 Anaconda

Se trata del software para trabajo con Python y Jupyter notebooks. Durante la instalación de este software se instala el interprete de Python y otras múltiples utilidades. La versión de Anaconda utilizada es la 2021.05, siendo el intérprete de Python la versión 3.8.8.

El software se descarga desde https://www.anaconda.com/products/individual

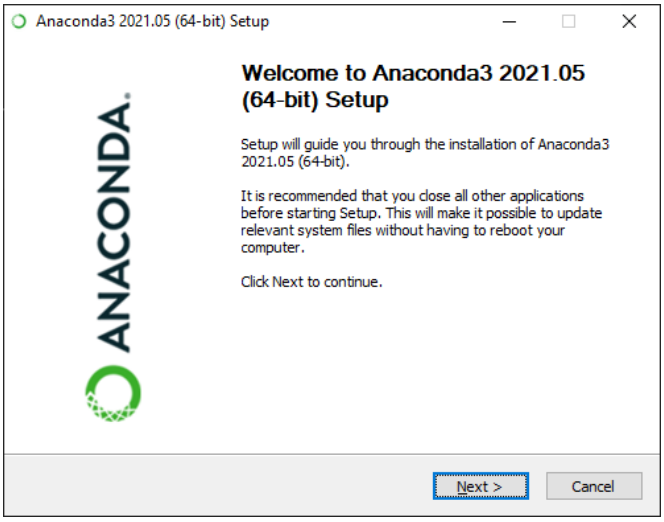


Ilustración 18. Instalación Anaconda

Se aconseja usar la opción de instalación para el usuario que tengamos activado en Windows, de esta forma evitaremos problemas en la resolución de rutas de directorio.

Ha de registrarse el software como entorno Python predeterminado y se activará la opción de añadir las modificaciones a la variable PATH.

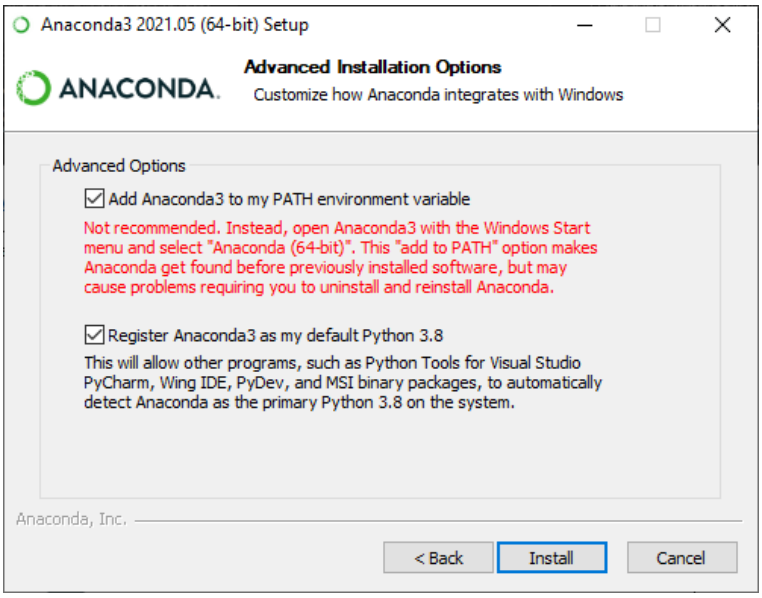


Ilustración 19. Instalación Anaconda

2.3.3 Cmake

Cmake es la utilidad de compilación multisistema más habitual. Conectará con el software Visual Studio desplegado previamente, para la compilación del software que necesitaremos posteriormente. Se descarga desde https://cmake.org/download/

La versión utilizada en nuestro proyecto es la 3.21.3.

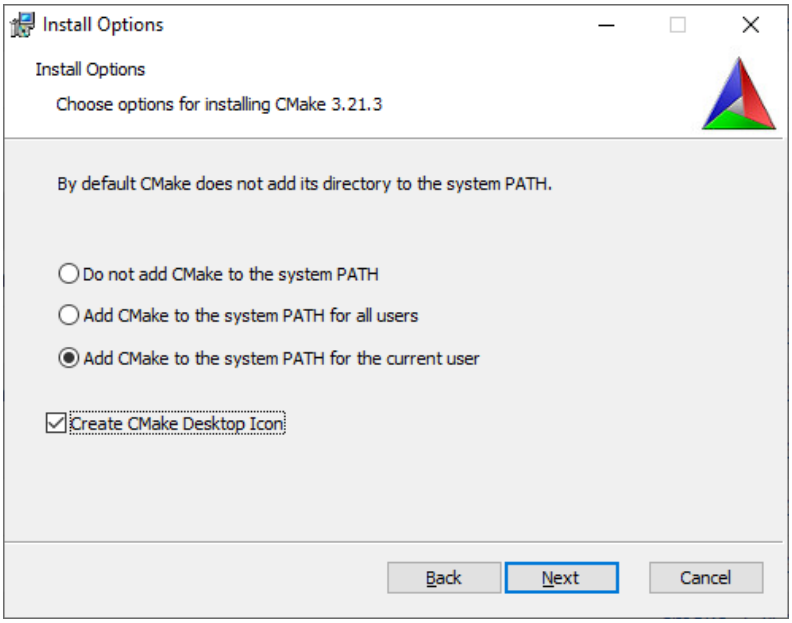


Ilustración 20. Instalación Cmake

2.3.4 Git

Git es uno de los sistemas más populares de control de versiones de *software*, con capacidad de descarga de los paquetes seleccionados. Se descarga desde https://git-scm.com/download/win

Utilizaremos las opciones de configuración que se muestran a continuación. La referencia completa de los pasos de instalación se añadido en el apartado de Anexos del presente trabajo.

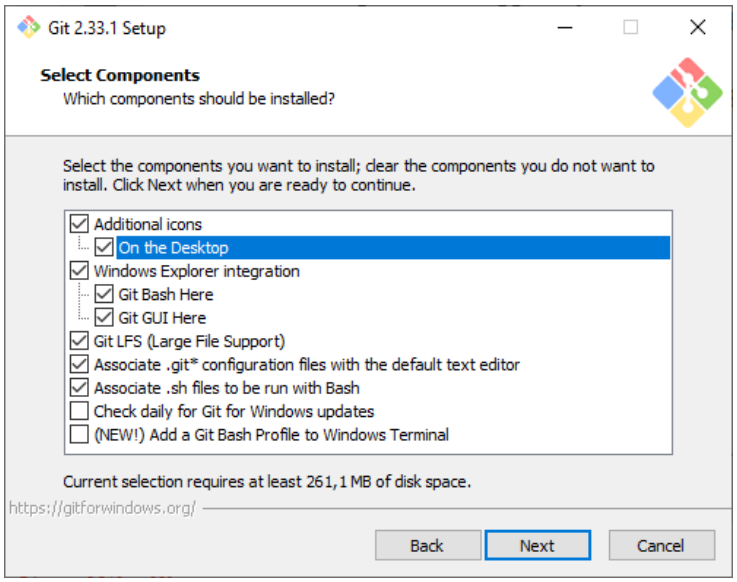


Ilustración 21. Instalación Git

2.3.5 Instalación Vcpkg

Se trata de un instalador de paquetes y mantenedor de dependencias de Microsoft para uso en Windows.

Para instalarlo clonaremos el repositorio usando Git:

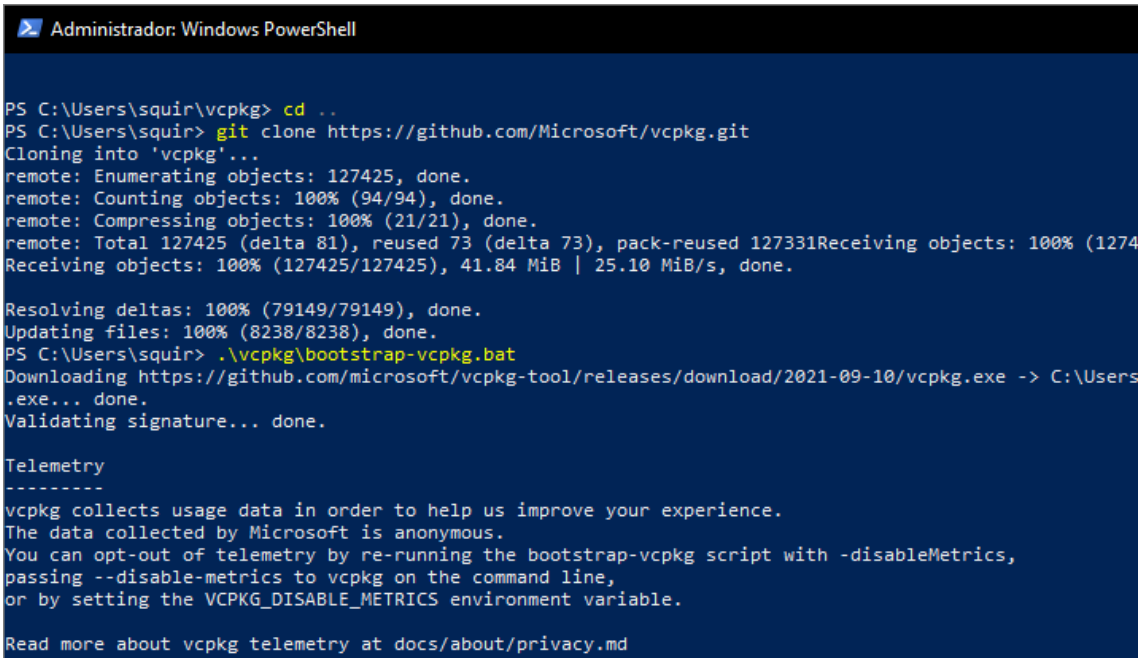


Ilustración 22. Instalación Vcpkg

Después comprobamos la instalación correcta instalando un paquete de ejemplo:

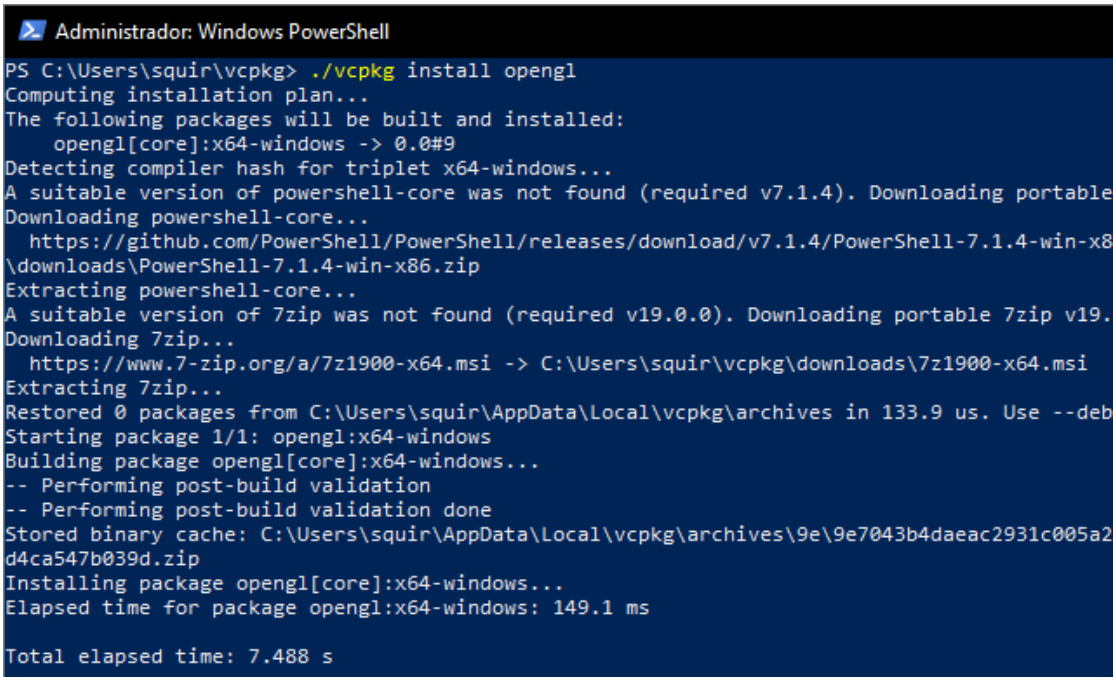


Ilustración 23. Instalación Vcpkg

Por último, añadiremos las siguientes variables de entorno a Windows:

- VCPKG_ROOT=C:\Users\squir\vcpkg
- VCPKG_DEFAULT_TRIPLET=x64-windows

2.3.6 NVIDIA CUDA

Se trata de una de las piezas más importantes de este proyecto. **CUDA** es el conjunto de librerías de **NVIDIA** que permiten que los programas compilados con soporte **CUDA** utilicen las unidades de ejecución o núcleos **CUDA** de las tarjetas gráficas, para procesamiento masivo en paralelo.

El uso de **CUDA** es lo que nos permite la capacidad de detección de placas de matrícula en tiempo real y uno de los motivos de tener que realizar compilaciones específicas para la consecución de los objetivos del proyecto.

El software se descarga desdehttps://developer.nvidia.com/cuda-downloads

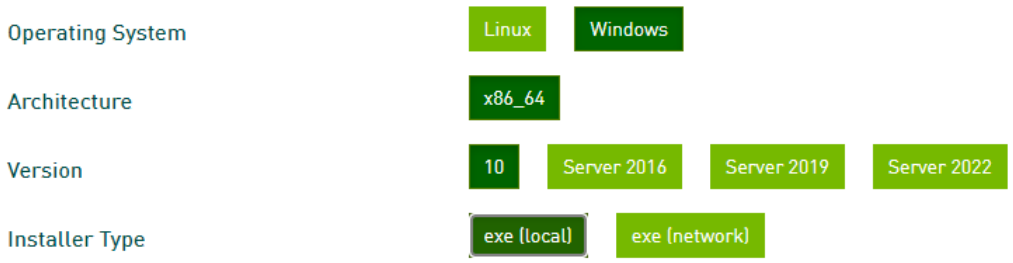


*Ilustración 24. Selección opciones **CUDA***

En nuestro caso instalaremos la versión 11.4.

*Ilustración 25. Instalación **CUDA***

Una vez instalado el paquete, es importante comprobar el correcto funcionamiento del mismo. Para ello realizamos una compilación de uno de los ejemplos que se suministran. En concreto compilaremos Smoke Particles, para lo que usaremos Visual Studio.

```
Salida
Mostrar salida de: Compilación
Operación Compilar iniciada..
1>------ Operación Compilar iniciada: proyecto: smokeParticles, configuración: Debug x64 ------
1>Compiling CUDA source file ParticleSystem_cuda.cu...
1>
1>C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.4\5_Simulations\smokeParticles>"C:\Program Files\NVIDIA GPU Computing Too
  gencode=arch=compute_35,code=\"sm_35,compute_35\" -gencode=arch=compute_37,code=\"sm_37,compute_37\" -gencode=arch=compute_5
  gencode=arch=compute_52,code=\"sm_52,compute_52\" -gencode=arch=compute_60,code=\"sm_60,compute_60\" -gencode=arch=compute_6
  gencode=arch=compute_70,code=\"sm_70,compute_70\" -gencode=arch=compute_75,code=\"sm_75,compute_75\" -gencode=arch=compute_8
  gencode=arch=compute_86,code=\"sm_86,compute_86\" --use-local-env -ccbin "C:\Program Files (x86)\Microsoft Visual Studio\201
  \x64" -x cu   -I./ -I../../common/inc -I./ -I"C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.4\/include" -I../../com
  Toolkit\CUDA\v11.4\include"  -G   --keep-dir x64\Debug  -maxrregcount=0  --machine 64 --compile -cudart static --threads 0 -
  EHsc /W3 /nologo /Od /Fdx64/Debug/vc142.pdb /FS /Zi /RTC1 /MTd " -o x64/Debug/ParticleSystem_cuda.cu.obj "C:\ProgramData\NVI
  \smokeParticles\ParticleSystem_cuda.cu"
1>CUDACOMPILE : nvcc warning : The 'compute_35', 'compute_37', 'compute_50', 'sm_35', 'sm_37' and 'sm_50' architectures are de
   -Wno-deprecated-gpu-targets to suppress warning).
1>particlesystem_cuda.cu
1>tmpxft_00003520_00000000-7_particlesystem_cuda.compute_86.cudafe1.cpp
1>Compilación del proyecto "smokeParticles_vs2019.vcxproj" terminada.
1>GLSLProgram.cpp
1>ParticleSystem.cpp
1>SmokeRenderer.cpp
1>SmokeShaders.cpp
1>framebufferObject.cpp
1>particleDemo.cpp
1>renderbuffer.cpp
1>Generando código...
1>   Creando biblioteca ../../bin/win64/Debug/smokeParticles.lib y objeto ../../bin/win64/Debug/smokeParticles.exp
1>smokeParticles_vs2019.vcxproj -> C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.4\bin\win64\Debug\smokeParticles.exe
========== Compilar: 1 correctos, 0 incorrectos, 0 actualizados, 0 omitidos ==========
```

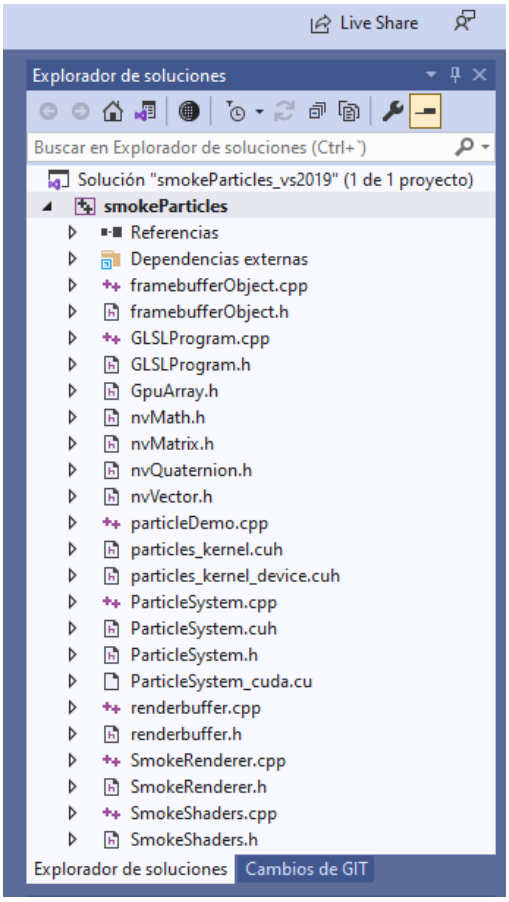


Ilustración 26. Compilación Smoke Particles

Si todo resulta correcto, veremos en pantalla una simulación con físicas basadas en modelos realistas, de una bola de humo desplazándose.

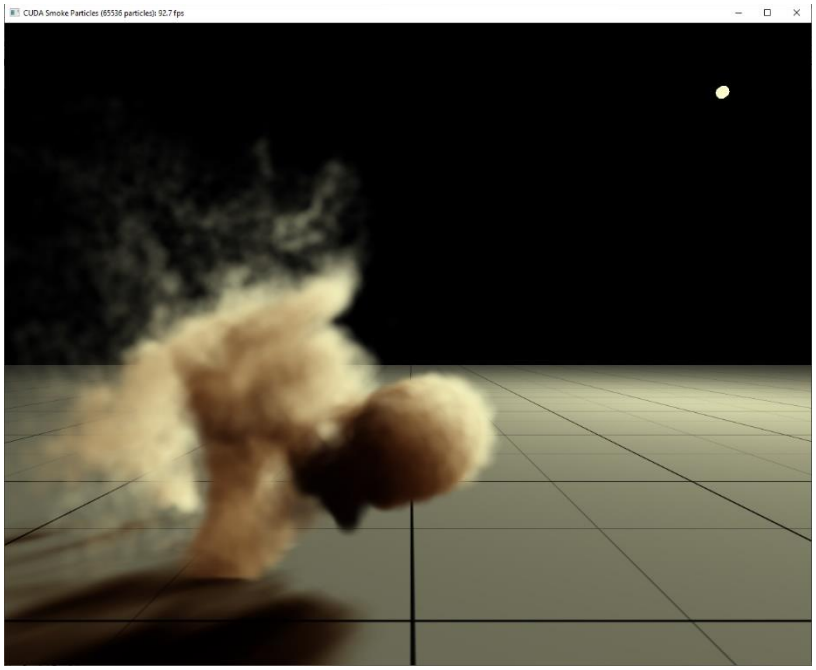


Ilustración 27. Resultado compilación Smoke Particles

2.3.7 cuDNN

Otra de las piezas clave para conseguir nuestro objetivo de procesamiento en tiempo real. Si **CUDA** nos da la capacidad de utilizar la tarjeta gráfica de **NVIDIA** para procesamiento paralelo, **cuDNN** nos va a permitir acelerar la computación basada en modelos neuronales mediante uso de las unidades de ejecución para procesar operaciones matemáticas basadas en tensores.

Sin el conjunto de las dos piezas, **CUDA** + **cuDNN**, estaríamos limitados a utilizar únicamente la capacidad de la CPU del ordenador portátil para el procesamiento de las imágenes.

Se descarga desde https://developer.nvidia.com/cudnn-download-survey

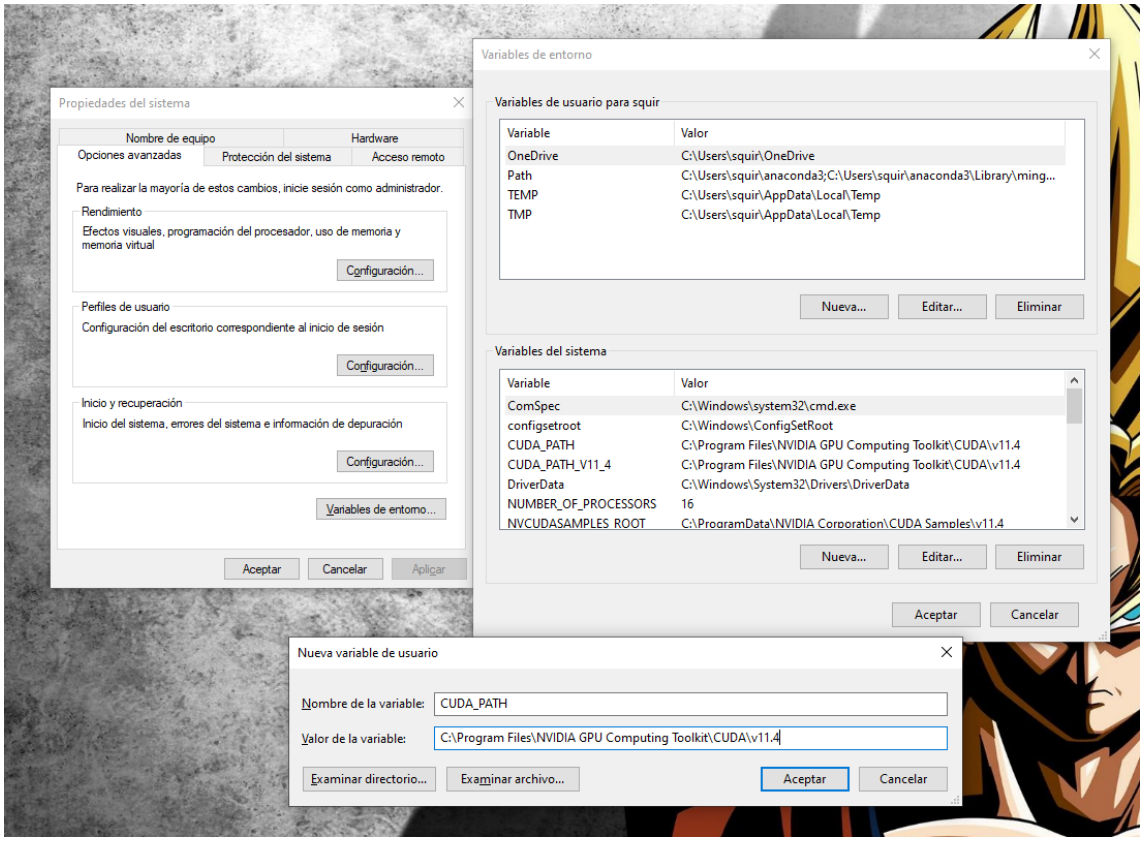
Es necesario crear un perfil previamente en el portal y contestar una encuesta para poder descargar el software. En nuestro caso, usaremos la versión 11.4. Es de vital importancia asegurar la compatibilidad de la versión **cuDNN** con la versión **CUDA**.

Una vez descargado, ha de descomprimirse y copiar los contenidos de sus carpetas de la siguiente forma:

- Copiar el contenido de la carpeta \cuda\bin a la carpeta bin dentro de C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.4
- Copiar el contenido de la carpeta \cuda\include a la carpeta include dentro de C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.4
- Copiar el contenido de la carpeta \cuda\lib\x64 a la carpeta lib\x64 dentro de C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.4

Será necesario crear una variable de entorno CUDA_PATH con valor igual a C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.4

En Windows esto puede realizarse escribiendo “Editar las variables de entorno del sistema” y ejecutando la aplicación y, posteriormente, modificando el aparto de Variables de entorno”.



2.3.8 OpenCV

Es uno de los softwares más importantes para conseguir la funcionalidad del proyecto. Se trata de una plataforma Open Source especializada en procesamiento de imagen que incluye numerosas características y una de las API más extensas y capaces de toda la industria.

Esta pieza es la más compleja de desplegar adecuadamente en Windows. Aunque es posible descargar librerías precompiladas de **OpenCV** en formato binario para Windows, estas librerías carecen de las capacidades de utilización de las librerías **CUDA** y **cuDNN** que hemos desplegado previamente y que son condición indispensable para poder tener procesamiento en tiempo real.

Por tanto, se hace necesario realizar una compilación específica de este *software* que nos permita utilizar **CUDA** y **cuDNN**. Este apartado es muy complejo, ya que no está correctamente documentado y, en la comunidad Open Source, hay múltiples procedimientos para realizar dicha compilación que, lamentablemente, son muy dependientes de la configuración y versionado de la plataforma específica Windows y de las versiones del software previo necesario que hemos ido instalando hasta este punto.

La compilación se produce en dos fases:

- Primero se utiliza la herramienta Cmake para gestionar todas las opciones de compilación necesarias. En este punto se especifican todos los modificadores que nos van a permitir emplear las librerías **CUDA** y **cuDNN**
- Posteriormente procedemos a la compilación propiamente dicha

El proceso de compilación es largo y tedioso. Cada compilación completa toma un tiempo superior a las 2 horas. En nuestro caso ha sido necesario realizar al menos 10 compilaciones diferentes para conseguir una compilación correcta que incorporase las opciones deseadas, dado que, como se ha indicado, existen numerosas variables que influencian el resultado exitoso de la compilación.

A continuación, se describen dos fórmulas de despliegue diferentes.

2.3.8.1 Instalación mediante Vcpkg

Este método es sencillo, pero tiene la limitación de que no genera las dependencias adecuadas para usar **OpenCV** desde Python, algo que es importante de cara a evoluciones futuras de la plataforma.

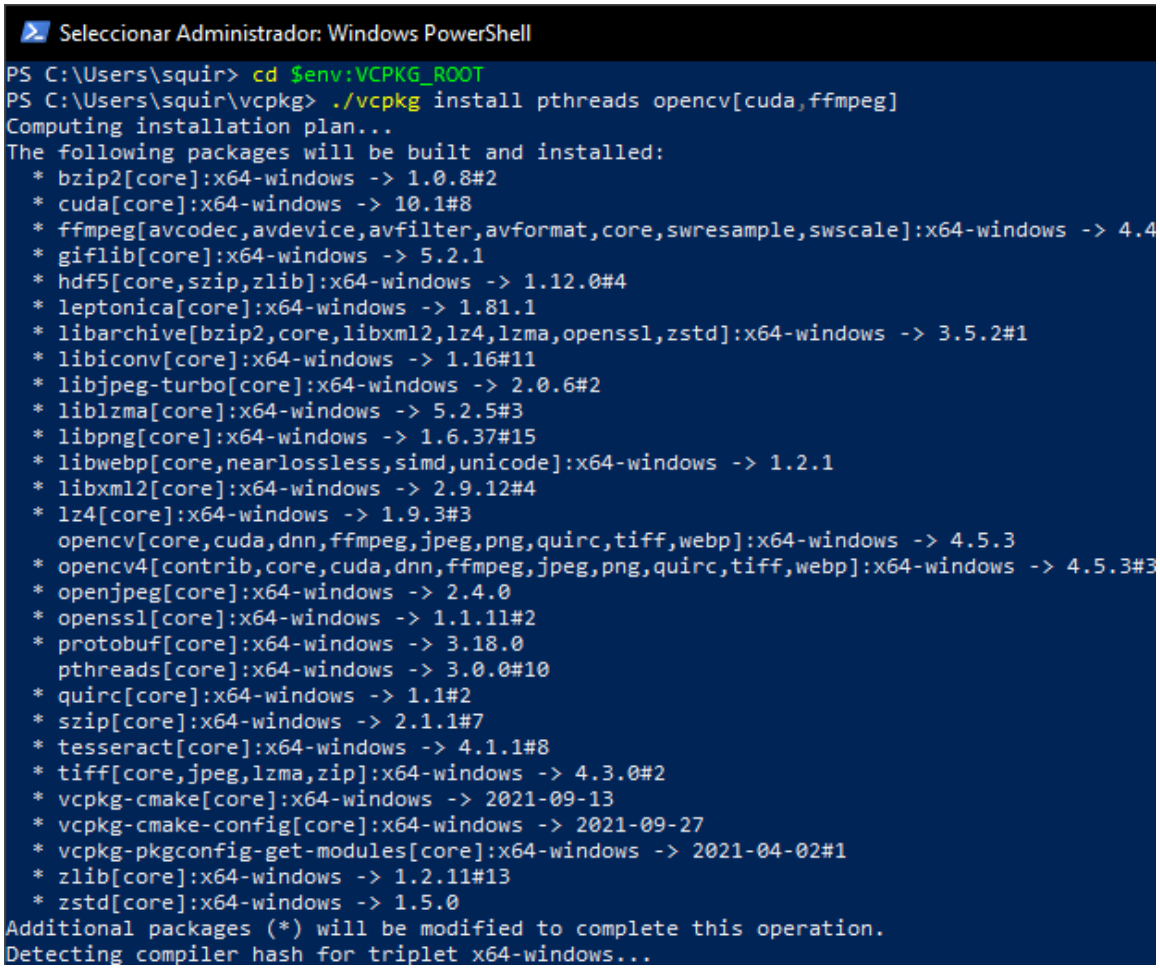


*Ilustración 28. Instalando **OpenCV***

Si todo va bien, veremos este resultado al final:

```
-- Extracting source C:/Users/squir/vcpkg/downloads/pthreads4w-code-v3.0.0.zip
-- Applying patch fix-arm-macro.patch
-- Applying patch use-md.patch
-- Using source at C:/Users/squir/vcpkg/buildtrees/pthreads/src/49e541b66c-53502e357
-- Building x64-windows-rel
-- Building x64-windows-rel done
-- Building x64-windows-dbg
-- Building x64-windows-dbg done
-- Installing: C:/Users/squir/vcpkg/packages/pthreads_x64-windows/share/pthreads/cop
-- Performing post-build validation
-- Performing post-build validation done
Stored binary cache: C:\Users\squir\AppData\Local\vcpkg\archives\aa\aa28d4031e76888b
5aae6a6c9dfb.zip
Installing package pthreads[core]:x64-windows...
Elapsed time for package pthreads:x64-windows: 17.6 s

Total elapsed time: 1.822 h

PS C:\Users\squir\vcpkg>
```

*Ilustración 29. Instalando **OpenCV***

Pese a la sencillez del proceso, **OpenCV** queda compilado para soporte **CUDA** y con la opción de aceleración de vídeo ffmpeg.

2.3.8.2 Instalación mediante compilación específica

Este procedimiento es tremendamente complejo y largo y puede requerir de múltiples compilaciones. El resultado merece la pena pues **OpenCV** queda disponible con todas las opciones necesarias, ligado a Python y con soporte **CUDA** y **cuDNNN.**

Descargaremos el paquete fuente de la versión 4.5.5 de OpenCV desde la página web oficial en https://github.com/opencv/opencv/archive/4.5.4.zip Lo descomprimimos en la carpeta opencv dentro de nuestro directorio de trabajo.

Descargamos el paquete de herramientas de terceros **OpenCV** contrib desde aquí: https://github.com/opencv/opencv_contrib/releases/tag/4.5.4 Lo descomprimimos en la carpeta opencv_contrib dentro del directorio de trabajo

```
Administrador: Windows PowerShell

PS C:\Windows\system32> cd C:\Users\
PS C:\Users> cd .\squir\
PS C:\Users\squir> cd .\opencv\
PS C:\Users\squir\opencv> cd build
```

*Ilustración 30. Instalando **OpenCV***

Procedemos ahora a crear el archivo de compilación con todas las opciones. Este paso es crítico y complejo ya que dependiendo de nuestra plataforma y dependencias puede dar muchos quebraderos de cabeza. Se aconseja usar PowerShell.

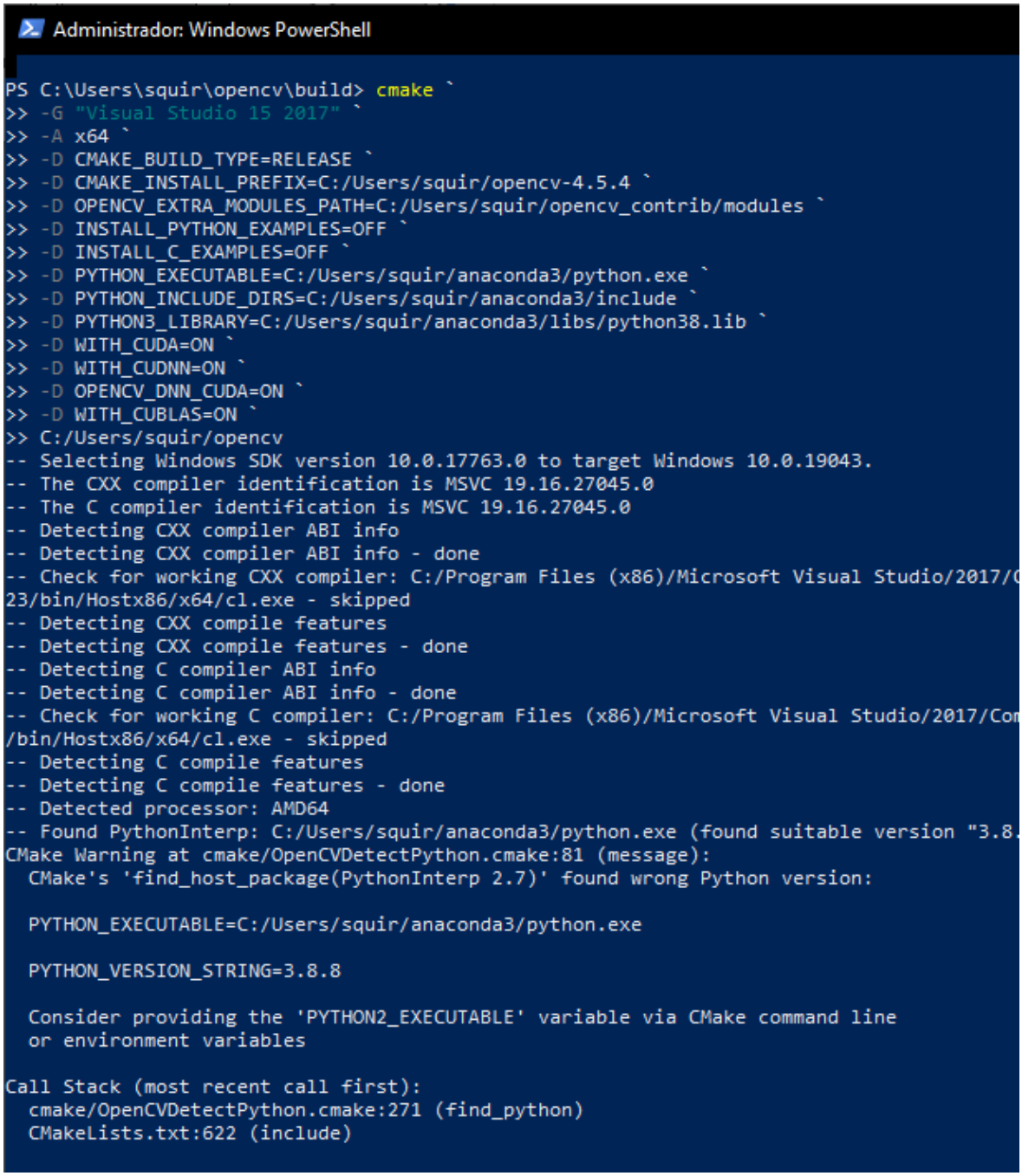


*Ilustración 31. Instalando **OpenCV***

El resultado será el siguiente:

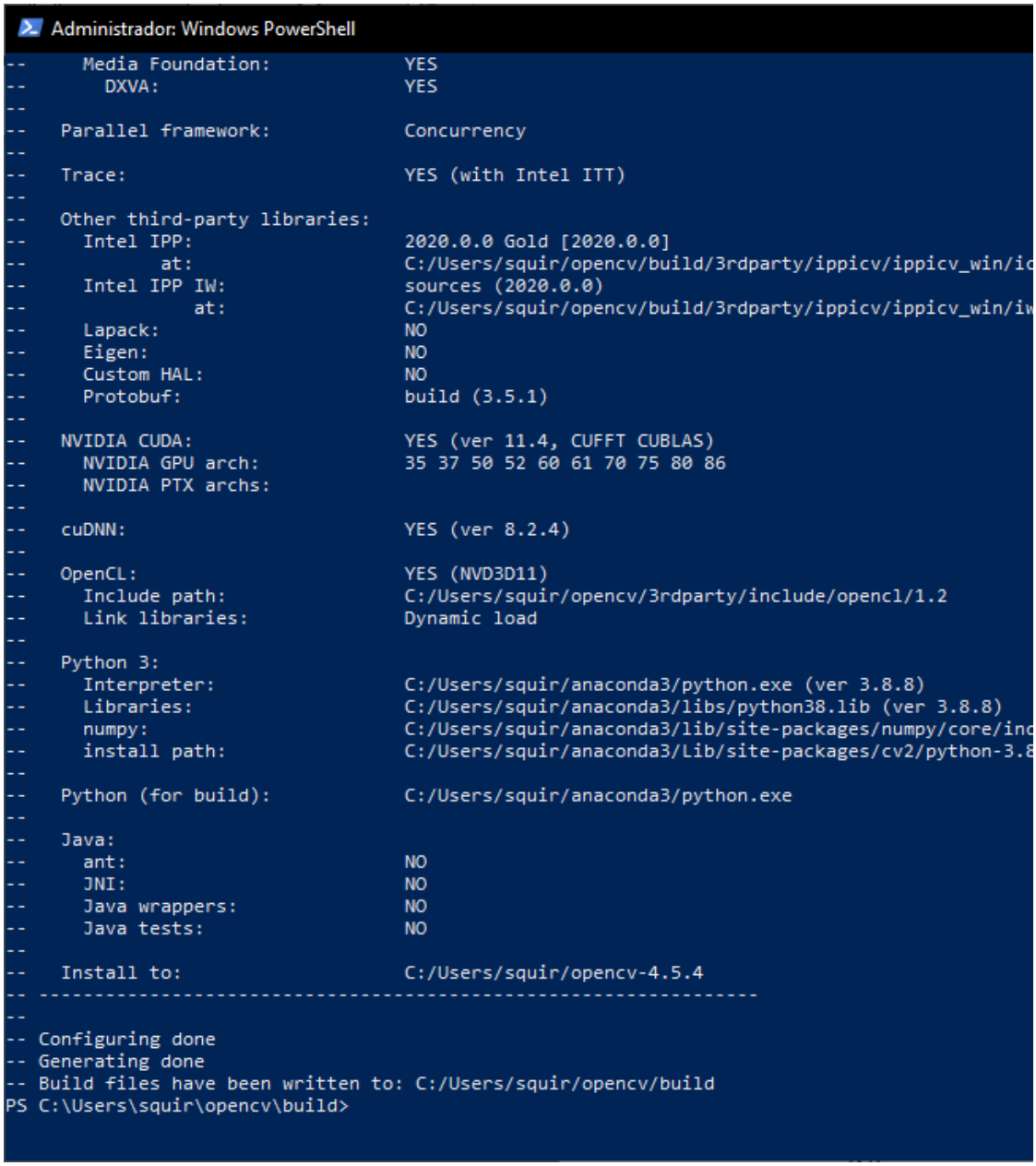


*Ilustración 32. Instalando **OpenCV***

Las opciones de configuración anteriores se detallan, por su importancia, en los anexos.

Finalmente procedemos a la compilación propiamente dicha con:

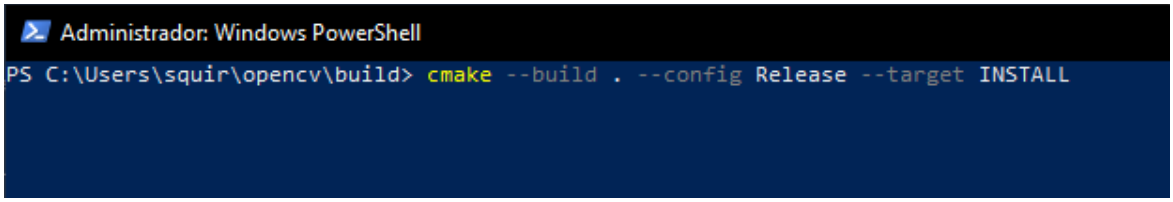


*Ilustración 33. Instalando **OpenCV***

2.3.9 Darknet

Es la otra pieza clave del proyecto. **Darknet** (Redmon & Farhadi, 2018) es una implementación en C/C++ del algoritmo **YOLO** de los mismos autores, basada en **Redes Neuronales Convolucionales**. El sistema **Darknet** es el software que nos permitirá hacer el entrenamiento de la red neuronal necesario para realizar la inferencia de detección del objeto paca de matrícula sobre imágenes y vídeos.

Darknet se apoya en **OpenCV** para una gran cantidad de procesos, de ahí la importancia del adecuado despliegue de **OpenCV**. El mecanismo de funcionamiento de **Darknet** se ha explicado en el capítulo anterior, en este punto nos centraremos únicamente en el despliegue de la plataforma.

La plataforma **Darknet** que se ha seleccionado es la evolución de la plataforma original de Redmon, que término su desarrollo en la versión 3. Usaremos la versión 4 (Bochkovskiy, Wang, & Mark Liao, 2020) que es accesible desde https://github.com/AlexeyAB/darknet

El proceso de compilación de Darknet es mucho más sencillo que el de **OpenCV**, aunque igualmente costoso en tiempo. Cada compilación de Darknet toma 5 horas en la máquina de ejecución del proyecto, habiendo sido necesarias al menos 5 compilaciones completas para integrar las opciones necesarias para el procesamiento en tiempo real. Principalmente, el proceso de las recompilaciones ha sido necesario para conseguir la integración completa de Darknet con **OpenCV**, con soporte para **CUDA** y **cuDNN**.

Primero elevamos los permisos de la sesión. Despues descargamos el paquete usando Git y posteriormente realizamos la compilación usando Vcpkg con todas las dependencias.

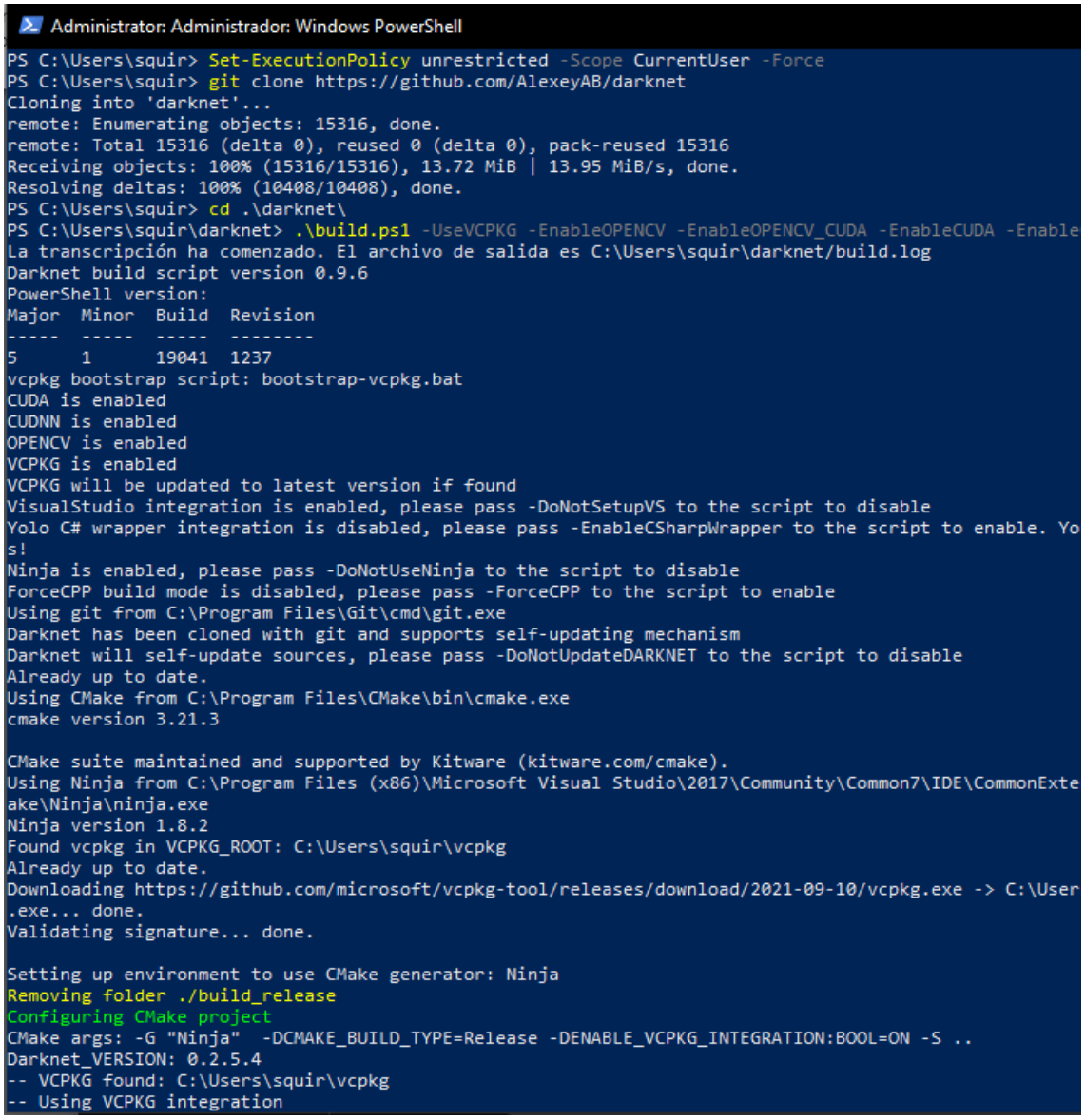


*Ilustración 34. Instalando **Darknet***

Después de varias horas de compilación, además de una gran sensación de felicidad, el resultado será el siguiente:

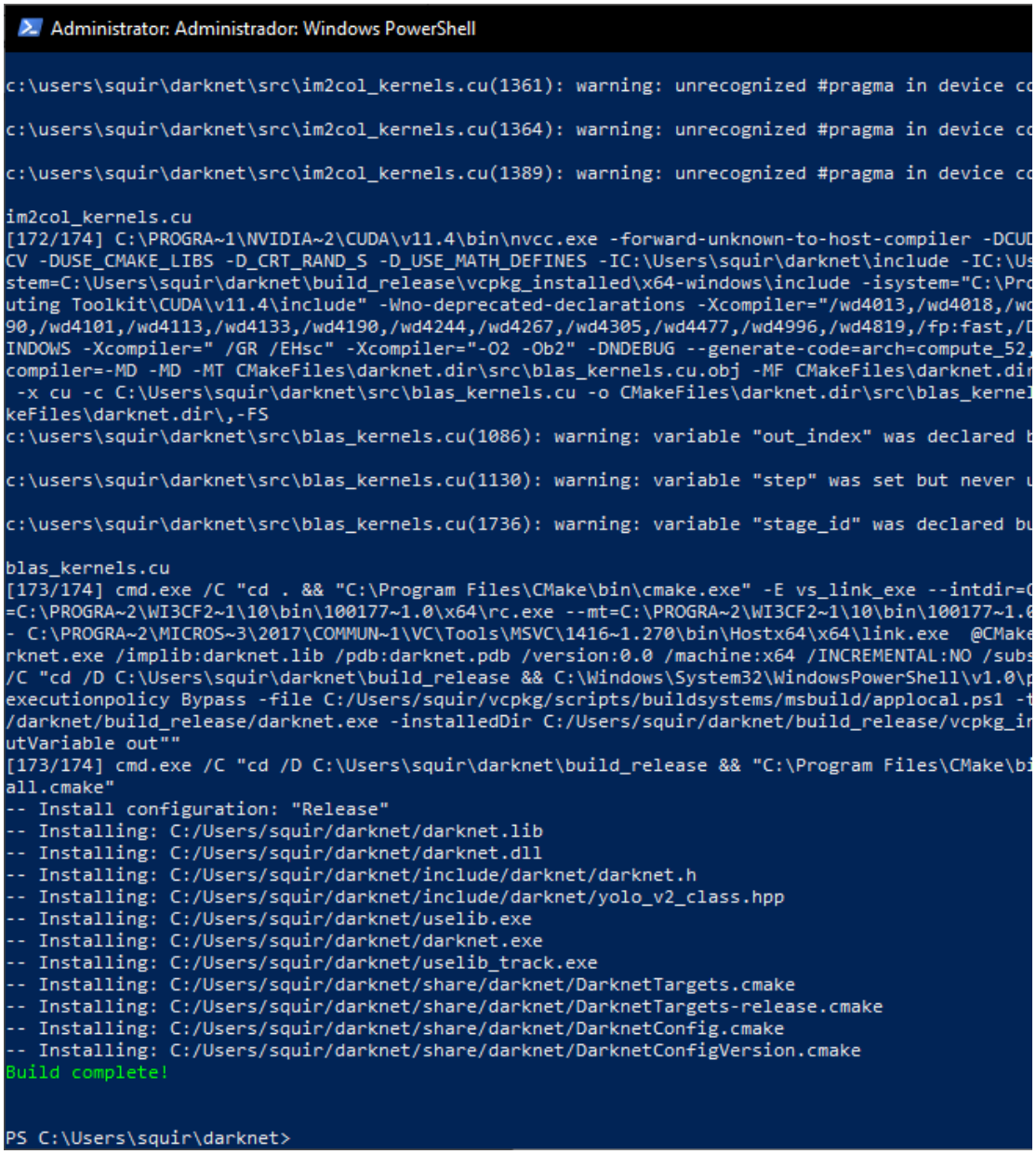


*Ilustración 35. Instalando **Darknet***

Por último, sólo nos queda comprobar la correcta compilación haciendo una pequeña prueba:

```
PS C:\Users\squir\darknet> ./darknet
usage: C:\Users\squir\darknet\darknet.exe <function>
PS C:\Users\squir\darknet> ./darknet imtest data/eagle.jpg
 CUDA-version: 11040 (11040), cuDNN: 8.2.4, GPU count: 1
 OpenCV version: 4.5.3
L2 Norm: 371.762238
-0.014967 0.933583 0.949526
PS C:\Users\squir\darknet>
```

*Ilustración 36. Instalando **Darknet***

El resultado en pantalla será similar a este, indicando que la plataforma funciona correctamente.

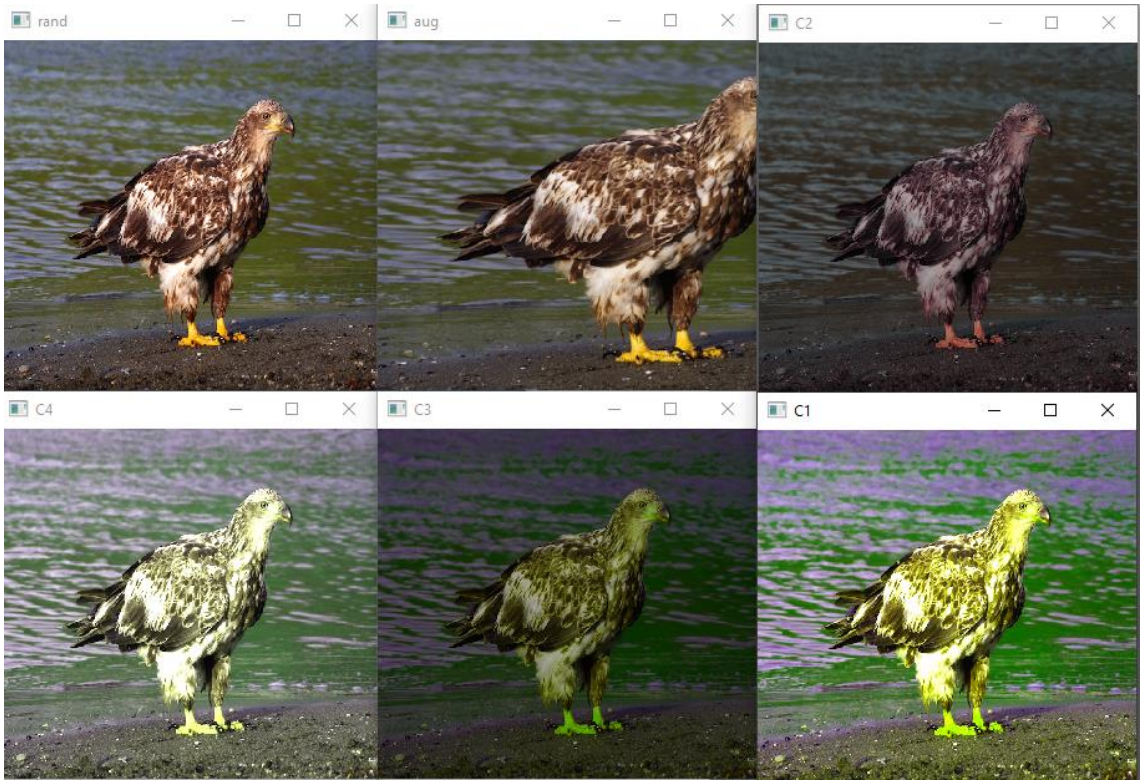


*Ilustración 37. Resultado de prueba compilación **Darknet***

CAPÍTULO 3. ENTRENAMIENTO DE LA PLATAFORMA

En entrenamiento es uno de los apartados más importantes en un sistema de aprendizaje. En el caso de un sistema de visión basado en redes neuronales, en esta fase identificamos de forma manual los objetos en el conjunto de imágenes de entrenamiento, con la meta de hacer conocer al sistema cuáles son las posibles salidas deseadas ante las entradas en el sistema.

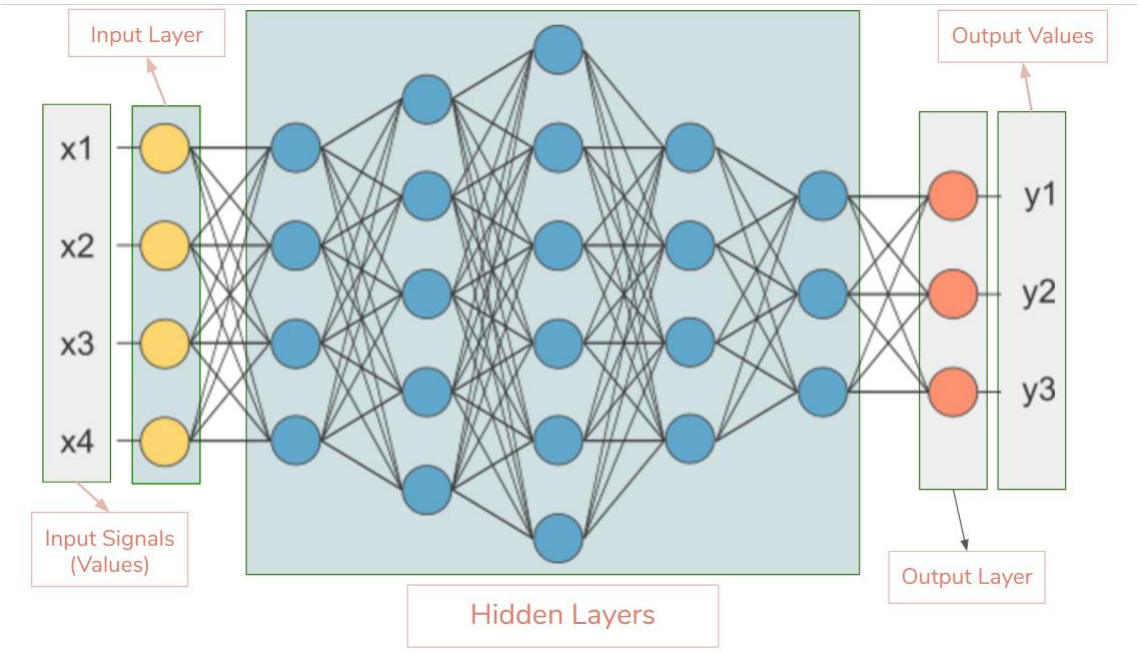


Ilustración 38. Modelo general de red neuronal (Ahirwar, 2020)

El esquema de utilización general de un sistema de **Machine Learning**, sería como sigue:

Ilustración 39. Las fases de solución de un problema con Machine Learning (Nelson, 2020)

En nuestro caso, tenemos claramente identificado el problema, que es el reconocimiento de placas de matrícula en tiempo real, por lo que llegamos al punto de proceder a las fases siguientes que describiremos a continuación.

3.1 Consecución de imágenes de entrenamiento

Para este apartado, se realizó un trabajo de campo de captura de 200 imágenes reales en entornos exteriores e interiores con una cámara. Dichas imágenes tienen una resolución de 72 dpi y un tamaño de 2752x2064 píxeles.

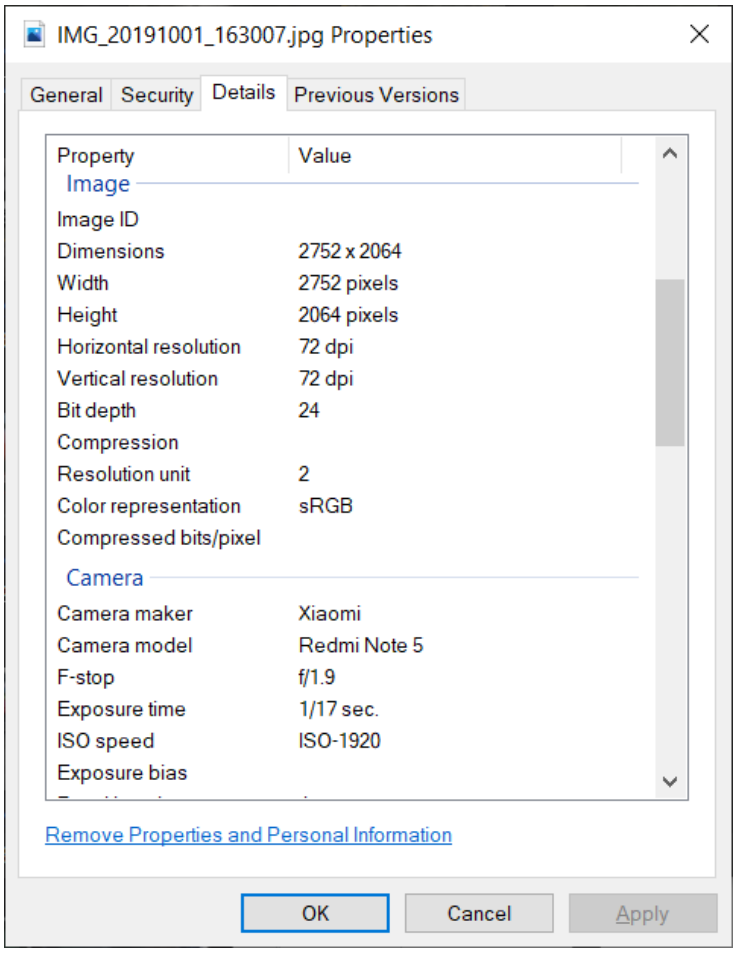


Ilustración 40. Características imágenes de entrenamiento

Las imágenes se tomaron en diferentes condiciones de iluminación y con múltiples objetos de entrenamiento (placas de matrícula) en cada una de ellas. El objetivo principal era conseguir la mayor diversidad posible de tamaños, ángulos y fondos alrededor las placas de matrícula. Este punto se explica en el siguiente apartado.

3.2 Preparación de los datos. "Cajeado" y etiquetado sobre las imágenes de entrenamiento

Una vez contamos con las imágenes de entrenamiento, el siguiente punto es identificar las porciones de estas que han de ser identificadas como placas de matrícula. Para ello, es necesario marcar dichas zonas a efectos de coordenadas, para que sean procesadas posteriormente en la fase de entrenamiento.

Para realizar el "cajeado", existen numerosas opciones, desde el uso de aplicaciones específicas a contratar un servicio en Internet de personas que realizarán el cajeado por nosotros a través de una transacción comercial (Anolytics, 2021).

En nuestro caso, se ha decidido utilizar una herramienta Open Source para este proceso. En concreto, utilizaremos la aplicación LabelImg, que puede descargarse desde https://github.com/tzutalin/labelImg

Ejecutaremos la herramienta desde Python y realizaremos los siguientes pasos de configuración en la interfaz gráfica:

- Seleccionar el directorio de origen "Open Dir". Es la carpeta en la que tenemos las imágenes a cajear.
- Seleccionaremos el directorio de destino "Change Save Dir", que es donde se almacenan las coordenadas de las etiquetas de los objetos que se vayan etiquetando.
- Definición de la etiqueta que usaremos para el objeto matrícula que, en este caso, hemos denominado "matrícula coche".
- Seleccionaremos el tipo de fichero de etiquetado de salida, que puede ser **YOLO** o **PascalVOC**. Seleccionaremos **YOLO**.

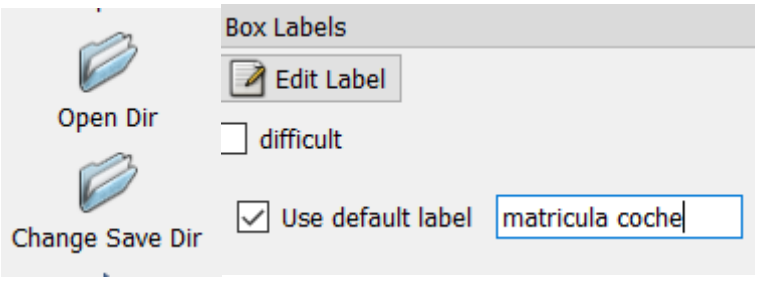


Ilustración 41. Opciones LabelImg

El cajeado propiamente dicho, se realiza pulsando sobre la opción "Create/nRectBox" y marcando los vértices de la caja alrededor de la placa de matrícula. Es muy importante que marquemos sólo el menor espacio posible que contenga la matrícula en su totalidad.

Ilustración 42. Ejemplo cajeado

Como se ha comentado previamente, se han realizado cajeados en todas las condiciones posibles, incluyendo desde el interior de vehículos e intentando que existan múltiples placas de matrícula simultáneamente, para aumentar la capacidad de entreno.

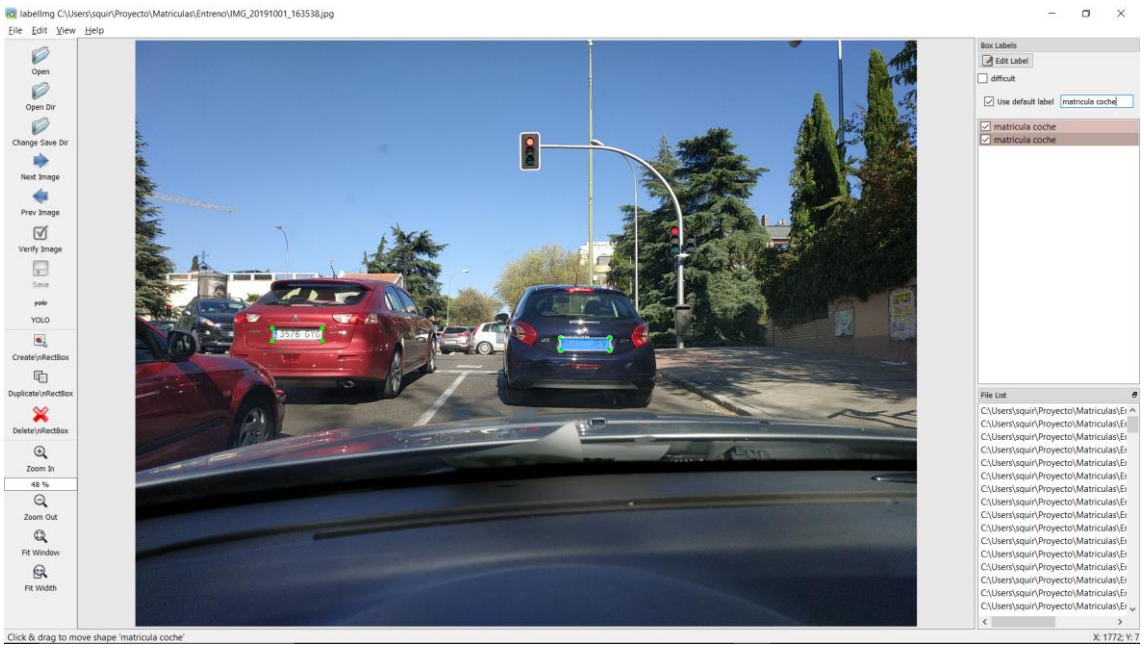


Ilustración 43. Ejemplo Cajeado

No cajearemos matrículas que apenas puedan ser reconocidas a simple vista o que aparezcan completas, para evitar confundir al sistema en la fase de entrenamiento.

Ilustración 44. Ejemplo cajeado

Por cada una de las 200 imágenes, se ha procedido a realizar un cajeado manual de todos los objetos de placa de matrícula existentes. El resultado de cajeado, son ficheros de texto que incluyen 4 datos numéricos por cada matrícula identificada manualmente en la imagen, correspondientes a los cuatro vértices de la caja. Un ejemplo concreto sería similar a este:

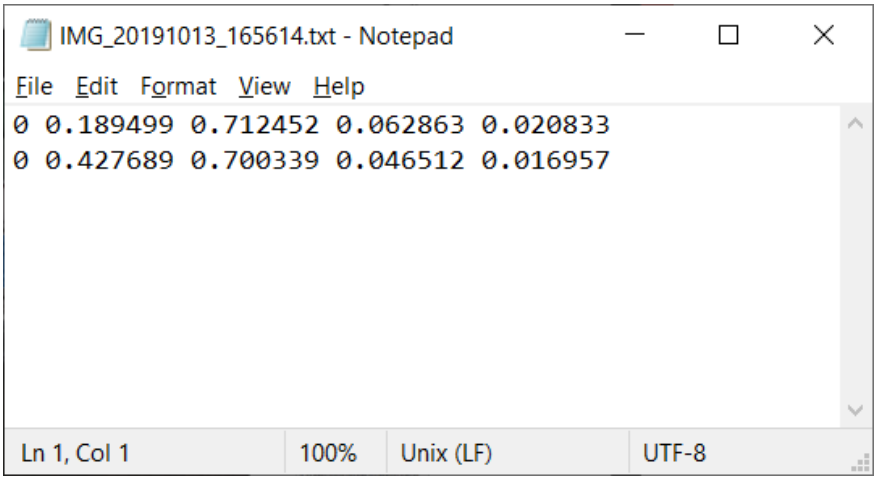


Ilustración 45. Coordenadas objetos en LabelImg

3.3 Entrenamiento del sistema

Una vez identificadas las zonas de las imágenes que queremos sean asociadas a objetos reconocibles para el sistema, el siguiente paso es el entrenamiento propiamente dicho del software. En esta fase, lo que conseguiremos es que la **Red Neuronal Multinivel** que genera la plataforma asigne los pesos adecuados para garantizar la diferenciación de las características de las imágenes y la posterior clasificación por las categorías definidas; para nuestro ejemplo “matricula coche”.

Para ello, se ha de definir un fichero de configuración de la plataforma denominado “yolo-obj.cfg”, con todas las opciones de configuración del proyecto neuronal. Este fichero se almacena en el directorio “cfg” dentro de la carpeta de instalación “darknet”. En el capítulo de Anexos, se aporta el fichero de configuración completo utilizado.

El procedimiento habitual de generación del fichero de configuración pasa por modificar un fichero previo del sistema, en el que se definen todas las opciones personalizadas para el proyecto.

Utilizaremos las siguientes configuraciones:

- batch = 64
- subdivisions = 32
- maxbatches = 2000
- steps = 1600,1800
- width = 416
- height = 416
- classes = 1 (repetir en cada una de las capas 3 capas [yolo])
- filters = 18 (en cada una las 3 capas [convolutional] antes de las capas [yolo] anteriores)

Adicionalmente generamos los ficheros “obj.names” y “obj.data” de la forma que se da a continuación. Estos ficheros han de almacenarse en la carpeta “data”:

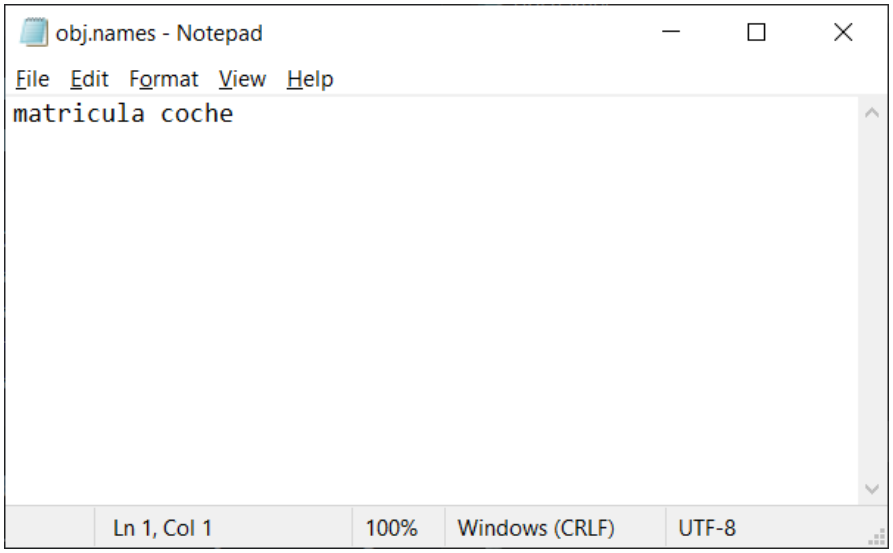


Ilustración 46. Fichero obj.names

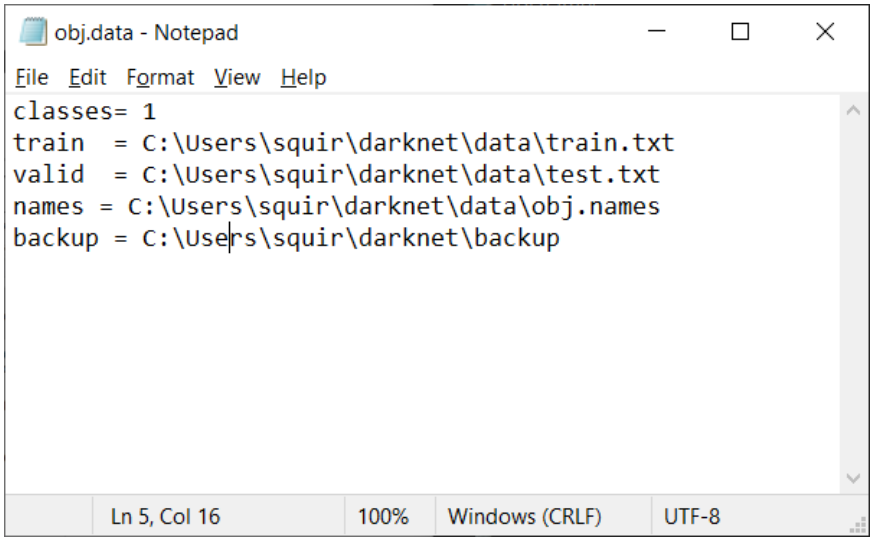


Ilustración 47. Fichero obj.data

En la carpeta "obj" dentro de la carpeta "data", incluiremos las imágenes de entrenamiento junto con los ficheros de etiquetado que hemos generado para ellas al utilizar LabelImg. De esta forma, cada fichero de imagen, ejemplo: imagen1.jpg, tendrá un fichero de tipo texto llamado de igual forma con extensión txt, donde se detallan las coordenadas de las placas de matrícula para el entrenamiento.

En la carpeta "data" generaremos un fichero de texto "train.txt" con las rutas a las imágenes de entreno relativas a **Darknet** de la siguiente forma:

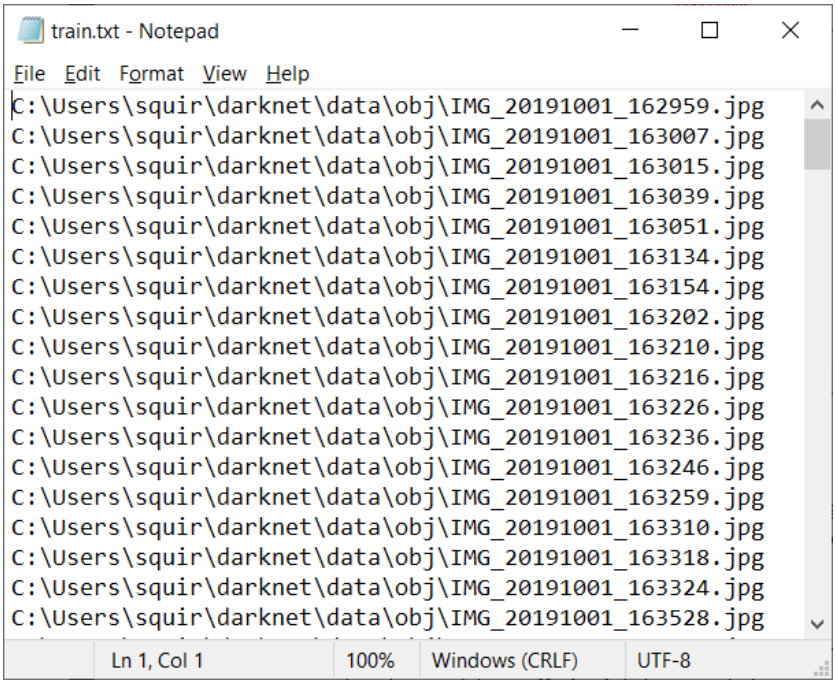


Ilustración 48. Fichero train.txt

Antes de proceder al entrenamiento, debemos establecer unos pesos por defecto para la red neuronal de Darknet. Esto se consigue descargando ficheros de pesos preentrenados sobre la base de datos COCO Dataset (COCO, 2021). Estos ficheros pueden descargarse al directorio Darknet y, dependiendo del número de capas neuronales que deseemos utilizar, podemos usar una de estas opciones:

- darknet53.conv.74 con 74 capas convolucionales, descargable de https://pjreddie.com/media/files/darknet53.conv.74
- yolo4.conv.137 con 137 capas convolucionales, descargable de https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137

En nuestro caso, el entrenamiento que mejores resultados ha dado para el procesamiento en tiempo real, ha sido el ejecutado sobre la base de darknet53.conv.74. Al tener un número menor de capas de procesamiento, la velocidad de inferencia es superior, garantizando un adecuado nivel de reconocimiento, como veremos en el siguiente capítulo.

Para el entrenamiento utilizamos el comando:

/darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.74

El proceso de entrenamiento es bastante lento. En la máquina de pruebas, el entrenamiento toma aproximadamente 10 horas. Se realizaron diversas pruebas de

entrenamiento, persiguiendo conseguir los mejores tiempos de inferencia con el objetivo de procesamiento en tiempo real.

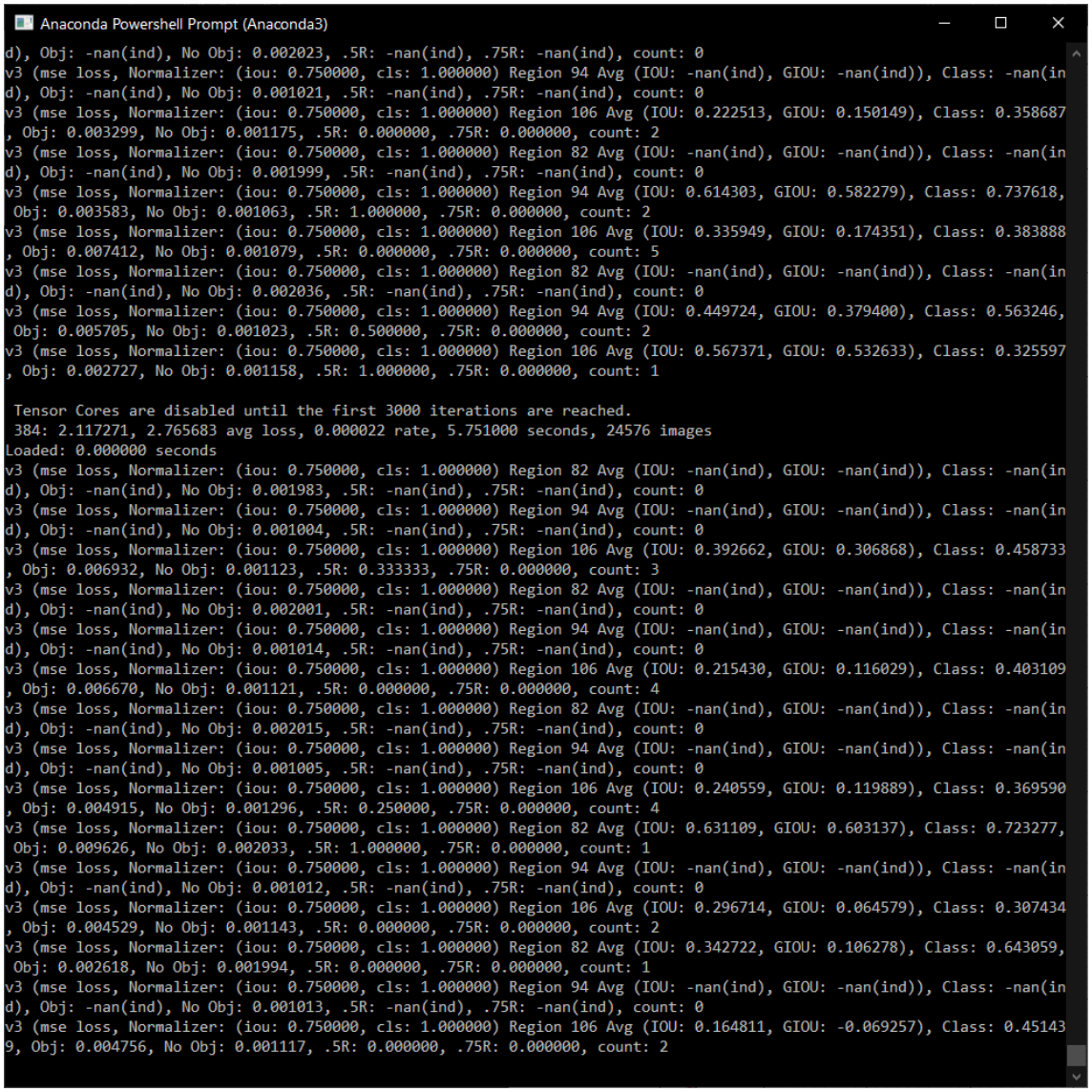


Ilustración 49. El entrenamiento del sistema es un proceso largo que toma 10 horas

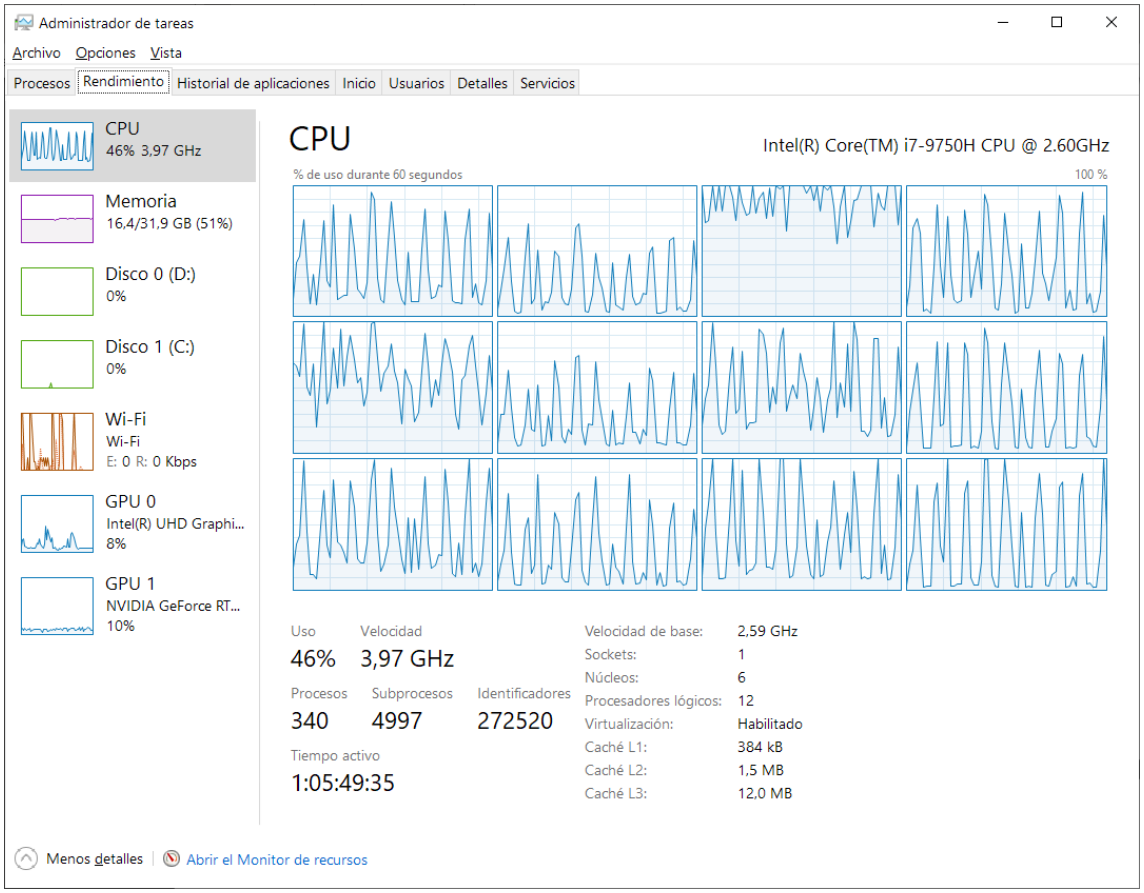


Ilustración 50. Carga de trabajo en CPU en tiempo de entrenamiento

El resultado del proceso de entrenamiento es el fichero “yolo-obj_last.weights” que se sitúa en la ruta “darknet\backup\”. Dicho fichero tiene un tamaño de 233 Mbytes.

CAPÍTULO 4. RESULTADOS Y EVALUACIÓN

Para evaluar el resultado del entrenamiento, se ha realizado un test sobre un conjunto de imágenes extraídas de Internet realizando una simple búsqueda de imágenes de Google.

4.1 Test con imágenes de prueba

Para el testeo utilizamos el comando:

./darknet.exe detector test C:\Users\squir\darknet\data\obj.data C:\Users\squir\darknet\cfg\yolo-obj.cfg C:\Users\squir\darknet\backup\yolo-obj_last.weights

Donde indicamos al sistema que vamos a proceder a la evaluación de imágenes de prueba mediante el fichero de resultado de entrenamiento previo "yolo-obj_last.weights".

Inmediatamente, el sistema procederá a cargar los valores de los pesos entrenados, dándonos la información completa de las capas convolucionales:

```
compute_capability = 750, cudnn_half = 1
  layer   filters  size/strd(dil)     input            output
  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
  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
  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
 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
 15 Shortcut Layer: 12
 16 conv    128      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 17 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 18 Shortcut Layer: 15
 19 conv    128      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 20 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 21 Shortcut Layer: 18
 22 conv    128      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 23 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 24 Shortcut Layer: 21
```

```
25 conv    128      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
26 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
27 Shortcut Layer: 24
28 conv    128      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
29 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
30 Shortcut Layer: 27
31 conv    128      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
32 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
33 Shortcut Layer: 30
34 conv    128      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
35 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
36 Shortcut Layer: 33
37 conv    512      3 x 3/ 2    52 x  52 x 256 ->   26 x  26 x 512 1.595 BF
38 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
39 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
40 Shortcut Layer: 37
41 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
42 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
43 Shortcut Layer: 40
44 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
45 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
46 Shortcut Layer: 43
47 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
48 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
49 Shortcut Layer: 46
50 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
51 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
52 Shortcut Layer: 49
53 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
54 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
55 Shortcut Layer: 52
56 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
57 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
58 Shortcut Layer: 55
59 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
60 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
61 Shortcut Layer: 58
62 conv   1024      3 x 3/ 2    26 x  26 x 512 ->   13 x  13 x1024 1.595 BF
63 conv    512      1 x 1/ 1    13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
64 conv   1024      3 x 3/ 1    13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
65 Shortcut Layer: 62
66 conv    512      1 x 1/ 1    13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
67 conv   1024      3 x 3/ 1    13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
68 Shortcut Layer: 65
69 conv    512      1 x 1/ 1    13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
70 conv   1024      3 x 3/ 1    13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
71 Shortcut Layer: 68
72 conv    512      1 x 1/ 1    13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
73 conv   1024      3 x 3/ 1    13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
74 Shortcut Layer: 71
75 conv    512      1 x 1/ 1    13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
```

```
 76 conv   1024      3 x 3/ 1    13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
 77 conv    512      1 x 1/ 1    13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
 78 conv   1024      3 x 3/ 1    13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
 79 conv    512      1 x 1/ 1    13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
 80 conv   1024      3 x 3/ 1    13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
 81 conv     18      1 x 1/ 1    13 x  13 x1024 ->   13 x  13 x  18 0.006 BF
 82 yolo
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
 83 route  79
 84 conv    256      1 x 1/ 1    13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
 85 upsample             2x    13 x  13 x 256 ->   26 x  26 x 256
 86 route  85 61
 87 conv    256      1 x 1/ 1    26 x  26 x 768 ->   26 x  26 x 256 0.266 BF
 88 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
 89 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
 90 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
 91 conv    256      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
 92 conv    512      3 x 3/ 1    26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
 93 conv     18      1 x 1/ 1    26 x  26 x 512 ->   26 x  26 x  18 0.012 BF
 94 yolo
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
 95 route  91
 96 conv    128      1 x 1/ 1    26 x  26 x 256 ->   26 x  26 x 128 0.044 BF
 97 upsample             2x    26 x  26 x 128 ->   52 x  52 x 128
 98 route  97 36
 99 conv    128      1 x 1/ 1    52 x  52 x 384 ->   52 x  52 x 128 0.266 BF
100 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
101 conv    128      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
102 conv    256      3 x 3/ 1    52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
103 conv     18      1 x 1/ 1    52 x  52 x 256 ->   52 x  52 x  18 0.025 BF
104 conv     18      3 x 3/ 1    52 x  52 x  18 ->   52 x  52 x  18 0.016 BF
105 conv     18      1 x 1/ 1    52 x  52 x  18 ->   52 x  52 x  18 0.002 BF
106 yolo
[yolo] params: iou loss: mse, iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
Total BFLOPS 63.535
 Allocate additional workspace_size = 52.43 MB
Loading weights from C:\Users\squir\darknet\backup\yolo-obj_last.weights...
 seen 64
Done! Loaded 107 layers from weights-file
```

4.2 Resultados obtenidos

Como se observa a continuación, los resultados son muy buenos. En la mayor parte de los casos, se detectan las placas de matrícula en su totalidad, existiendo únicamente en algunos casos, tamaños de caja superiores a lo deseable.


matricula coche
3685 HDP


matricula coche
0807 HTT


matricula coche
0019 GVM


matricula coche
2093 GSW


matricula coche
0724 HPH


matricula coche
4273 GFK


matricula coche
0411 KMT


matricula coche
7466 HWJ

Ilustración 51. Conjunto de imágenes de prueba

Con las imágenes de prueba, el tiempo medio de inferencia por cada una de las imágenes de prueba es de 29 milisegundos. Con este dato, podemos confirmar que se cumple el objetivo de procesamiento en tiempo real.

Para poder cumplir un objetivo de inferencia suficiente para procesar 24 *fps*, sería necesario procesar una imagen cada 41 milisegundos. Si añadimos a este número un 10% de margen extra en concepto de latencia de procesamiento entre fotogramas, nuestro objetivo de procesamiento se sitúa en 37 milisegundos. Por tanto, podemos afirmar que el procesamiento en tiempo real para la detección de placas de matrícula es posible sobre la base de la plataforma propuesta.

Tanto es así, que incluso podríamos cubrir una tasa de frames de 30 *fps*, que necesitarían 30 milisegundos de tiempo de procesamiento por *frame*, incluido el margen de seguridad del 10%.

Pasemos a evaluar el funcionamiento del sistema sobre un vídeo extraído de YouTube, en el que se recorren las calles de Madrid. Se trata de un vídeo en calidad Full HD 1080p con una tasa de bits de 127 Kbps. El vídeo es accesible desde https://www.youtube.com/watch?v=3QjVRzQjwYE

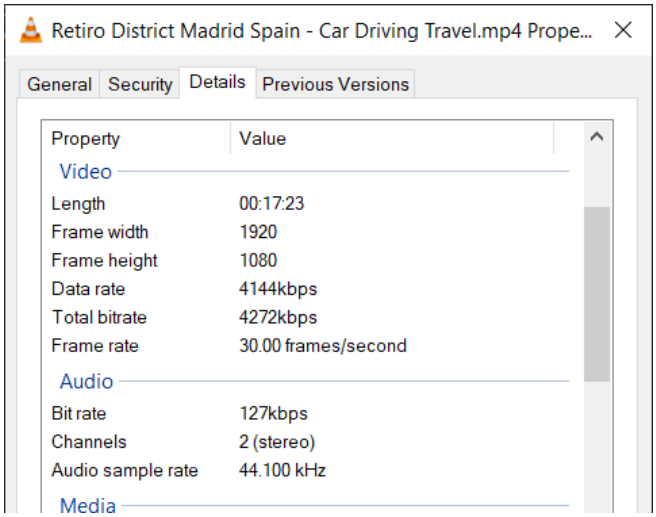
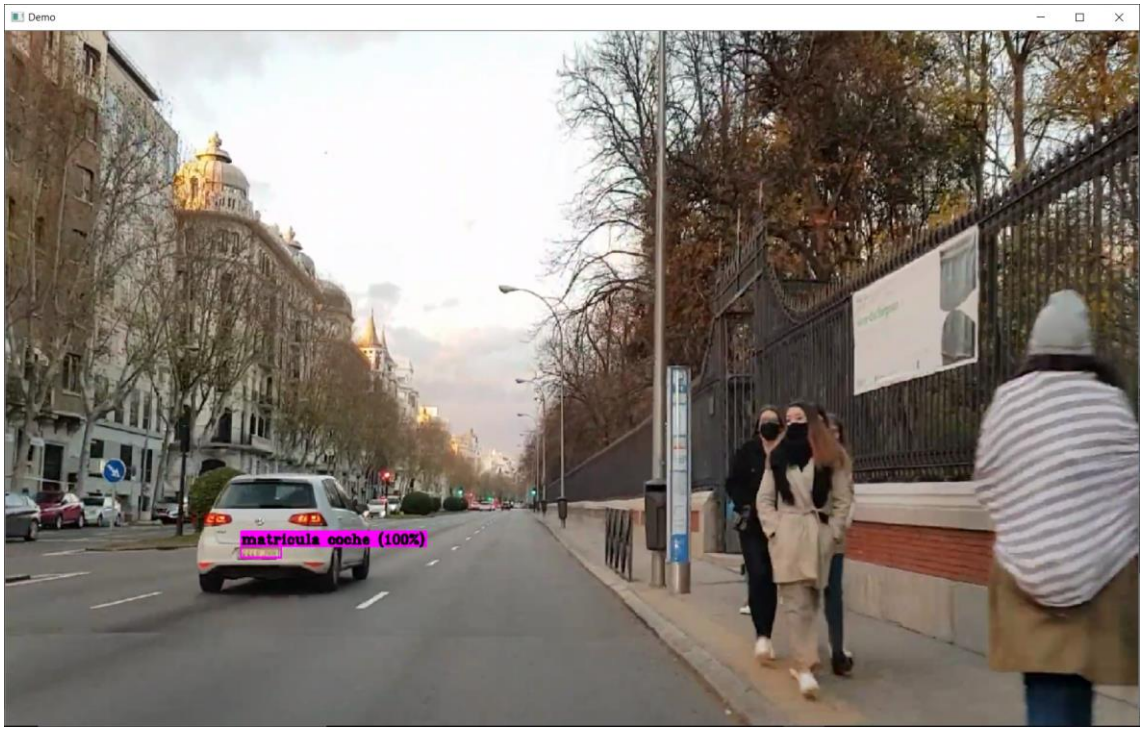
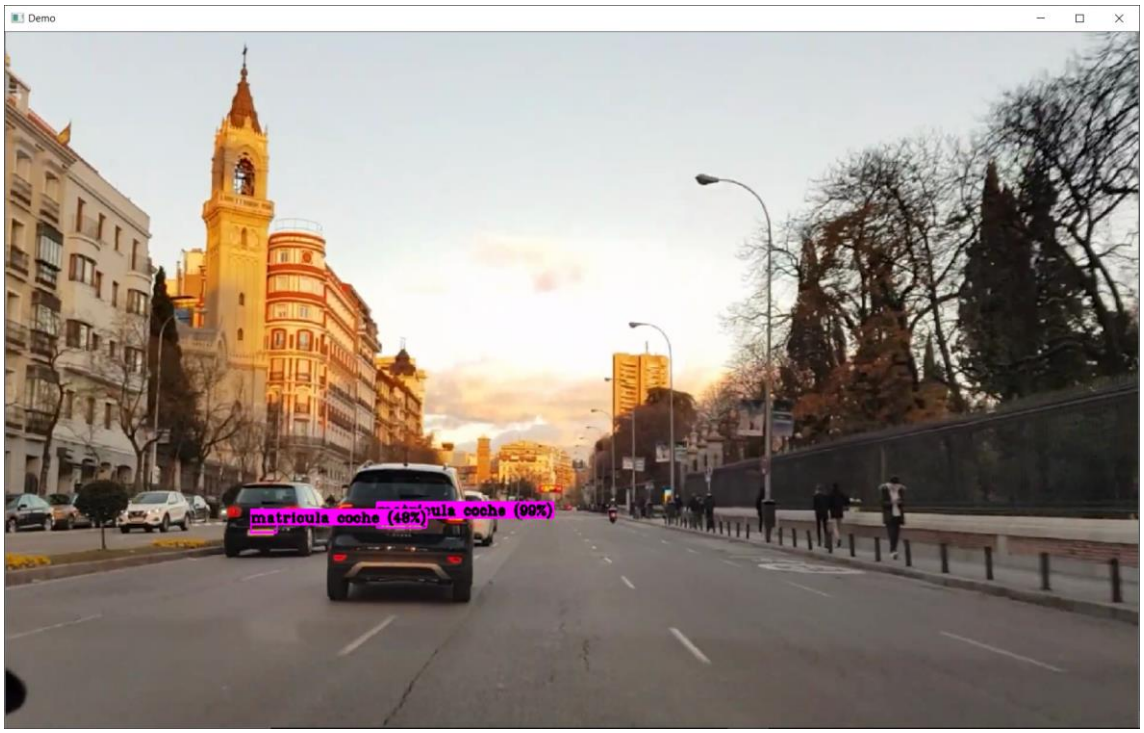


Ilustración 52. Características vídeo de prueba

Los resultados son excelentes. Con los tiempos de procesamiento indicados previamente, se confirma la capacidad de detectar las placas de matrícula, incluso sobre un vídeo de elevada tasa de bits en alta resolución. En las siguientes imágenes se muestran ejemplos de captura extraídos sobre el procesamiento del vídeo en tiempo real.


Demo
matricula coche (100%)


Demo
matricula coche (48%)
matricula coche (99%)

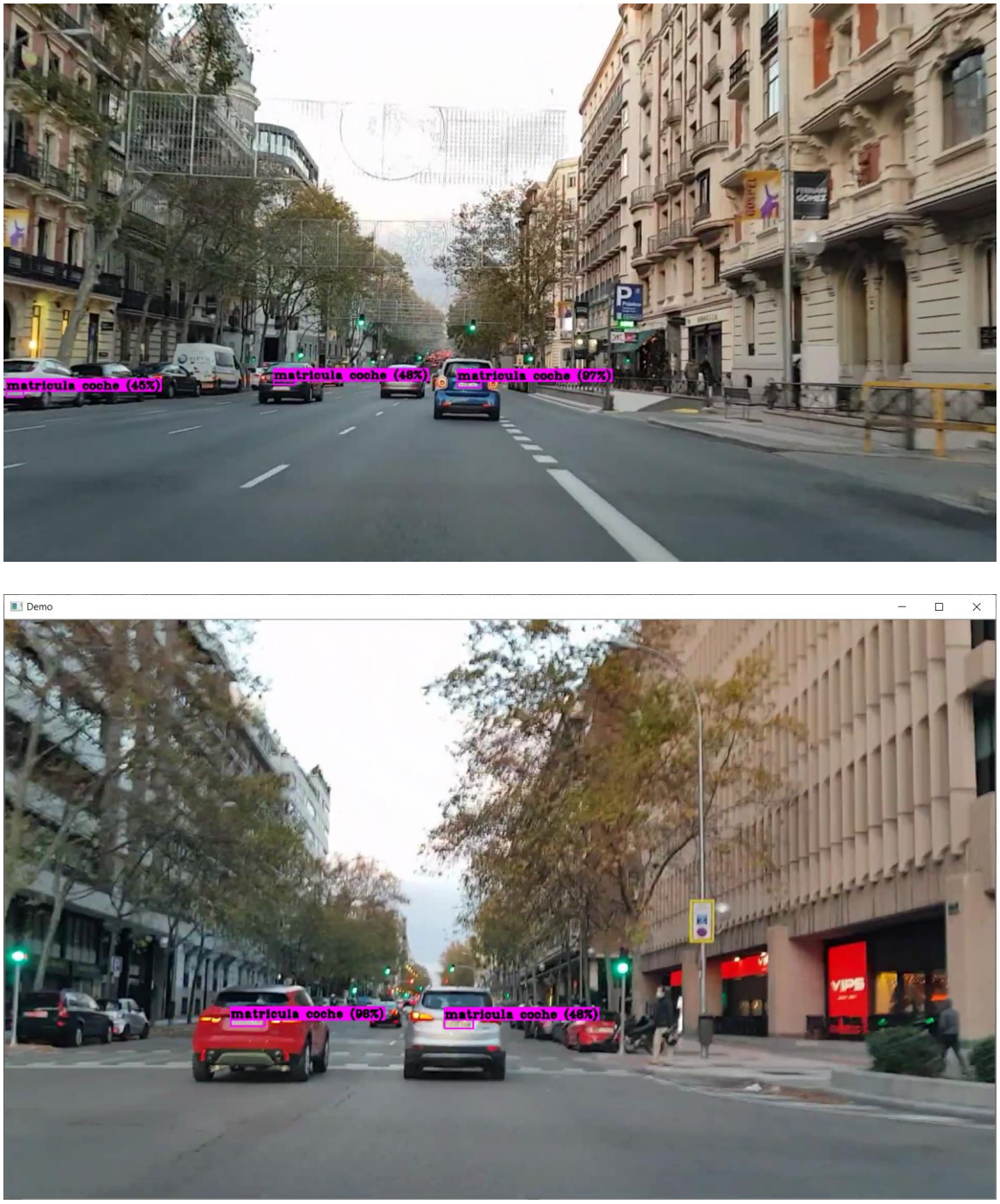


Ilustración 53. Ejemplos de inferencia sobre vídeo

4.2.1 Falsos positivos

En general, el sistema es bastante inmune a falsos positivos. La mayor parte del tiempo, no existen falsos positivos sb

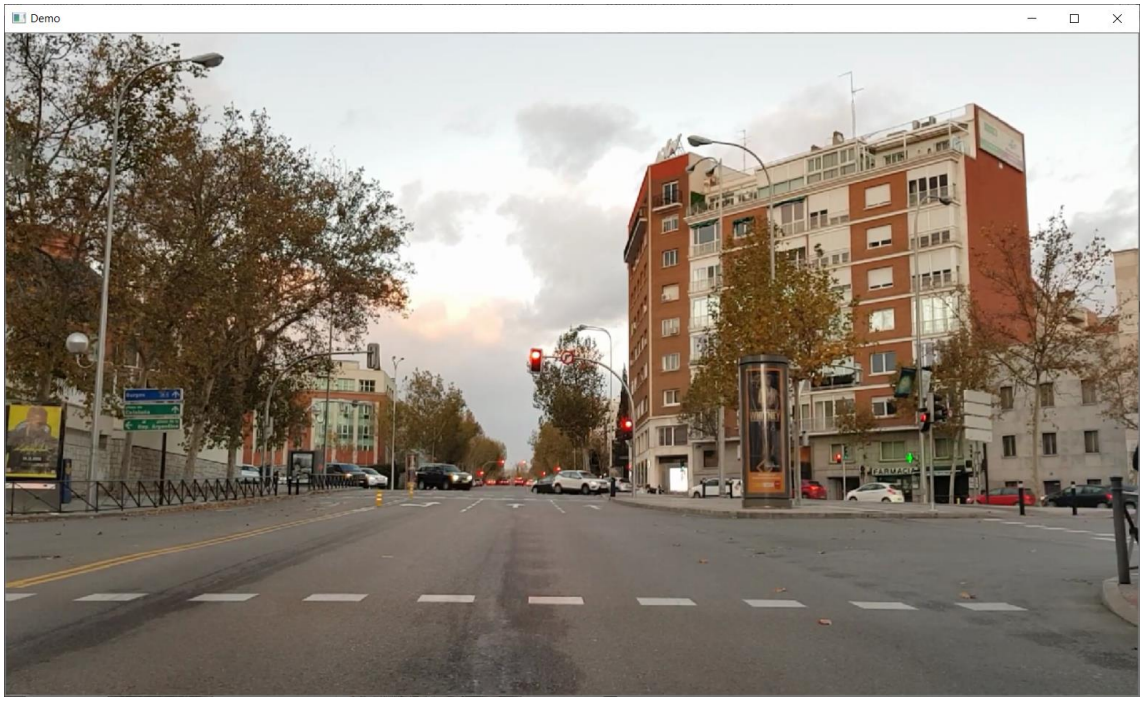


Ilustración 54. Ejemplo ausencia de falsos positivos

En momentos puntuales pueden aparecer algunos falsos positivos como en los ejemplos siguientes:

Ilustración 55. Falso positivo matrícula duplicada en coche de policía

Ilustración 56. Falso positivo en vídeo

4.2.2 Falsos negativos

Los falsos negativos son prácticamente inexistentes. En esta imagen puede apreciarse una matrícula que no es detectada en la zona central, y algunas en las zonas laterales, pero dichas placas son igualmente difíciles de percibir al ojo humano.

Adicionalmente, hay que tender en cuenta que las imágenes de video en movimiento, por la forma en la que son captadas y almacenadas, llevan implícito un grado de *motion blur* o efecto borroso debido al movimiento, que dificulta las tareas de inferencia.

4.2.3 Discusión

Los resultados son muy positivos, no solamente se detectan las regiones de pantalla asociadas a placas de matrícula, sino que se garantiza la capacidad de procesamiento en tiempo real.

Para mejorar los resultados, sería necesario realizar un entrenamiento sobre una base de imágenes superior e, incluso, realizar el entrenamiento sobre *frames* individuales en secuencias de vídeo, para mejorar el reconocimiento en situaciones con *motion blur*.

CAPÍTULO 5. CONCLUSIONES

El presente trabajo demuestra que es posible realizar reconocimiento de objetos en tiempo real y, en particular, placas de matrícula de vehículos, sin utilizar un hardware especialmente potente. Esto abre la puerta a su utilización en innumerables proyectos, ya sea en vigilancia, sistemas de guiado, aplicaciones industriales, etc.

No obstante, la plataforma demostrada está lejos de ser "productivizada". Sería necesario comprobar su robustez y probar la capacidad de construir aplicaciones por encima de ella con enfoque comercial.

En cuanto al uso de la licencia, el autor escribe lo siguiente de forma muy simpática:

```
YOLO LICENSE
Version 1, July 10 2015
Version 2, July 29 2016

THIS SOFTWARE LICENSE IS PROVIDED "ALL CAPS" SO THAT YOU KNOW IT IS SUPER
SERIOUS AND YOU DON'T MESS AROUND WITH COPYRIGHT LAW BECAUSE YOU WILL GET IN
TROUBLE HERE ARE SOME OTHER BUZZWORDS COMMONLY IN THESE THINGS WARRANTIES
LIABILITY CONTRACT TORT LIABLE CLAIMS RESTRICTION MERCHANTABILITY SUBJECT TO
THE FOLLOWING CONDITIONS:

1. #yolo
2. #swag
3. #blazeit
LIABILITY CONTRACT TORT LIABLE CLAIMS RESTRICTION MERCHANTABILITY. NOW HERE'S
THE REAL LICENSE:

0. Darknet is public domain.
1. Do whatever you want with it.
2. Stop emailing me about it!
```

Ilustración 57. Licencia YOLO – Darknet

Por tanto, el sistema podría ser utilizado de forma comercial, en caso de plantearse un caso de negocio adecuado.

5.1 Aspectos éticos

Precisamente por motivos éticos, el autor original de YOLO – Darknet, Joseph Redmon, ha abandonado el desarrollo del software en la versión 3, al indicar que sentía miedo de cómo podrían utilizarse estas tecnologías en el futuro. La versión 4 y posteriores están siendo desarrolladas y mantenidas bajo el liderato de Alexey Bochkovskiy.

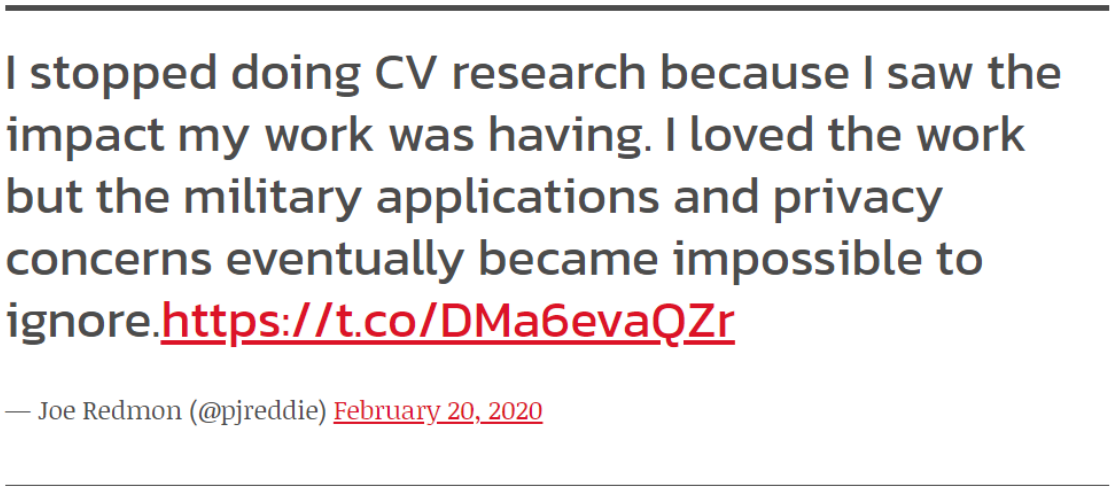


*Ilustración 58. Twitter de Joseph Redmon en el que comunica su abandono del proyecto **YOLO** - **Darknet***

Y es que los trabajos en **Inteligencia Artificial** no están exentos de polémica. Elon Musk ha llegado a calificar estas tecnologías como el mayor riesgo que puede afrontar la humanidad a futuro (The European Business Review, 2021). O incluso, recientemente, científicos chinos han revelado un manifiesto en relación al uso ético de la **IA** (Xinmei, 2021).

El manifiesto chino destaca la necesidad de que la **IA**, sea controlable, confiable, orientada al bienestar humano y que promueva la equidad y justicia, protegiendo la privacidad y seguridad, y elevando la cultura ética.

Sin duda, los aspectos éticos son muy relevantes en el empleo de esta tecnología y serán objeto de debate y estudio en los próximos años.

5.2 La última vuelta de tuerca

Hemos visto que, en el ejemplo del trabajo, un humano es necesario para entrenar a la **IA**, pero ¿podría ocurrir el caso inverso? ¿Podrían una IA apoyar el aprendizaje de los humanos? La respuesta, aparentemente, es afirmativa. Al igual que la **IA** tiene aplicación para el guiado de automóviles como, por ejemplo, en el caso de Tesla, una nueva **IA** es capaz de facilitar el aprendizaje de los humanos en la conducción.

Ese es el caso del proyecto en Shanghái, donde se está probando este concepto (South China Morning Post, 2021)

*Ilustración 59. **IA** capaz de entrenar a humanos*

CAPÍTULO 6. FUTURAS LÍNEAS DE TRABAJO

El trabajo ha demostrado la viabilidad técnica del caso de uso, pero a futuro serían necesarios trabajos adicionales si se desease usar esta plataforma o una similar en un entorno real en producción.

6.1 Evolución técnica

Los aspectos principales a desarrollar serían:

- Hacer reconocimiento de los caracteres de placa de matrícula y extraerlos
- Establecer un sistema de base de datos para el almacenamiento y consulta de imágenes y, eventualmente el almacenamiento de los dígitos extraidos
- Crear una **API** de tipo **REST** (Application Programming Interface) que permita la explotación de la tecnología desde la nube

6.2 Mejora de entrenamiento

Las mejoras de entrenamiento vendrían dadas por la capacidad de identificar placas de matrícula en formatos antiguos o históricos o, incluso placas de matrícula de otros países.

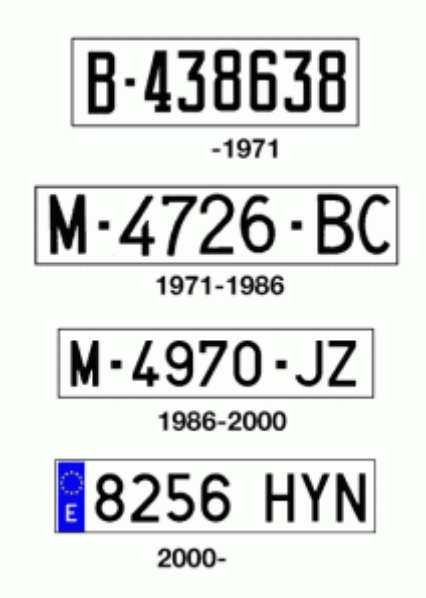


Ilustración 60. Evolución de matrículas en España (ITV, 2021)

Adicionalmente, también se debería hacer un entrenamiento específico del sistema que garantizase el reconocimiento de otras tipologías de matrícula, como la de los ciclomotores, cuerpo diplomático, temporales, etc.

Ilustración 61. Tipos de matrículas adicionales (ITV, 2021)

6.3 Mejora de rendimiento

Vendría dada por el testeo de una nueva plataforma denominada **YOLOR**, o ***Scaled* YOLO**. (Bochkovskiy A. , 2020). Las pruebas son prometedoras de momento, aunque la plataforma está menos extendida y, seguramente requiera mucho trabajo adicional en estabilización de las compilaciones.

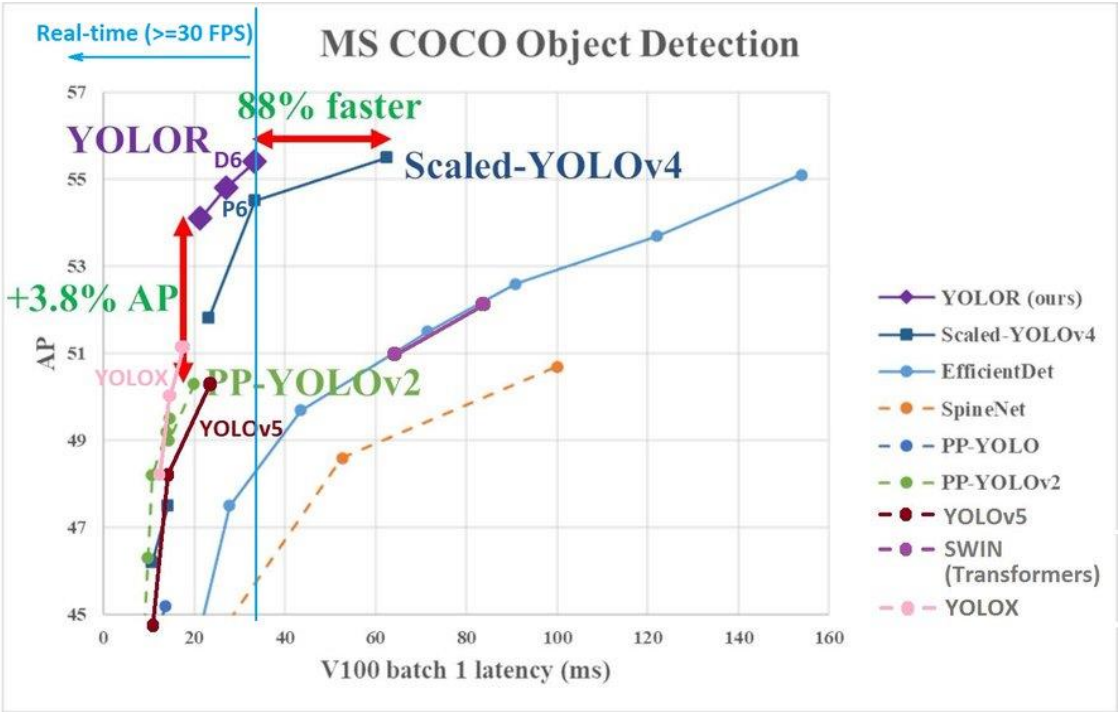


*Ilustración 62. Rendimiento **Scaled YOLO***

6.4 Tesseract OCR

La vía más aconsejable a futuro, para la implementación de un sistema óptico de reconocimiento de caracteres (OCR), pasa por la implementación de la herramienta Open Source, **Tesseract** OCR.

Se introducen a continuación, los pasos que habría que seguir para el despliegue de la misma en la máquina de prueba en la que se incluye **Darknet**. Sobre la herramienta **Tesseract**, sería necesario realizar un desarrollo software que utilizase el cajeado obtenido con **Darknet** y que alimentase el sistema de reconocimiento óptico de caracteres.

- Descargar el software Tesseract desde https://github.com/UB-Mannheim/tesseract/wiki y procecer a su instalación mediante el ejecutable

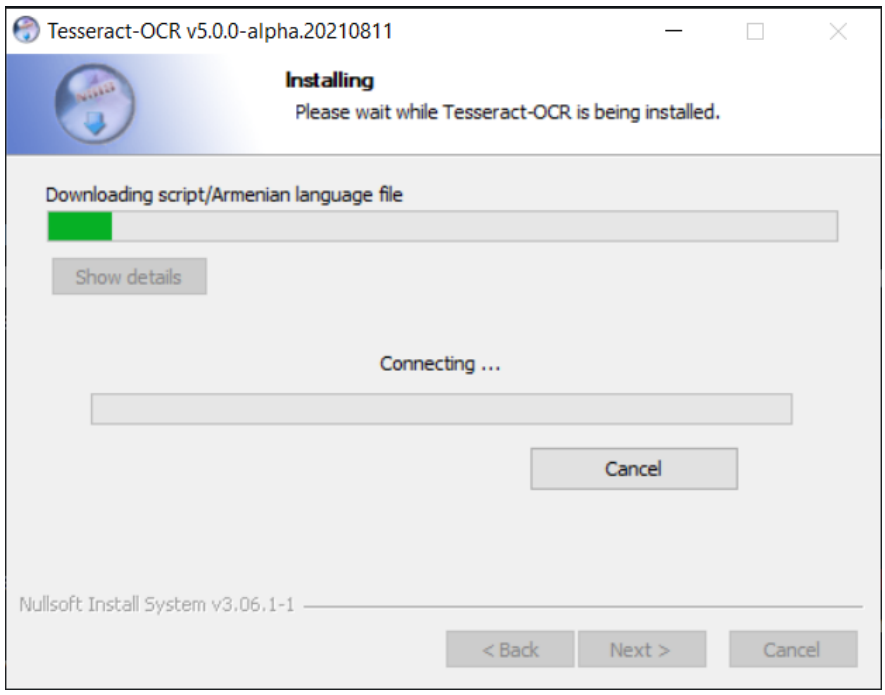


Ilustración 63. Instalación Tesseract

- Instalación del paquete Tesseract para Python mediante consola

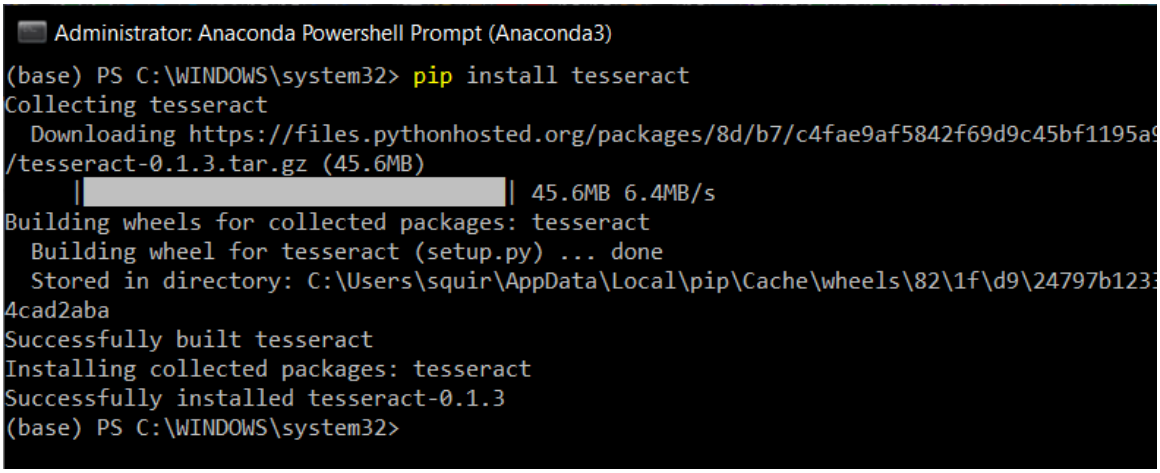


Ilustración 64. Instalación Tesseract

- Añadir variable de entorno al PATH de Windows

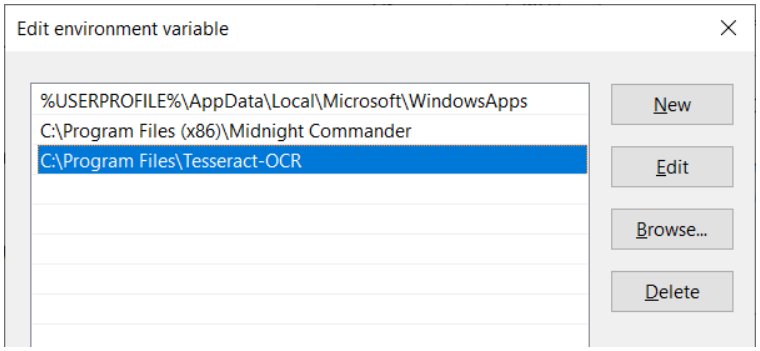


Ilustración 65. Instalación Tesseract

CAPÍTULO 7. ESTUDIO COSTES DESPLIEGUE

En este punto, no se realiza un estudio de costes detallado, pero sí se hace una aproximación al coste de mantenimiento de un equipo capaz de ejecutar la carga de trabajo que supondría utilizar un sistema ya entrenado en la nube.

Se ha escogido la opción de usar **Google Cloud**, que su modelo denominado *Always Free*, es muy interesante ya que, se aplica sobre el nivel inicial de las tarifas estándar. Esto significa que, si estamos iniciando un negocio, podemos comenzar con infraestructura real para ofrecer el servicio y empezar a pagar cuando se sobrepase el límite de gratuidad.

Para este ejemplo de cálculo se ha escogido una máquina tipo, dimensionada para procesamiento bajo demanda sin necesidad de respuesta en tiempo real. Este sistema sería válido para responder a peticiones de detección de placas de matrícula sobre imágenes estáticas.

Consideraré, además que el aplicativo funcionaría en modo *stateless*. Teniendo en cuenta, lo anterior, la máquina presupuestada sería un Intel Xeon E7, 14 *cores* a 2 GHz, 28 GB RAM, 2 TB de disco, 500 MB/s entrada salida y 1 Gbps de velocidad de red.

El número de *cores* se plantea con el objetivo de poder dar servicio al menos, a 10 usuarios concurrentes en un modelo sin procesamiento en tiempo real, ya que no se incluye tarjeta gráfica para apoyo en el procesamiento neuronal.

El almacenamiento persistente es necesario para garantizar la "vida" de los datos entre reinicios de las máquinas virtuales. Los servidores virtuales únicamente llevan un pequeño disco para almacenar el sistema operativo de forma persistente. Màs información acerca de los Zonal Persistent Disks en el apartado de Google al efecto en su web: https://cloud.google.com/compute/docs/disks/#ssdperformance.

Con estas condiciones, el coste de una máquina en **Google Cloud**, sería de unos 850 Euros al mes, que podrían bajar al entorno de 600 euros mensuales con compromiso de 3 años de contrato. Google nos garantiza un SLA del 99,99% de *uptime* (https://cloud.google.com/compute/sla).

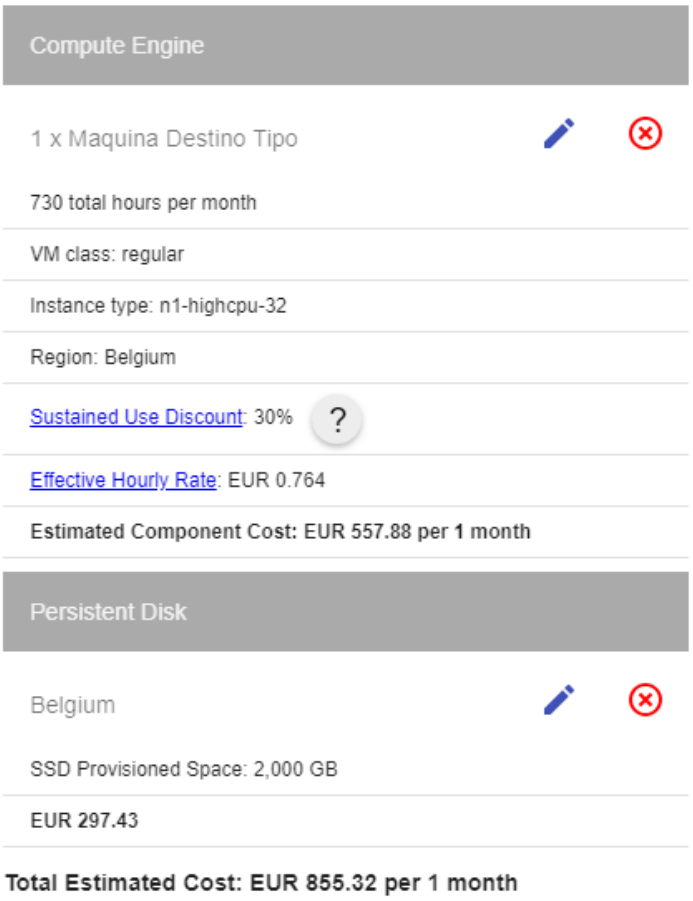


Ilustración 66. Cálculo de costes en Google Cloud para máquina servidor

Si quisiéramos tener el mayor grado de seguridad en la nube, tendríamos que hacer balanceo de cargas entre zonas regionales lo que tiene costes extra tanto a nivel del balanceo como de consumo de red. Y habría que tener en cuenta aspectos de arquitectura, pues no es lo mismo hacer el balanceo de carga para los servidores web, por ejemplo, que en la base de datos, que probablemente requiera un montaje *Master-Slave*.

Un ejemplo posible de arquitectura de máquinas para el proyecto a futuro, podría ser el que se muestra en el diagrama a continuación: con un servidor duplicado de cabecera con balanceo de carga mediante routing y un conjunto de máquinas para procesamiento adicional, bases de datos, y almacenamiento general.

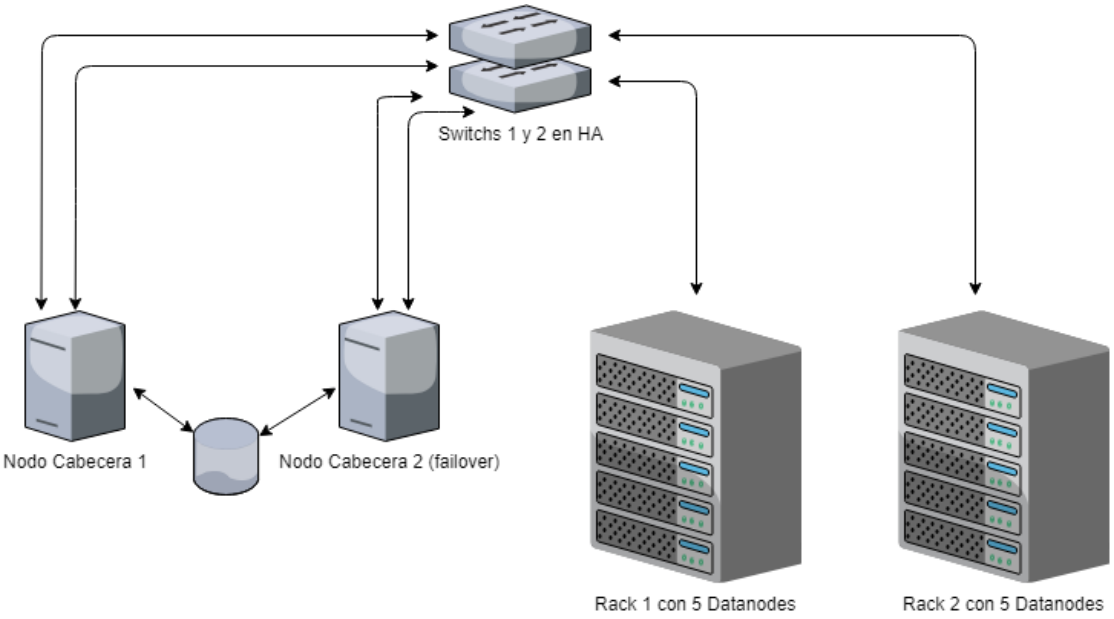
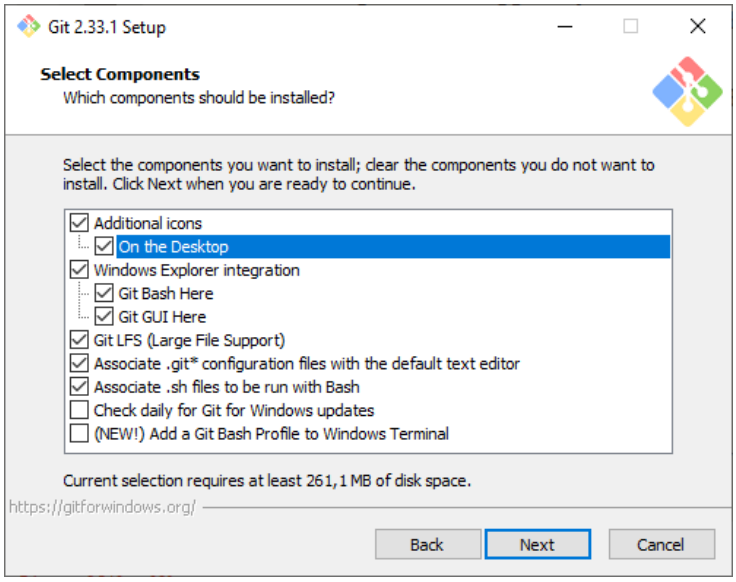
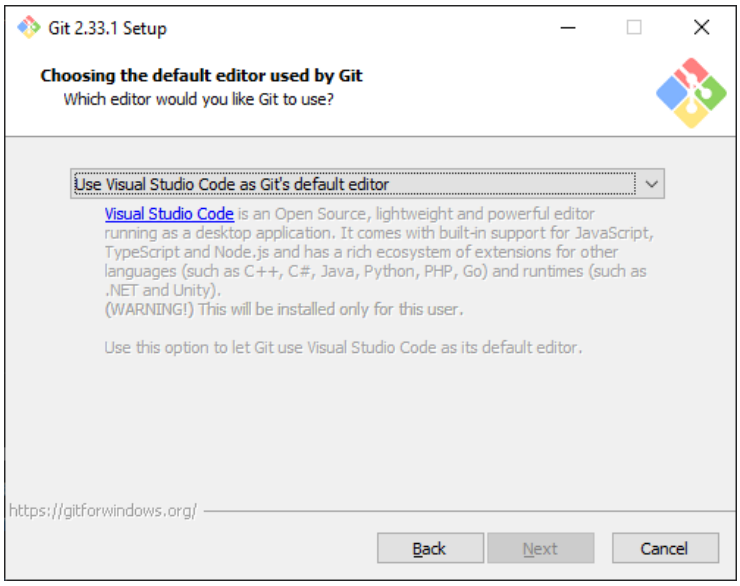


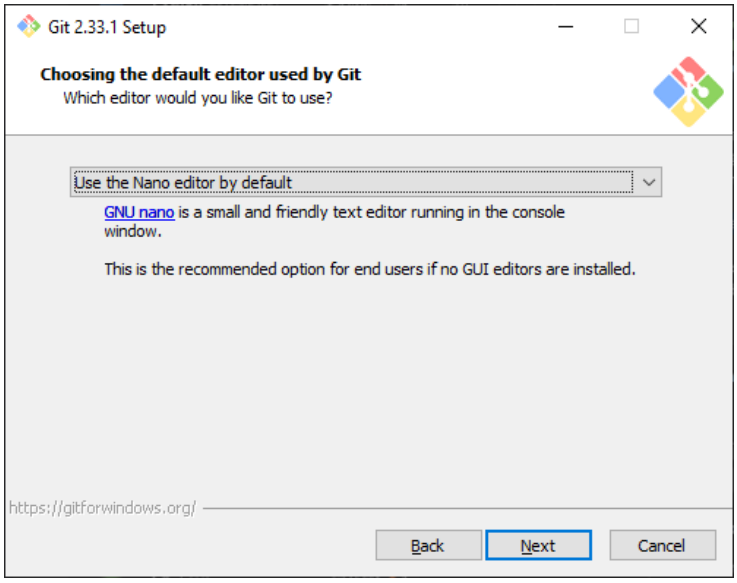
Ilustración 67. Ejemplo de posible arquitectura evolucionada

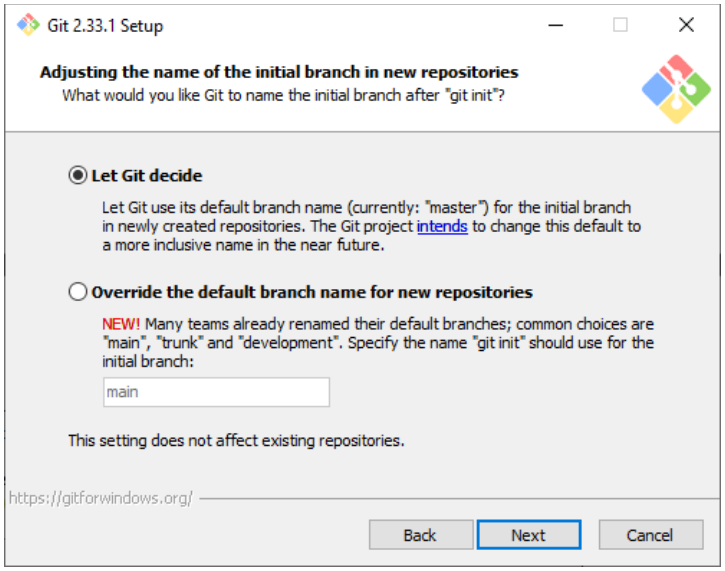
ANEXOS

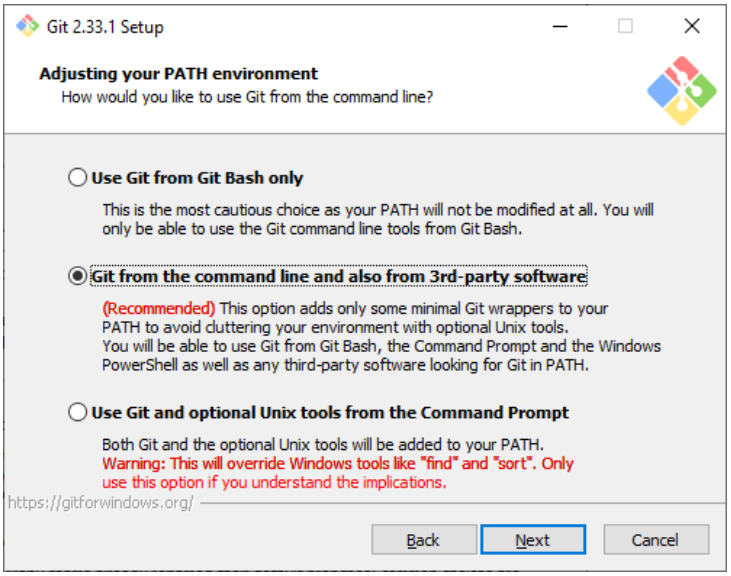
7.1 Opciones de configuración completas para Git

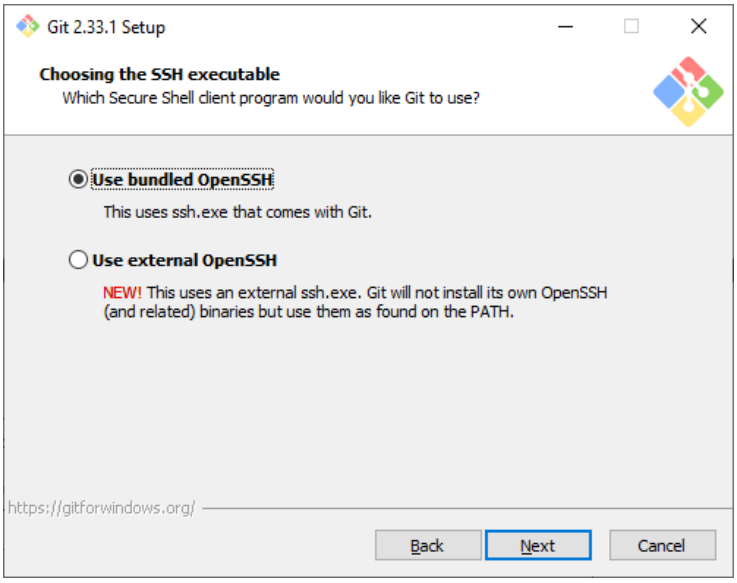


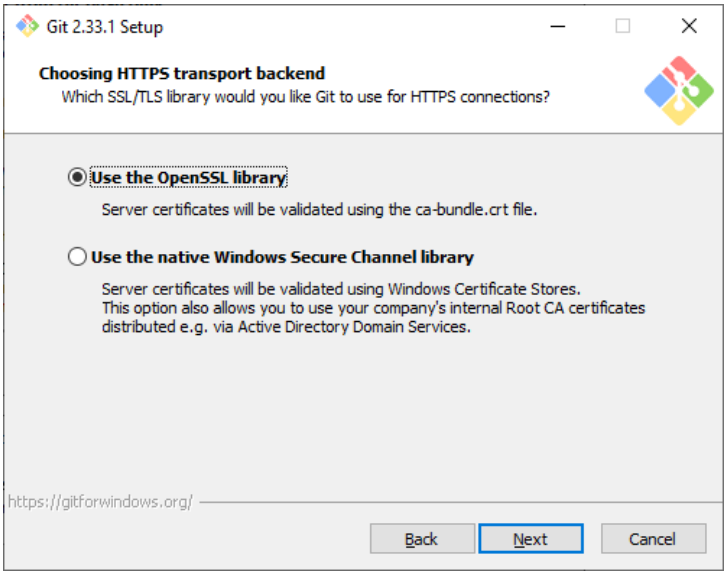


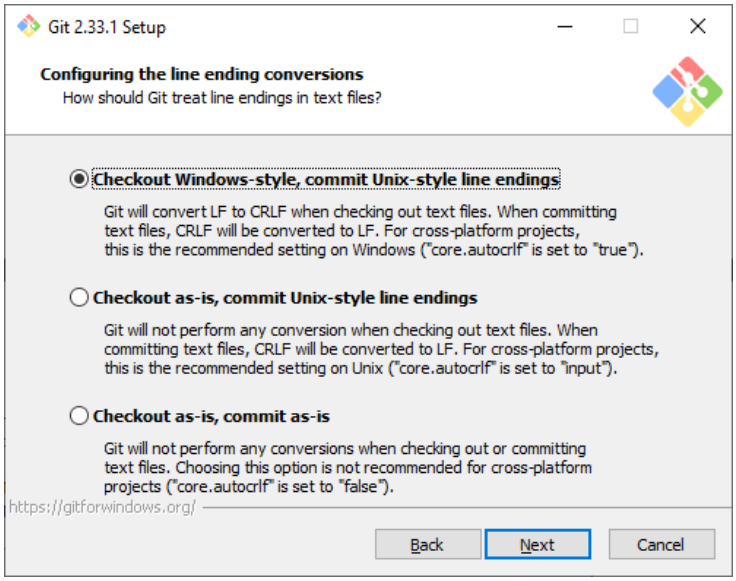

Git 2.33.1 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Use the Nano editor by default
GNU nano is a small and friendly text editor running in the console window.
This is the recommended option for end users if no GUI editors are installed.
https://gitforwindows.org/
Back
Next
Cancel

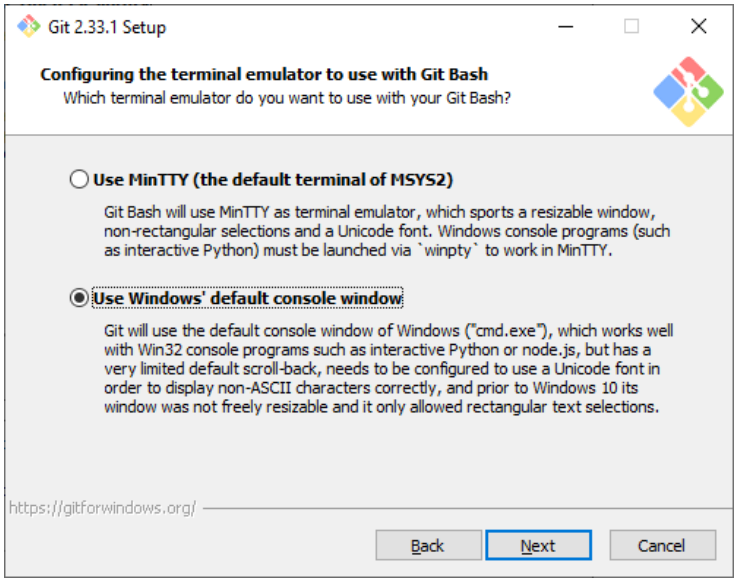

Git 2.33.1 Setup
Adjusting the name of the initial branch in new repositories
What would you like Git to name the initial branch after "git init"?
Let Git decide
Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project intends to change this default to a more inclusive name in the near future.
Override the default branch name for new repositories
NEW! Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:
main
This setting does not affect existing repositories.
https://gitforwindows.org/
Back
Next
Cancel

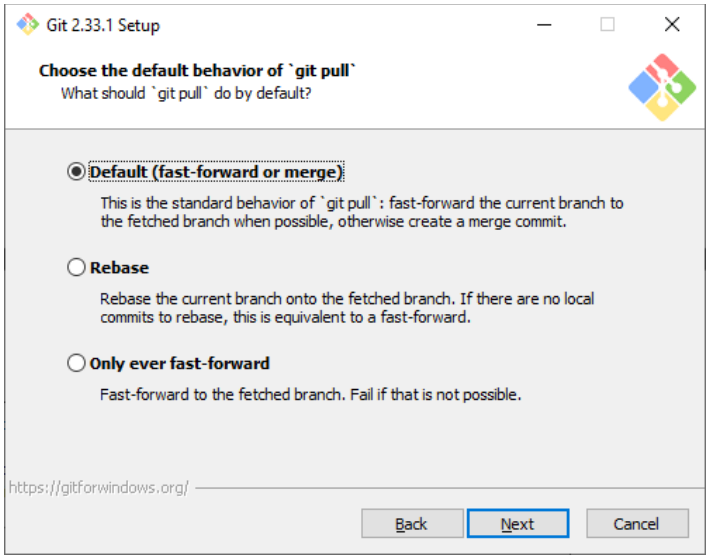

Git 2.33.1 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://gitforwindows.org/
Back
Next
Cancel

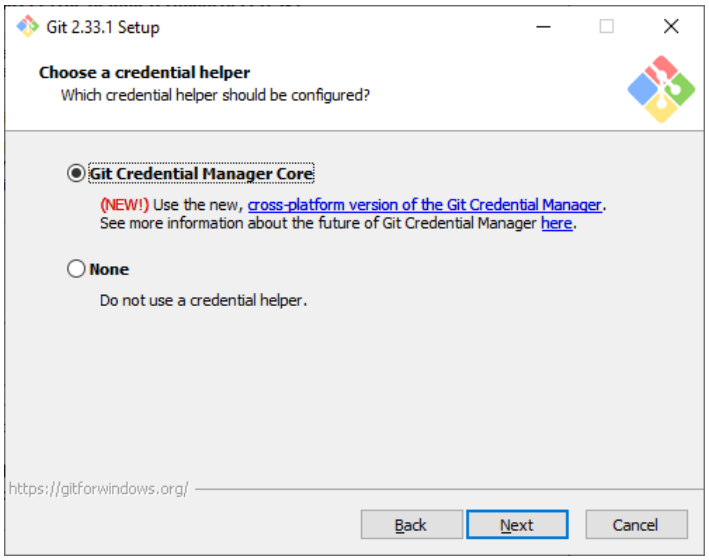

Git 2.33.1 Setup
Choosing the SSH executable
Which Secure Shell client program would you like Git to use?
Use bundled OpenSSH
This uses ssh.exe that comes with Git.
Use external OpenSSH
NEW! This uses an external ssh.exe. Git will not install its own OpenSSH (and related) binaries but use them as found on the PATH.
https://gitforwindows.org/
Back
Next
Cancel

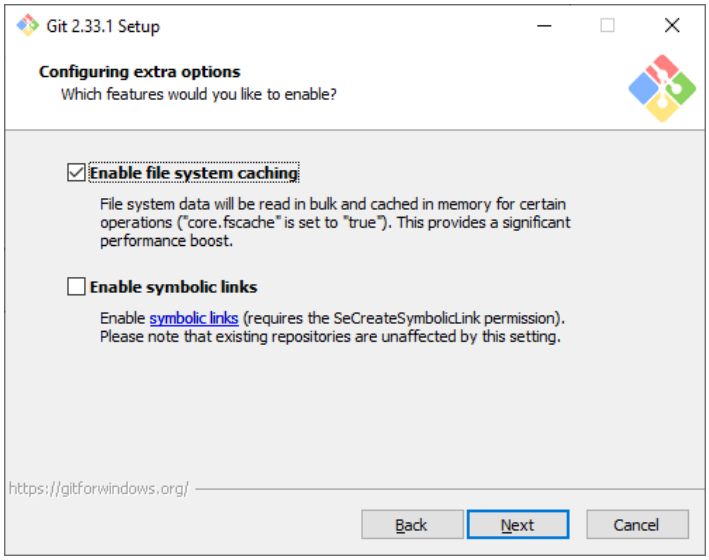

Git 2.33.1 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.33.1 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
Back
Next
Cancel


Git 2.33.1 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.
Use Windows' default console window
Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.33.1 Setup
Choose the default behavior of `git pull`
What should `git pull` do by default?
Default (fast-forward or merge)
This is the standard behavior of `git pull`: fast-forward the current branch to the fetched branch when possible, otherwise create a merge commit.
Rebase
Rebase the current branch onto the fetched branch. If there are no local commits to rebase, this is equivalent to a fast-forward.
Only ever fast-forward
Fast-forward to the fetched branch. Fail if that is not possible.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.33.1 Setup
Choose a credential helper
Which credential helper should be configured?
Git Credential Manager Core
(NEW!) Use the new, cross-platform version of the Git Credential Manager.
See more information about the future of Git Credential Manager here.
None
Do not use a credential helper.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.33.1 Setup
Configuring extra options
Which features would you like to enable?
Enable file system caching
File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.
Enable symbolic links
Enable symbolic links (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.
https://gitforwindows.org/
Back
Next
Cancel

7.2 Opciones de configuración OpenCV

```
-- General configuration for OpenCV 4.5.1 =====================================
--   Version control:               4.5.1
--
--   Extra modules:
--     Location (extra):          C:/Users/squir/opencv_contrib/modules
--     Version control (extra):     4.5.1
--
--   Platform:
--     Timestamp:                 2021-10-15T10:39:23Z
--     Host:                      Windows 10.0.19043 AMD64
--     CMake:                     3.21.3
--     CMake generator:           Visual Studio 16 2019
--     CMake build tool:          C:/Program Files (x86)/Microsoft Visual
Studio/2019/Community/MSBuild/Current/Bin/MSBuild.exe
--     MSVC:                      1929
--
--   CPU/HW features:
--     Baseline:                  SSE SSE2 SSE3
--       requested:               SSE3
--     Dispatched code generation:  SSE4_1 SSE4_2 FP16 AVX AVX2 AVX512_SKX
--       requested:               SSE4_1 SSE4_2 AVX FP16 AVX2 AVX512_SKX
--       SSE4_1 (17 files):         + SSSE3 SSE4_1
--       SSE4_2 (2 files):          + SSSE3 SSE4_1 POPCNT SSE4_2
--       FP16 (1 files):            + SSSE3 SSE4_1 POPCNT SSE4_2 FP16 AVX
--       AVX (5 files):             + SSSE3 SSE4_1 POPCNT SSE4_2 AVX
--       AVX2 (31 files):           + SSSE3 SSE4_1 POPCNT SSE4_2 FP16 FMA3 AVX AVX2
--       AVX512_SKX (7 files):      + SSSE3 SSE4_1 POPCNT SSE4_2 FP16 FMA3 AVX AVX2
AVX_512F AVX512_COMMON AVX512_SKX
--
--   C/C++:
--     Built as dynamic libs?:      YES
--     C++ standard:                11
--     C++ Compiler:                C:/Program Files (x86)/Microsoft Visual
Studio/2019/Community/VC/Tools/MSVC/14.29.30133/bin/Hostx64/x64/cl.exe  (ver
19.29.30136.0)
--     C++ flags (Release):         /DWIN32 /D_WINDOWS /W4 /GR  /D
_CRT_SECURE_NO_DEPRECATE /D _CRT_NONSTDC_NO_DEPRECATE /D
_SCL_SECURE_NO_WARNINGS /Gy /bigobj /Oi  /fp:precise     /EHa /wd4127 /wd4251 /wd4324
/wd4275 /wd4512 /wd4589 /MP  /MD /O2 /Ob2 /DNDEBUG
--     C++ flags (Debug):           /DWIN32 /D_WINDOWS /W4 /GR  /D
_CRT_SECURE_NO_DEPRECATE /D _CRT_NONSTDC_NO_DEPRECATE /D
_SCL_SECURE_NO_WARNINGS /Gy /bigobj /Oi  /fp:precise     /EHa /wd4127 /wd4251 /wd4324
/wd4275 /wd4512 /wd4589 /MP  /MDd /Zi /Ob0 /Od /RTC1
--     C Compiler:                  C:/Program Files (x86)/Microsoft Visual
Studio/2019/Community/VC/Tools/MSVC/14.29.30133/bin/Hostx64/x64/cl.exe
--     C flags (Release):           /DWIN32 /D_WINDOWS /W3  /D _CRT_SECURE_NO_DEPRECATE
/D _CRT_NONSTDC_NO_DEPRECATE /D _SCL_SECURE_NO_WARNINGS /Gy /bigobj /Oi
/fp:precise     /MP   /MD /O2 /Ob2 /DNDEBUG
```

```
--    C flags (Debug):           /DWIN32 /D_WINDOWS /W3  /D _CRT_SECURE_NO_DEPRECATE
/D _CRT_NONSTDC_NO_DEPRECATE /D _SCL_SECURE_NO_WARNINGS /Gy /bigobj /Oi
/fp:precise     /MP /MDd /Zi /Ob0 /Od /RTC1
--    Linker flags (Release):     /machine:x64  /INCREMENTAL:NO
--    Linker flags (Debug):       /machine:x64  /debug /INCREMENTAL
--    ccache:                  NO
--    Precompiled headers:        YES
--    Extra dependencies:         cudart_static.lib nppc.lib nppial.lib nppicc.lib nppidei.lib nppif.lib
nppig.lib nppim.lib nppist.lib nppisu.lib nppitc.lib npps.lib cublas.lib cudnn.lib cufft.lib -
LIBPATH:C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.4/lib/x64
--    3rdparty dependencies:
--
--  OpenCV modules:
--    To be built:              aruco bgsegm bioinspired calib3d ccalib core cudaarithm cudabgsegm
cudacodec cudafeatures2d cudafilters cudaimgproc cudalegacy cudaobjdetect cudaoptflow
cudastereo cudawarping cudev datasets dnn dnn_objdetect dnn_superres dpm face
features2d flann fuzzy gapi hdf hfs highgui img_hash imgcodecs imgproc intensity_transform
line_descriptor mcc ml objdetect optflow phase_unwrapping photo plot python3 quality rapid
reg rgbd saliency shape stereo stitching structured_light superres surface_matching text
tracking ts video videoio videostab xfeatures2d ximgproc xobjdetect xphoto
--    Disabled:                 world
--    Disabled by dependency:     -
--    Unavailable:              alphamat cnn_3dobj cvv freetype java julia matlab ovis python2 sfm
viz
--    Applications:             tests perf_tests apps
--    Documentation:              NO
--    Non-free algorithms:        NO
--
--  Windows RT support:           NO
--
--  GUI:
--    Win32 UI:                 YES
--    VTK support:                NO
--
--  Media I/O:
--    ZLib:                   build (ver 1.2.11)
--    JPEG:                    build-libjpeg-turbo (ver 2.0.6-62)
--    WEBP:                     build (ver encoder: 0x020f)
--    PNG:                     build (ver 1.6.37)
--    TIFF:                   build (ver 42 - 4.0.10)
--    JPEG 2000:                build (ver 2.3.1)
--    OpenEXR:                  build (ver 2.3.0)
--    HDR:                    YES
--    SUNRASTER:                YES
--    PXM:                    YES
--    PFM:                    YES
--
--  Video I/O:
--    DC1394:                   NO
--    FFMPEG:                    YES (prebuilt binaries)
--      avcodec:                YES (58.91.100)
```

```
--     avformat:               YES (58.45.100)
--     avutil:                 YES (56.51.100)
--     swscale:                YES (5.7.100)
--     avresample:             YES (4.0.0)
--   GStreamer:                NO
--   DirectShow:               YES
--   Media Foundation:         YES
--     DXVA:                   YES
--
--   Parallel framework:       Concurrency
--
--   Trace:                    YES (with Intel ITT)
--
--   Other third-party libraries:
--     Intel IPP:              2020.0.0 Gold [2020.0.0]
--           at:               C:/Users/squir/opencv/build/3rdparty/ippicv/ippicv_win/icv
--     Intel IPP IW:           sources (2020.0.0)
--           at:               C:/Users/squir/opencv/build/3rdparty/ippicv/ippicv_win/iw
--     Lapack:                 NO
--     Eigen:                  NO
--     Custom HAL:             NO
--     Protobuf:               build (3.5.1)
--
--   NVIDIA CUDA:              YES (ver 11.4, CUFFT CUBLAS)
--     NVIDIA GPU arch:        35 37 50 52 60 61 70 75 80 86
--     NVIDIA PTX archs:
--
--   cuDNN:                    YES (ver 8.2.4)
--
--   OpenCL:                   YES (NVD3D11)
--     Include path:           C:/Users/squir/opencv/3rdparty/include/opencl/1.2
--     Link libraries:         Dynamic load
--
--   Python 3:
--     Interpreter:            C:/Users/squir/anaconda3/envs/env/python.exe (ver 3.8.8)
--     Libraries:              C:/Users/squir/anaconda3/libs/python3 (ver 3.8.8)
--     numpy:                  C:/Users/squir/anaconda3/envs/env/lib/site-
packages/numpy/core/include (ver 1.20.1)
--     install path:           C:/Users/squir/anaconda3/envs/env/Lib/site-packages/cv2/python-
3.8
--
--   Python (for build):       C:/Users/squir/anaconda3/envs/env/python.exe
--
--   Java:
--     ant:                    NO
--     JNI:                    NO
--     Java wrappers:          NO
--     Java tests:             NO
--
--   Install to:               C:/Users/squir/OpenCV-4.5.1
-- -----------------------------------------------------------------
```

```
--
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/squir/opencv/build
```

7.3 Fichero de configuración YOLO yolo-obj.cfg

```
[net]
# Testing
batch=64
subdivisions=32
# 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 = 2000
policy=steps
steps=1600,1800
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

# Downsample

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=128
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=256
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
```

```
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
```

```
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=512
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
```

```
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
```

```
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
# Downsample

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

######################

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
```

```
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=1
```

```
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1


[route]
layers = -4

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 61



[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
```

```
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[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=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1



[route]
layers = -4

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
```

```
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 36



[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=18
size=1
stride=1
pad=1
activation=leaky
```

```
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=18
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

BIBLIOGRAFÍA / REFERENCIAS

Ahirwar, K. (7 de 5 de 2020). *Secret Sauce Behind Deep Learning*. Obtenido de Hackernoon: https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491

Anolytics. (1 de 10 de 2021). Obtenido de Bounding Box Services: https://www.anolytics.ai/bounding-box-annotation-services/

Bochkovskiy, A. (2020). Scaled YOLO v4 is the best neural network for object detection on MS COCO dataset. *Medium*.

Bochkovskiy, A., Wang, C.-Y., & Mark Liao, H.-Y. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection.* arXiv.

COCO. (18 de septiembre de 2021). Obtenido de Common Objects in Context: https://cocodataset.org/#home

Gandhi, R. (9 de julio de 2018). *Towards Data Science*. Obtenido de R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

García, C., Díez-Pastor, J., Rodríguez, J., & Maudes, J. (2008). License Plate Number Recognition. New Heuristics and a Comparitive Study of Classifiers. *Fifth International Conference on Informatics in Control, Automation and Robotics, Robotics and Automation* (págs. 268-273). Funchal, Madeira, Portugal: DBLP.

ITV. (12 de 5 de 2021). *ITV*. Obtenido de Tipos de matrículas de vehículos en España: ¿Cúantas hay y cuáles son?: https://itv.com.es/tipos-de-matriculas-espana

Krizhevsky, A., Sutskever, I., & Hinton, E. (2012). Imagenet Classification with Deep Convolutional Networks. *Advances in Neural Information Processing Systems*, 1097-1105.

Kromann, R. (30 de Agosto de 2018). *Linkedin*. Obtenido de Comparison of YOLO in Darknet versus YOLO in Darkflow: https://www.linkedin.com/pulse/comparison-yolo-darknet-versus-darkflow-rasmus-kromann/

Leonor, D. (2021 de Febrero de 2021). *Future*. Obtenido de Blog de Innovación en Seguros: https://future.inese.es/mapfre-utiliza-la-ia-para-la-identificacion-automatica-de-danos-con-el-movil/

Miró Julià, J. (2010). *Recursos para aprender a escribir.* Recuperado el septiembre de 2012, de http://bioinfo.uib.es/~joemiro/RecEscr/manual.pdf

Nelson, J. (9 de Enero de 2020). *Roboflow*. Obtenido de Training a YOLOv3 Object Detection Model with a Custom Dataset: https://blog.roboflow.com/training-a-yolov3-object-detection-model-with-a-custom-dataset/

Nelson, J. (9 de Enero de 2020). *Roboflow*. Obtenido de Training a YOLOv3 Object Detection Model with a Custom Dataset: https://blog.roboflow.com/training-a-yolov3-object-detection-model-with-a-custom-dataset/

Pai, A. (17 de February de 2020). *Analytics Vidhya*. Obtenido de CNN vs. RNN vs. ANN: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. *arXiv*.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv*.

Redmon, J., Divala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *arXiv*.

South China Morning Post. (14 de Abril de 2021). Obtenido de AI Coaching in the Driver's Seat in China: https://www.scmp.com/video/china/3129550/ai-instructors-teach-student-drivers-shanghai-how-get-behind-wheel

Tesla Web Page. (8 de Agosto de 2021). Obtenido de https://www.tesla.com/AI

The European Business Review. (22 de Abril de 2021). Obtenido de Find What Elon Musk Said About Real World AI: https://www.europeanbusinessreview.com/find-what-elon-musk-said-about-real-world-ai/

Triggs, R. (14 de Enero de 2019). *Android Authority*. Obtenido de Facial recognition technology explained: https://www.androidauthority.com/facial-recognition-technology-explained-800421/

Xinmei, X. (3 de Octubre de 2021). *South China Morning Post*. Obtenido de Chinese AI gets ethical guidelines for the first time, aligning with Beijing's goal of reining in Big Tech: https://www.scmp.com/tech/big-tech/article/3150789/chinese-ai-gets-ethical-guidelines-first-time-aligning-beijings-goal