UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID
Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Departamentos de Sistemas Informáticos y Computación

Universitat Politècnica de València

Sistema de Detección de Objetos para Reconocimiento Gestual mediante Redes Neuronales Convolucionales

Trabajo Fin de Máster

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Kevin García Santos

Tutor: Roberto Paredes Palacios

2018-2019

Resumen

El objetivo de la visión artificial o visión por computador es proporcionar a las máquinas las herramientas necesarias para ser capaces de ver. Actualmente existen diferentes técnicas o algoritmos para reproducir la visión en las máquinas, y una de ellas son las redes neuronales, la teoría inicia en las décadas de los 40-50, pero es en estos tiempos, con el hardware actual, cuando se ha conseguido llevar a cabo, debido a la gran demanda de cómputo que exigían, la cual, no se podía satisfacer. En lo que se refiere al procesamiento de imágenes, se utilizan redes neuronales convolucionales, las cuales, analizan la imagen a través de la aplicación de diferentes filtros mediante una ventana de tamaño fijo que recorre la imagen completa y obtiene características de esta. La detección de objetos utiliza estas redes neuronales convolucionales, para hallar la localización de un objeto y su clase dentro de una imagen. El contenido de esta memoria explica cómo utilizando este sistema de detección de objetos se ha desarrollado un reconocedor de gestos que a través de imágenes de personas realizando gestos con las manos, es capaz de localizar y clasificar dichos gestos.

Palabras clave: Reconocimiento Gestual, Reconocimiento de Formas, Reconocedor de Gestos, Detección de Objetos, Redes Neuronales Convolucionales, Visión Artificial, Visión por Computador, Faster R-CNN, Microsoft CNTK

Abstract

The objective of artificial vision or computer vision is to provide machines with the necessary tools to be able to see. Currently there are different techniques or algorithms to reproduce the vision in the machines, and one of them is neural networks, the theory started in the decades of 40-50, but it is in these times, with the current hardware, when it has been achieved , due to the great demand of computation that they demanded, which, could not be satisfied. Regarding image processing, convolutional neural networks are used, which analyze the image through the application of different filters through a window of fixed size that runs through the complete image and obtains characteristics of it. Object detection uses these convolutional neural networks to find the location of an object and its class within an image. The content of this report explains how, using this object detection system, a gesture recognizer has been developed that, through images of people making gestures with their hands, is able to locate and classify these gestures.

Keywords: Gestural Recognition, Recognition of Forms, Recognizer of Gestures, Object Detection, Convolutional Neural Networks, Artificial Vision, Computer Vision, Faster R-CNN, Microsoft CNTK

Agradecimientos

Me gustaría agradecer en primer lugar a mi familia, que siempre me ha apoyado y ayudado en todo, y de la cual me siento muy afortunado de formar parte. En especial a mi Aita por todo lo que ha hecho por mí y lo que supone en mi vida.

A Esther por haber sido una gran ayuda para mí en todo el trayecto, dándome fuerzas en los peores momentos y facilitándome la vida mientras yo estaba centrado en el trabajo de fin de máster. Sin ella el camino hubiera sido mucho más difícil.

A mis amigos por siempre estar ahí cuando se les necesita y poder contar con ellos en todo momento. También a mis compañeros de trabajo por amenizar las horas que pasamos trabajando juntos.

Y a mi tutor, Roberto Paredes, por supervisar y revisar la memoria de este trabajo de fin de máster.

Sic parvis magna.

La grandeza nace de pequeños comienzos.

Sir Francis Drake

Tabla de contenidos

Índice de Figuras

Índice de Tablas

1. Introducción

Las redes neuronales artificiales actualmente se han convertido en una herramienta muy útil, y necesaria en ciertos contextos, que tiene muchas aplicaciones en el desarrollo de herramientas, sistemas y funcionalidades en todo el ámbito digital. Tanto, en la visión artificial o visión por computador, que es la parte que se tratará en esta memoria, como en otros entornos, como el reconocimiento del habla en sistemas asistentes **[1]**, la conducción autónoma de automóviles, la detección de enfermedades en la medicina, la predicción de series temporales y muchos otros ejemplos.

En este caso se han utilizado estas redes neuronales artificiales, con el objetivo de generar un sistema capaz de reconocer varios gestos realizados con las manos por una persona. Esto proporciona la capacidad de poder ejecutar acciones sobre otro sistema, sin necesidad de periféricos externos o accionamiento manual físico de ningún tipo. Además de la ventaja añadida de poder hacerlo a distancia y no tener que desplazarte para el accionamiento manual, lo cual también proporciona una ganancia en tiempo de proceso y de mayor ergonomía.

Al final de esta memoria se encuentra un glosario de términos que puede ser de utilidad para una mejor compresión de algunos términos que aparecerán en esta memoria.

1.1 Motivación

En cuanto a la motivación, por una parte, surge de las ganas y curiosidad de poner en práctica los conocimientos aprendidos en el máster en la implementación de un sistema completo basado en redes neuronales. Y, por otra parte, del deseo de una empresa cliente de integrar un sistema más ergonómico y eficiente al utilizado actualmente por las personas que trabajan allí. El trabajo es un proyecto realizado en empresa, perteneciente a la empresa Orbita Ingeniería S.L.

El problema es el siguiente, la empresa cliente tiene 3 líneas de producción. A cada línea de producción, llega un carro transportado por un vehículo de guiado automático (AGV, de su nombre en inglés, Automatic Guided Vehicle (**Figura 1**)). Estos carros son donde se coloca la mercancía a transportar a la siguiente estación de trabajo. El proceso de producción se ejecuta de la siguiente forma, se coloca la mercancía en los carros anteriormente mencionados, después, se efectúa una validación de que el producto es el correcto y está en buen estado. Y, por último, se le debe indicar al AGV que prosiga a la siguiente estación, esto se realiza mediante el accionamiento de un interruptor situado en una mesa, o en caso especial un botón situado en el propio AGV. Estos accionamientos contienen ineficiencias, el accionamiento del interruptor de la mesa conlleva un desplazamiento hasta el lugar donde está situado, provocando pérdida de tiempo en el ciclo de producción. Y el accionamiento en el botón propio del AGV se utiliza en caso de fallo en la

comunicación o para volver a arrancar el vehículo, y provoca mala ergonomía para el operario que lo efectúa, debido al movimiento de agache que es necesario realizar para pulsar dicho botón. Por lo que, mediante las cámaras instaladas en cada una de las líneas de producción, se quiso implementar un sistema capaz de reconocer varios gestos del operario. Cuando la secuencia de estos gestos en un orden fijado es finalizada y detectada, se enviaría una señal para comunicar al AGV que continuara la marcha a la siguiente estación. Esta solución consigue un sistema más ergonómico y sin pérdida de tiempo por desplazamientos, además de la ventaja añadida de poder efectuarlo a distancia.

Figura 1 - Kivnon Automatic Guided Vehicle (AGV)

Los métodos tradicionales de visión artificial no eran adecuados, debido a las condiciones en las que había que hacer la detección, la variabilidad de la imagen era muy alta, en cualquiera de sus características, ya sea, en cuanto a contraste, condiciones lumínicas, localización, y orientación del gesto, se puede observar en las imágenes de la **Figura 2**. Por tanto, según las condiciones y especificaciones anteriores, la opción para desarrollar dicho sistema era utilizar un sistema de detección de objetos basado en redes neuronales convolucionales, el cual, se entrará en detalle en el siguiente capítulo de esta memoria.

Figura 2 - Imágenes líneas de producción 1, 2 y 3 por filas

Para realizar este proyecto se optó por utilizar la herramienta de Microsoft de aprendizaje profundo, CNTK (Microsoft Cognitive Toolkit). Por una parte, la empresa en la que se realizó este proyecto había desarrollado un proyecto con esta herramienta en otra área y habían conseguido obtener un sistema valido, y, por otra parte, a nivel de empresa, una herramienta con el nombre de la marca Microsoft generaba confianza dentro de la empresa y en la compañía cliente, por lo que, finalmente, se tuvo que desarrollar el proyecto con esta herramienta.

Había otras alternativas para el desarrollo del proyecto, una de las principales era utilizar el sistema de detección de objetos YOLO (You Only Look Once) con su última versión, la tercera, el cual, proporcionaba una detección mucho más rápida por la propia estructura del sistema y con buenos resultados de rendimiento en detección. Otra de las opciones era el SSD (Single Shot Detector), que también obtenía buenos resultados en cuanto a tiempo y en rendimiento de detección. Pero debido a los motivos comentados anteriormente se decidió utilizar el Microsoft CNTK, el cual, incluye el Faster R-CNN como detector de objetos, que proporcionaba buenos índices de acierto en detección, ya que, es uno de los detectores de objetos con mejores resultados, a pesar de tener una estructura que implica una menor rapidez de procesamiento de imágenes.

1.2 Objetivos

Según las especificaciones marcadas anteriormente, se busca cambiar el método de indicar al AGV la señal de salida para continuar la marcha, para evitar desplazamientos de los trabajadores y dar la posibilidad de accionamiento a distancia, y además reducir el tiempo de la secuencia de trabajo. Para ello, se debe crear un sistema que incluya la detección de gestos de los operarios, por tanto, para realizar dicha tarea, se necesitan cumplir los siguientes objetivos:

- Estudio de la detección de objetos con redes neuronales convolucionales.
- Conocimiento del entorno de trabajo a utilizar en el desarrollo Microsoft CNTK.
- Adquisición de imágenes con los objetos a detectar.
- Etiquetado de imágenes para proporcionar localización y clase.
- Generar código para realizar aumentación de los datos (data augmentation) en detección de objetos, inclusión de la aumentación de bounding boxes.
- Creación de los conjuntos de datos (datasets) para entrenamiento del modelo de detección de objetos.
- Entrenamiento del modelo de redes neuronales convolucionales con el dataset creado.
- Análisis de los resultados del entrenamiento.
- Implementación de la solución final con la integración del modelo entrenado de detección de gestos para procesamiento de las imágenes.
- Validar funcionamiento del sistema en el entorno final.

1.3 Estructura de la memoria

La estructura de esta memoria está organizada de la siguiente manera:

- 1. Introducción: Este capítulo contiene una descripción inicial del contenido de la memoria, la motivación del proyecto, tanto, personal como para la empresa y la estructuración del contenido de la memoria.

- 2. Estado del Arte: En este apartado se hará un recorrido entre los diferentes detectores de objetos basados en redes neuronales convolucionales, desde cual fue el precursor de estos sistemas, hasta posteriores evoluciones y nuevas arquitecturas.

- 3. Microsoft CNTK – Detección de Objetos: Aquí nos introduciremos en el entorno de trabajo utilizado en el proyecto. Explicación del funcionamiento, herramientas, características y entrenamiento, centrándonos en el área de detección de objetos.

- 4. Hardware Empleado: En este capítulo se hará un repaso de las herramientas empleadas para el desarrollo del proyecto. Además, también se describe el proceso de la adquisición de las imágenes.

- 5. Preproceso y Datasets: El proceso de entrenamiento de las redes neuronales convolucionales requiere mucho trabajo previo, como es el etiquetado manual de las fotos en detección de objetos, la aumentación de los datos a partir de las imágenes etiquetadas y la generación de los conjuntos de datos de entrenamiento. Todo esto se verá en este capítulo.

- 6. Resultados: El capítulo de resultados incluye todos los datos de las diferentes pruebas realizadas a lo largo del desarrollo del sistema.

- 7. Aplicación: En esta sección se añaden las características y el desarrollo que incluye la aplicación final implementada.

- 8. Conclusiones: En este capítulo se introducen las conclusiones del proyecto, así como reflexiones y posibles trabajos futuros.

- 9. Bibliografía: Apartado con las referencias utilizadas a lo largo del proyecto y la memoria de este trabajo de fin de máster.

- 10. Anexos: Al final de la memoria este apartado está reservado para mostrar contenido complementario a lo comentado dentro de la memoria. También se añade un glosario de términos para la explicación de ciertos términos que pueden aparecer en la memoria que no son comúnmente conocidos.

2. Estado del Arte – Detección de Objetos con Redes Neuronales Convolucionales

Un paso muy importante en el área de la detección de objetos se dio con la inclusión de las redes neuronales convolucionales. Las redes neuronales convolucionales consiguieron una mejoría notable en el rendimiento obtenido hasta el momento para este problema.

La detección de objetos con redes neuronales convolucionales empezó con los detectores basados en regiones, donde, el Region CNN fue la base, seguido de sus posteriores evoluciones Fast R-CNN y Faster R-CNN. Después, aparecieron los detectores en tiempo real, capaces de predecir los bounding boxes que contienen a los objetos y la clase a la que pertenecían a una velocidad mayor de 30 fps. Más adelante en esta memoria veremos el YOLO y el SSD de este tipo.

2.1 Detectores Basados en Regiones

2.1.1 Region CNN (R-CNN)

Este método es el precursor de todos los detectores de objetos con redes neuronales convolucionales basados en regiones. Antes de este sistema, las redes neuronales convolucionales no se habían tenido en cuenta como herramienta para la resolución del problema de la detección de objetos en imágenes. Las anteriores investigaciones se habían desarrollado en base a sistemas que incluían métodos de SIFT **[2]** y HOG **[3]**. A partir de la inclusión de las redes neuronales convolucionales en el área de la visión artificial, se empezaron a utilizar con este método para la detección de objetos.

El detector de objetos con redes neuronales convolucionales basado en regiones (Region based Convolutional Neural Networks, conocido como Region CNN o su abreviatura R-CNN), funcionaba de la siguiente manera. A partir de una imagen de entrada, propone múltiples regiones dentro de la imagen a través del algoritmo Selective Search **[4]**, el cual, obtiene alrededor de 2000 regiones diferentes en base a características de la imagen, que proporcionan potenciales zonas que pueden contener objetos. Estas regiones serán con las que se alimentará a la red neuronal convolucional para clasificarlas y obtener a la clase que pertenecen. Esto último se consigue a través de máquinas de vectores de soporte (SVM, abreviatura de Support Vector Machine) **[5][6]** lineales que proporcionan el resultado. Es decir, el algoritmo Selective Search propone las regiones, y la red neuronal obtiene estas regiones como

entrada y determina si pertenece a alguna clase según el nivel de confianza (score) obtenido proporcionado por la SVM. **[7]**

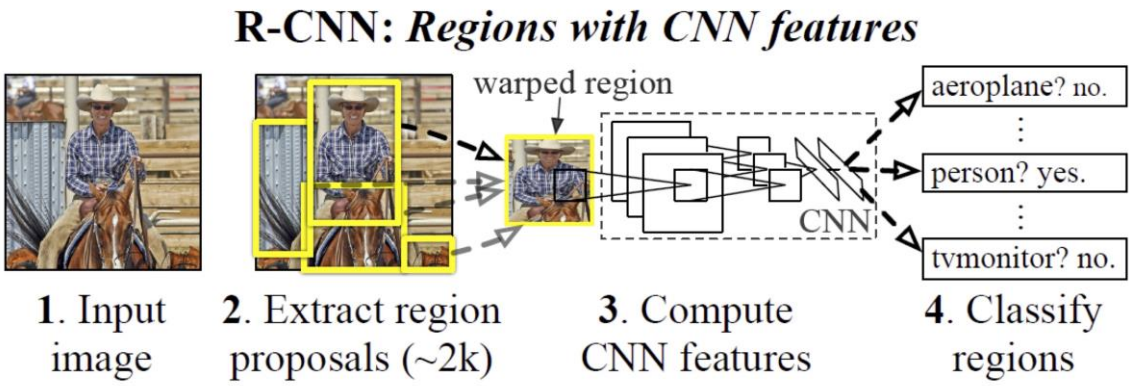

Figura 3 - Esquema Estructura Region CNN

En la **Figura 3** se indica como el sistema primero toma una imagen de entrada, segundo extrae alrededor de 2000 propuestas de regiones, seguido calcula características para cada propuesta utilizando una red neuronal convolucional (CNN, abreviatura de Convolutional Neural Network) y por último, clasifica cada región utilizando SVM lineales específicos de clase. **[7]**

El R-CNN logró una precisión media promedio (Mean Average Precision, abreviado a mAP) **[8]** de **53.7%** en la prueba de detección de objetos PASCAL VOC 2010 **[9]**. En comparación al **35.1%** de mAP obtenido por el Selective Search **[4],** utilizando las mismas propuestas de región.

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	**71.8**	**65.8**	**53.0**	**36.8**	**35.9**	**59.7**	**60.0**	**69.9**	**27.9**	**50.6**	**41.4**	**70.0**	**62.0**	**69.0**	**58.1**	**29.5**	**59.4**	**39.3**	**61.2**	**52.4**	**53.7**

Tabla 1 - Region CNN: Precisión media de detección (%) en prueba VOC 2010

2.1.2 Fast R-CNN

El Fast R-CNN **[10]** es la primera evolución realizada del algoritmo de R-CNN, en cuanto a la estructura es similar, pero con unificación de varias partes y cambios en la mecánica. Este método toma como entrada una imagen completa y un conjunto de propuestas de objetos, los denominados bounding boxes. Primero, la red procesa la imagen completa mediante las capas convolucionales y de Max pooling para producir

un mapa de características. Después, para cada propuesta de objeto, la RoI (Region of Interest) pooling layer extrae un vector de características de longitud fija del mapa de características. Cada vector de características se introduce en la secuencia de capas totalmente conectadas (FC, abreviatura de Fully Connected) que finalmente se ramifican en dos capas de salida: una que indica la probabilidad sobre *K* clases de objetos más una clase de "fondo" general (softmax) y otra capa que genera cuatro valores reales para cada una de las *K* clases de objetos. Cada conjunto de 4 valores codifica posiciones del bounding box refinadas para una de las clases *K*.

Las diferencias que contiene el Fast R-CNN con el Region CNN son, que, en lugar de extraer las características de la imagen a través de la red neuronal convolucional de forma independiente para cada propuesta de región, este modelo obtiene el mapa de características de toda la imagen, y las propuestas de región obtenidas comparten este mapa de características, por lo que, se evita ejecutar múltiples veces la red neuronal, como hacía el Region CNN con las regiones propuestas. Después, el mapa de características se ramifica para ser utilizado en el aprendizaje del clasificador final de los objetos y el regresor de bounding boxes.

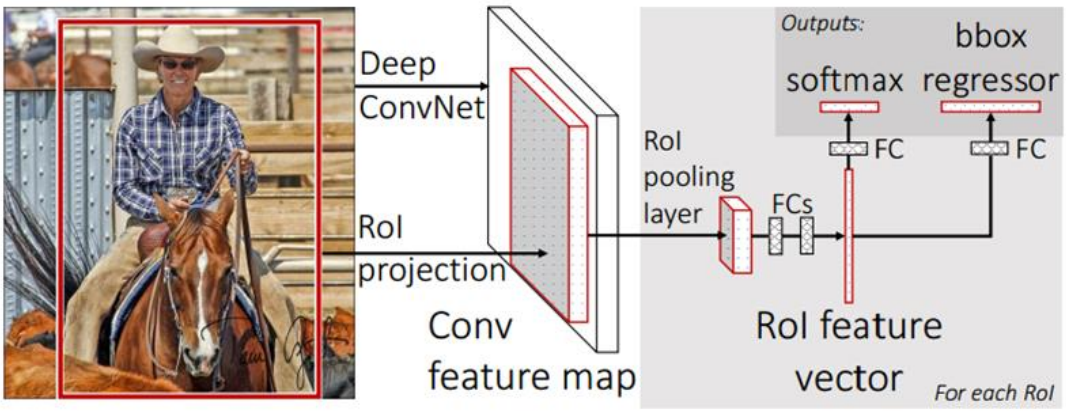

Figura 4 - Esquema Estructura Fast R-CNN

El rendimiento obtenido por este modelo logra ser el mejor hasta el momento. En la prueba de detección de objetos VOC12 **[9]** consiguiendo un mAP de hasta el **68.4%**, en el VOC10 un **68.8%** y en el VOC07 llegando hasta el **70%** de mAP, unos resultados que logran sobrepasar a todos los demás detectores de objetos hasta la fecha.

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Tabla 2 - Fast R-CNN: Precisión media de detección (%) en prueba VOC 2007

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	77.7	73.8	62.3	48.8	45.4	67.3	67.0	80.3	41.3	70.8	49.7	79.5	74.7	78.6	64.5	36.0	69.9	55.7	70.4	61.7	63.8
R-CNN BB [10]	12	79.3	72.4	63.1	44.0	44.4	64.6	66.3	84.9	38.8	67.3	48.4	82.3	75.0	76.7	65.7	35.8	66.2	54.8	69.1	58.8	62.9
SegDeepM	12+seg	**82.3**	75.2	67.1	50.7	**49.8**	71.1	69.6	88.2	42.5	71.2	50.0	85.7	76.6	81.8	69.3	**41.5**	**71.9**	62.2	73.2	**64.6**	67.2
FRCN [ours]	12	80.1	74.4	67.7	49.4	41.4	74.2	68.8	87.8	41.9	70.1	50.2	86.1	77.3	81.1	70.4	33.3	67.0	63.3	77.2	60.0	66.1
FRCN [ours]	07++12	82.0	**77.8**	**71.6**	**55.3**	42.4	**77.3**	**71.7**	**89.3**	**44.5**	**72.1**	**53.7**	**87.7**	**80.0**	**82.5**	**72.7**	36.6	68.7	**65.4**	**81.1**	62.7	**68.8**

Tabla 3 - Fast R-CNN: Precisión media de detección (%) en prueba VOC 2010

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6	63.2
NUS_NIN_c2000	Unk.	80.2	73.8	61.9	43.7	**43.0**	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	**38.6**	**68.3**	58.0	68.7	63.3	63.8
R-CNN BB [10]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	**82.3**	**78.4**	**70.8**	**52.3**	38.7	**77.8**	**71.6**	**89.3**	**44.2**	**73.0**	**55.0**	**87.5**	**80.5**	**80.8**	**72.0**	35.1	**68.3**	**65.7**	**80.4**	**64.2**	**68.4**

Tabla 4 - Fast R-CNN: Precisión media de detección (%) en prueba VOC 2012

El aumento en la velocidad de entrenamiento y ejecución es un factor importante entre un método y el otro, la mejoría en esta faceta es considerable, de las 84 horas de entrenamiento del Region CNN se pasó a entrenar el modelo en 9.5 horas, alrededor de 9 veces más rápido que el tiempo anterior. En fase de prueba el Region CNN empleaba 47 segundos en procesar una imagen, en cambio el Fast R-CNN conseguía procesar la foto en 0.32 segundos, una mejora de 146 veces más que su predecesor.

	Region CNN	Fast R-CNN
train time (h)	84	9.5 (8.8x)
test time (s/im)	47	0.32 (146x)

Tabla 5 - Tabla Velocidad R-CNN vs Fast R-CNN

2.1.3 Faster R-CNN

Después del Fast R-CNN, la siguiente evolución fue el Faster R-CNN **[11]**. Este método consta de dos módulos, el primer módulo es una red totalmente convolucional (FCN, abreviatura de Fully Convolutional Network)**[12]**, denominada red de propuesta de regiones (Region Proposal Network - RPN), y el segundo modulo es la red de detección de objetos Fast R-CNN **[10]** que utiliza las regiones propuestas de la RPN. El módulo RPN le dice al módulo Fast R-CNN donde mirar. Se puede observar la estructura en la **Figura 5**.

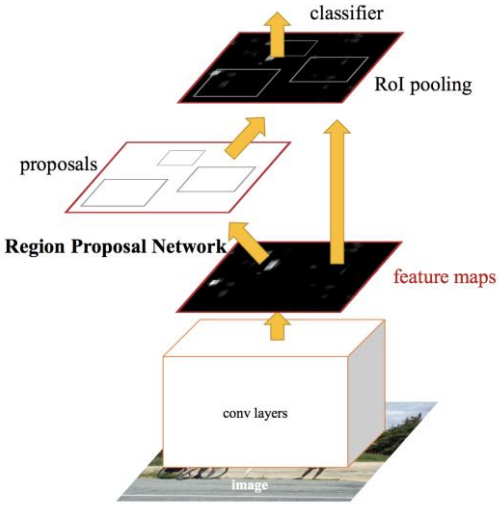

Figura 5 - Estructura Faster R-CNN

La Region Proposal Network coge una imagen como entrada y proporciona como salida un conjunto de propuestas de objeto rectangulares con un nivel de confianza respecto de si la propuesta contiene algún objeto o no. Se utilizó una FCN con el objetivo final de compartir el cálculo con el detector Fast R-CNN, esto permite a las dos redes neuronales utilizar el mismo mapa de características sin necesidad de realizar una segunda pasada de la imagen.

La generación de las propuestas de región se obtiene de la siguiente manera, la salida de la última capa convolucional compartida proporciona el mapa de características convolucional, sobre el cual, se desliza una pequeña red, la anteriormente mencionada Region Proposal Network. Esta red mediante una ventana espacial de tamaño fijo, *n x n*, obtiene las características y a cada ventana deslizante se le mapean las características a una menor dimensión. Después, estas características ya mapeadas se introducen en las dos capas totalmente conectadas hermanas, las cuales, una alimenta la capa de regresión de bounding boxes y la otra la capa de clasificación de si el bounding box contiene un objeto o no. Estas ventanas deslizantes mencionadas, obtienen múltiples propuestas de objetos cada una, con un máximo *k* de propuestas posibles. Estas propuestas son recuadros de referencia que llamamos anclajes (anchors, en inglés), los cuales, tienen asociados una escala y una relación de aspecto cada una. Se utilizan por defecto, 3 escalas y 3 relaciones de aspecto, teniendo así, un total de 9 anclajes, *k=9.* Por lo que, la salida de estas ventanas deslizantes tendrá *4k* salidas correspondientes a las coordenadas de los *k* bounding boxes, en el caso de la capa de regresión de bounding boxes, y *2k* salidas pertenecientes a la probabilidad de objeto o no para cada propuesta, para la capa de clasificación de objeto. Y de esta forma, es como la RPN obtiene las propuestas de región o propuestas de objeto de la imagen.

En la parte izquierda de la **Figura** 6 se puede observar una posición del proceso de la RPN, con los respectivos elementos que la constituyen.

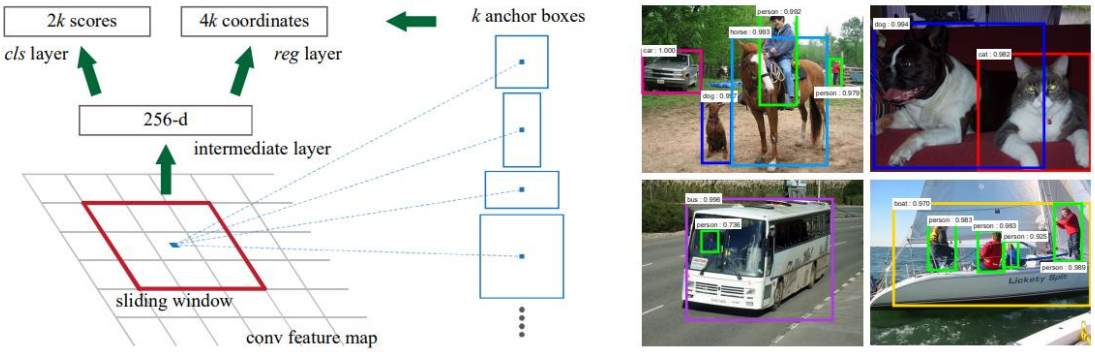

Figura 6 - Izquierda: Region Proposal Network (RPN), Derecha: Ejemplos Deteccion RPN en PASCAL VOC 2007 ***[9]***

El Faster R-CNN **[11]**, por tanto, es una red unificada a partir de dos redes neuronales con un propósito concreto cada una. La RPN se centra en proponer regiones en la imagen con el propósito de localizar los objetos y la segunda red, la Fast R-CNN **[10]**, en base a las características de la imagen, clasifica las regiones proporcionadas para determinar a qué objeto pertenecen, en caso de existir alguno.

Los resultados obtenidos por este método fueron bastante buenos, consiguiendo alzarse como referencia en el ámbito de los detectores de objetos para la fecha. Se hicieron múltiples pruebas en diferentes conjuntos de datos, en este caso, el Faster R-CNN también incluía resultados en el dataset de Microsoft COCO (Common Objects in Context) **0**. La columna con nombre **mAP@0.5** mostrada en la **Tabla 8** del dataset MS COCO, equivale a la métrica utilizada para el reto del PASCAL VOC, si supera un 0.5 score se contabiliza como detección positiva. Mientras que la columna de nombre **mAP@[.5, .95]** constituye la métrica estándar del conjunto de datos MS COCO, la cual, indica la media entre 10 niveles de score diferentes, empieza con 0.5 e incrementa 0.05 cada paso, hasta llegar a 0.95, la media de todos estos resultados es el valor que se indica en la columna.

Este método consigue un mAP de **78.8%** en la prueba de detección de objetos VOC07 **[9]** y un mAP de **75.9%** en el VOC12, un incremento importante respecto de los demás detectores de objetos. En cuanto al dataset MS COCO los resultados no son tan buenos, pero aun así consigue un mAP de **42.7%** en mAP@0.5 y un **21.9%** en mAP@[.5, .95], obteniendo mejores resultados que su predecesor, el Fast R-CNN.

method	# box	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
SS	2000	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
SS	2000	07+12	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
RPN*	300	07	68.5	74.1	77.2	67.7	53.9	51.0	75.1	79.2	78.9	50.7	78.0	61.1	79.1	81.9	72.2	75.9	37.2	71.4	62.5	77.4	66.4
RPN	300	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
RPN	300	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
RPN	300	COCO+07+12	**78.8**	**84.3**	**82.0**	**77.7**	**68.9**	**65.7**	**88.1**	**88.4**	**88.9**	**63.6**	**86.3**	**70.8**	**85.9**	**87.6**	**80.1**	**82.3**	**53.6**	**80.4**	**75.8**	**86.6**	**78.9**

Tabla 6 - Faster R-CNN: Precisión media de detección (%) en prueba VOC 2007

method	# box	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
SS	2000	12	65.7	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7
SS	2000	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	**65.7**	80.4	64.2
RPN	300	12	67.0	82.3	76.4	71.0	48.4	45.2	72.1	72.3	87.3	42.2	73.7	50.0	86.8	78.7	78.4	77.4	34.5	70.1	57.1	77.1	58.9
RPN	300	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
RPN	300	COCO+07++12	**75.9**	**87.4**	**83.6**	**76.8**	**62.9**	**59.6**	**81.9**	**82.0**	**91.3**	**54.9**	**82.6**	**59.0**	**89.0**	**85.5**	**84.7**	**84.1**	**52.2**	**78.9**	65.5	**85.4**	**70.2**

Tabla 7 - Faster R-CNN: Precisión media de detección (%) en prueba VOC 2012

method	proposals	training data	COCO val		COCO test-dev	
			mAP@.5	mAP@[.5, .95]	mAP@.5	mAP@[.5, .95]
Fast R-CNN [2]	SS, 2000	COCO train	-	-	35.9	19.7
Fast R-CNN [impl. in this paper]	SS, 2000	COCO train	38.6	18.9	39.3	19.3
Faster R-CNN	RPN, 300	COCO train	41.5	21.2	42.1	21.5
Faster R-CNN	RPN, 300	COCO trainval	-	-	**42.7**	**21.9**

Tabla 8 - Faster R-CNN: Precisión media de detección (%) en MS COCO

En cuanto a la velocidad del Faster R-CNN **[11]**, también obtiene una mejora en este aspecto. La RPN al compartir las convoluciones el coste de calcular las propuestas es bajo, alrededor de 10ms por imagen, un tiempo despreciable. Esto implica que el proceso completo sea también más rápido, en torno a 320ms que tardaba en procesar una imagen el Fast R-CNN **[10]** junto con el Selective Search **[4]**, al utilizar la RPN junto con el Fast R-CNN se obtiene un tiempo de procesamiento de **198ms**, aproximadamente **5 fps** (frames per second), utilizando como backend el modelo de red VGG16 **[14]**. Empleando el modelo ZF **[15]**, se obtiene una velocidad superior, llegando a los **17 fps,** alrededor de **59ms** por imagen.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

Tabla 9 - Tabla de Velocidad Faster R-CNN

2.2 Detectores en Tiempo Real

2.2.1 YOLO (You Only Look Once)

Un cambio de concepto en los métodos de detección de objetos vino con la aparición de YOLO **[16]**. Solo necesitas mirar una vez a la imagen para predecir que objetos hay en la imagen y donde están. El YOLO unifica las diferentes partes de la detección de objetos en una única red neuronal, que utiliza las características de toda

la imagen para predecir los bounding boxes para todas las clases simultáneamente. Este detector de objetos pertenece al grupo de detectores de objetos en tiempo real, esto se define mediante los fps a los que es capaz de procesar las imágenes. Si el detector de objetos tiene una velocidad de procesamiento de 30 fps o superior, se considera un detector de objetos en tiempo real. Esto implica que es apto para procesar las imágenes según van apareciendo, por ejemplo, en un video.

El sistema divide la imagen en una rejilla de tamaño *S x S*, y cada celda de la rejilla predice *B* bounding boxes. Para esos bounding boxes, también calcula la confianza de que haya un objeto y como de bueno es ese recuadro, esto viene dado por la siguiente formula *Pr(Object)* * IoU_{pred}^{truth}, donde la probabilidad de objeto del bounding box se multiplica a la intersección sobre la unión (IoU, abreviatura de Intersection over Union) entre el ground truth y el recuadro predicho, lo que nos devuelve cuánto solapa la región obtenida. Cada bounding box obtiene 5 componentes, el *(x, y)* para representar el centro, el *(w, h)* para el ancho y el alto relativo a toda la imagen y la confianza mencionada anteriormente. Además, cada celda también predice *C* probabilidades de clase condicionada a la existencia de objeto, *Pr*($Class_i$ |*Object)*. Lo que nos deja la siguiente fórmula para el cálculo de la puntuación de la confianza de los bounding boxes para cada clase, *Pr*($Class_i$ |*Object)* * *Pr(Object)* * IoU_{pred}^{truth} = *Pr*($Class_i$) * IoU_{pred}^{truth}. La **Figura 7** muestra un ejemplo del funcionamiento del sistema.

Finalmente, el tensor obtenido para las predicciones realizadas por este método tiene la siguiente forma, *S* × *S* × *(B* ∗ *5* + *C).*

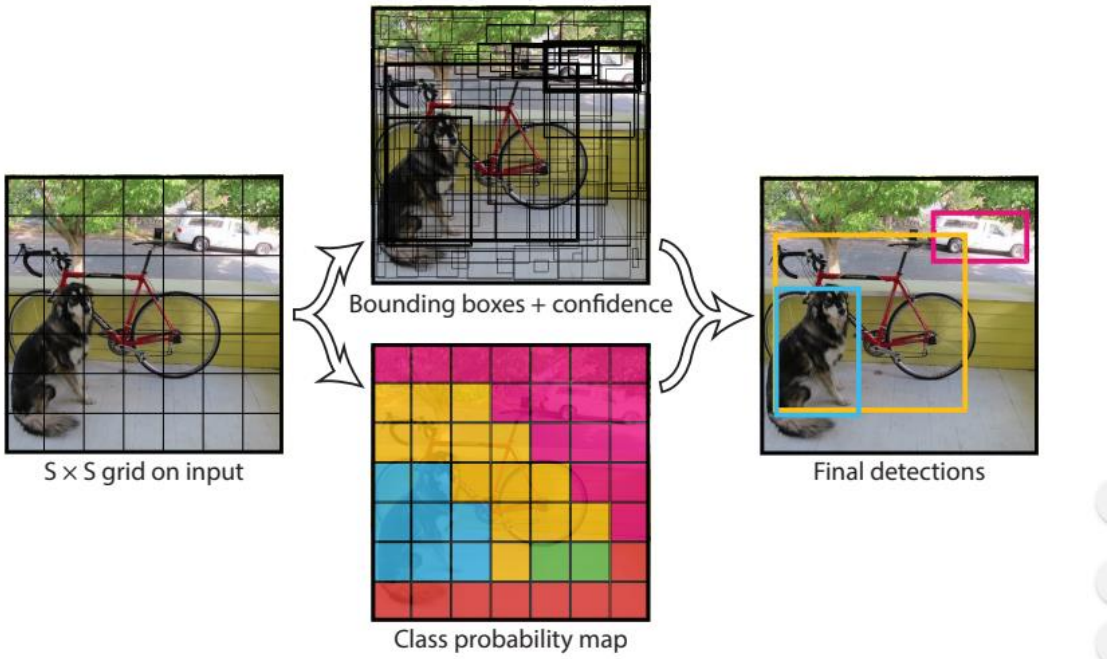

Figura 7 - Ejemplo Funcionamiento YOLO

La red está diseñada como se muestra en la **Figura 8**, está inspirada en la GoogleNet **[17]** sin crear módulos inception, tiene 24 capas convolucionales finalmente seguidas de 2 capas fully connected. También existe una red de menor tamaño, la cual, sustituye las 24 capas convolucionales del YOLO **[16]** estándar por 9 capas, que forman parte del Fast YOLO, diseñado para incrementar los límites de la velocidad en la detección de objetos.

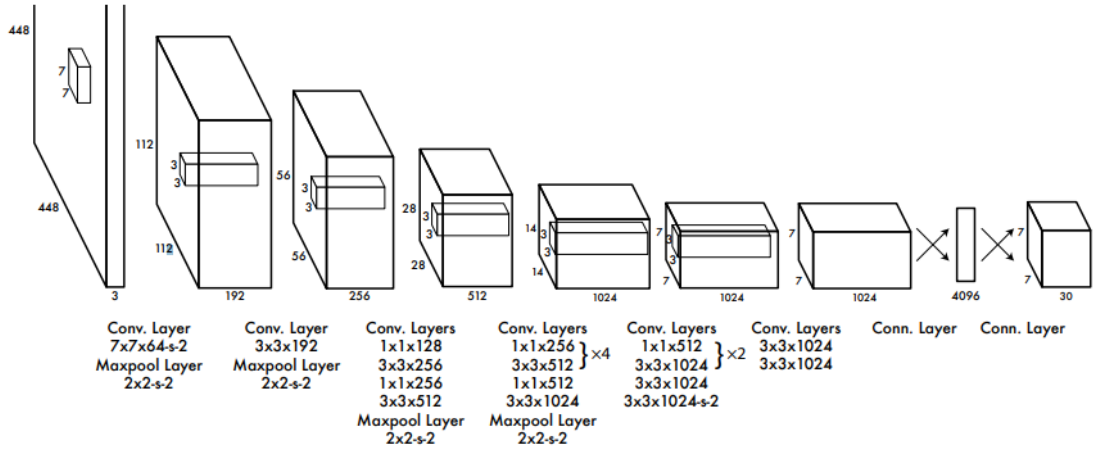

Figura 8 - Estructura Red YOLO

Los resultados de este sistema en cuanto a rendimiento no son una mejora de lo existente en la fecha, pero la velocidad obtenida de procesamiento es lo que le destaca de los demás de detectores de objetos. El YOLO obtiene unos resultados de **63.4%** de mAP en el VOC07 **[9]** con una velocidad de detección de **45 fps**, más rápido que cualquier otro detector hasta el momento, lo que le incluye en el grupo de detectores en tiempo en real. En cuanto a velocidad, la versión de menor tamaño del YOLO, conocida como Fast YOLO, obtiene un resultado de velocidad de **155 fps** sobre el VOC07, lo que proporciona un incremento de velocidad bastante grande, con un rendimiento decente del **52.7%** de mAP. También existe otra variante del YOLO estándar, la cual, utiliza como red neuronal la VGG16 **[14]** que obtiene un 3% más de mAP (**66.4%**), pero con el inconveniente de una bajada de velocidad, la cual, no le permite el procesamiento en tiempo real. Estos datos están visibles en la **Tabla 10** que se encuentra a continuación.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Tabla 10 - Tabla de Rendimiento y Velocidad VOC07 - YOLO y Fast YOLO

La comparativa del YOLO con el Fast R-CNN **[10]** en cuanto a rendimiento no es buena, el Fast R-CNN obtiene un mejor rendimiento. En cambio, si nos centramos en la detección de falsos positivos, este sistema obtiene un mejor resultado a la hora de diferenciar objetos del fondo, tiene un menor error a lo que se refiere a hacer detecciones de objetos que no están en la imagen. Esto se puede observar mediante la gráfica de la **Figura 9**.

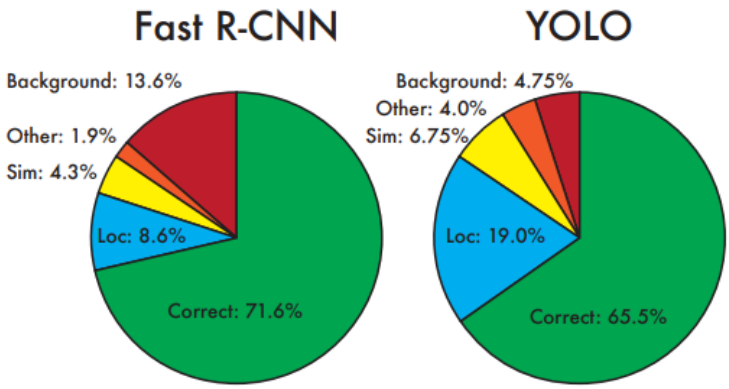

Figura 9 - Grafica Fast R-CNN vs YOLO

Las pruebas realizadas en el PASCAL VOC 2012 (**Tabla 11**), se obtienen a partir de del YOLO estándar y del YOLO como filtrador de falsos positivos de las detecciones del Fast R-CNN, es decir, sobre el resultado del Fast R-CNN se ejecuta posteriormente el YOLO para obtener mejores resultados y evitar las falsas detecciones adquiridas por el Fast R-CNN. El YOLO por si solo obtiene un resultado de **57.9%** de mAP en la prueba, mientras que la combinación del Fast R-CNN con el YOLO alcanza un **70.7%** de mAP.

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

*Tabla 11 - YOLO: Precisión media de detección (%) en prueba VOC 2012 **[9]***

Después de esta primera versión del YOLO, se introdujo una segunda mejora, esta tiene varias modificaciones respecto de la versión anterior, la cual, es denominada YOLO9000 **[18]** o YOLOv2. Previamente a la aparición de la segunda versión del YOLO, surgió el SSD, el cual, introduciremos en el siguiente apartado de esta memoria. Pero la evolución del YOLO no sé mantiene ahí, debido a que el YOLO en su última versión, ha llegado a la tercera evolución, el YOLOv3, que entraremos en detalle más adelante.

2.2.2 SSD (Single Shot Multibox Detector)

El SSD **[19]** es el primer detector de objetos basado en redes neuronales convolucionales que no remuestrea píxeles o características para cada región. El incremento que más repercusión ha tenido en la velocidad ha sido eliminar la parte de propuesta de regiones y el remuestreo de píxeles o características para cada región.

Este método utiliza un pequeño filtro convolucional aplicado a los mapas de características, para predecir las clases de los objetos y los desvíos en la localización de los bounding boxes, a través de un conjunto de bounding boxes por defecto. Para realizar detecciones en diferentes relaciones de aspecto utiliza filtros separados, y aplicando estos filtros a múltiples mapas de características de diferentes escalas de capas posteriores de la red, le proporciona la capacidad de hacer detecciones a múltiples escalas. Esto se puede observar en la **Figura 10.**

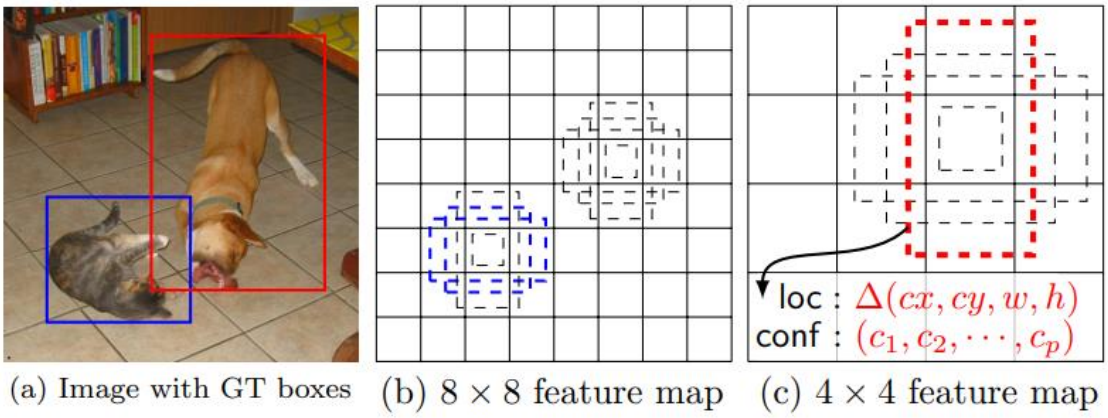

Figura 10 - Ejemplo SSD

El diseño también contiene un entrenamiento simple de extremo a extremo (end-to-end), que incluso para imágenes de baja resolución obtiene buenos resultados de precisión, consiguiendo una mejoría en la velocidad que compensa aún más la precisión obtenida.

El modelo de red creado para el SSD agrega varias capas más al final de la red base. Estas capas predicen las desviaciones de los bounding boxes por defecto a diferentes escalas y las relaciones de aspecto, y sus confianzas asociadas. Se puede ver una comparativa de los modelos de YOLO **[16]** y SSD en la **Figura 11** a continuación.

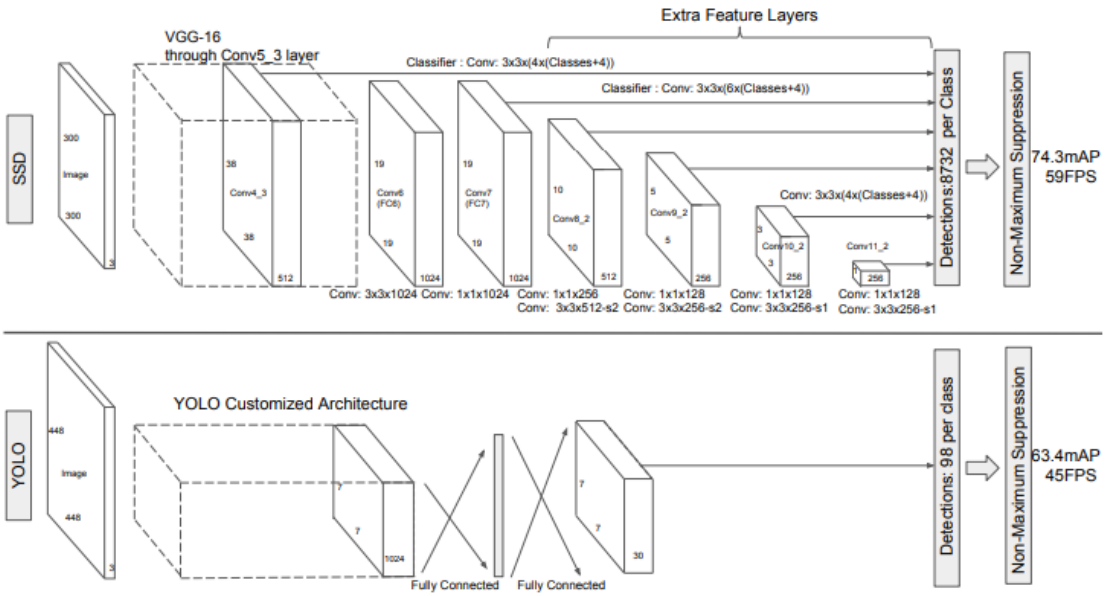

Figura 11 - Modelo SSD vs Modelo YOLO

En fase de entrenamiento, necesita determinar que regiones por defecto concuerdan mejor con el ground truth. Por tanto, se queda con las regiones por defecto que obtienen mejor porcentaje de IoU. Los recuadros por defecto respecto del ground truth si superan cierto umbral (típicamente 0.5) son marcados como positivos, mientras que los demás se marcan como negativos. Esto permite a la red aprender y obtener puntaciones altas en varios de los bounding boxes por defecto superpuestos en vez de solamente en el mejor.

Este sistema implica que hay muchas más muestras negativas que positivas, por lo que se introduce un método para clasificar las regiones por defecto que tienen mayor pérdida de confianza y poder hacer una relación 3:1, como mucho, de negativos y positivos. A este método se le conoce como minería de negativos dura (Hard Negative Mining).

También se introduce un método de data augmentation, el cual, cada imagen de entrenamiento aplica alguna de las opciones disponibles. Estas opciones son utilizar la imagen original completa, crear una muestra que tenga una superposición mínima de IoU con los objetos de 0.1, 0.3, 0.5, 0.7 o 0.9, o crear una muestra aleatoria. El tamaño de las muestras creadas oscila entre 0.1 y 1 del tamaño de la imagen original y la relación de aspecto entre ½ y 2.

La función de perdida para este método se calcula a través de una suma ponderada entre el loss de la localización y el loss de la confianza sobre las clases.

En cuanto a los resultados, el SSD consigue ponerse en lo alto de la tabla entre los detectores de objetos. En la prueba VOC07 **[9]** consigue incrementar el mAP conseguido hasta ahora con un **81.6%** (ver **Tabla 12**). Mientras, en el VOC12 obtiene un **80%** de mAP (ver **Tabla 13**). En el dataset de MS COCO **0** no consigue superar el 50% de mAP en la prueba, pero se coloca el primero con un mAP de **26.8%**, como se observa en la **Tabla 14**.

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
Fast [6]	07+12	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
Faster [2]	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
Faster [2]	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
Faster [2]	07+12+COCO	78.8	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9
SSD300	07	68.0	73.4	77.5	64.1	59.0	38.9	75.2	80.8	78.5	46.0	67.8	69.2	76.6	82.1	77.0	72.5	41.2	64.2	69.1	78.0	68.5
SSD300	07+12	74.3	75.5	80.2	72.3	66.3	47.6	83.0	84.2	86.1	54.7	78.3	73.9	84.5	85.3	82.6	76.2	48.6	73.9	76.0	83.4	74.0
SSD300	07+12+COCO	79.6	80.9	86.3	79.0	**76.2**	57.6	87.3	88.2	88.6	60.5	85.4	**76.7**	**87.5**	**89.2**	84.5	81.4	55.0	81.9	**81.5**	85.9	78.9
SSD512	07	71.6	75.1	81.4	69.8	60.8	46.3	82.6	84.7	84.1	48.5	75.0	67.4	82.3	83.9	79.4	76.6	44.9	69.9	69.1	78.1	71.8
SSD512	07+12	76.8	82.4	84.7	78.4	73.8	53.2	86.2	87.5	86.0	57.8	83.1	70.2	84.9	85.2	83.9	79.7	50.3	77.9	73.9	82.5	75.3
SSD512	07+12+COCO	**81.6**	**86.6**	**88.3**	**82.4**	76.0	**66.3**	**88.6**	**88.9**	**89.1**	**65.1**	**88.4**	73.6	86.5	88.9	**85.3**	**84.6**	**59.1**	**85.0**	80.4	**87.4**	**81.2**

Tabla 12 - SSD: Precisión media de detección (%) en prueba VOC 2007

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast[6]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster[2]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
Faster[2]	07++12+COCO	75.9	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2
YOLO[5]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD300	07++12+COCO	77.5	90.2	83.3	76.3	63.0	53.6	83.8	82.8	92.0	59.7	82.7	63.5	89.3	87.6	85.9	84.3	52.6	82.5	**74.1**	**88.4**	74.2
SSD512	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
SSD512	07++12+COCO	**80.0**	**90.7**	**86.8**	**80.5**	**67.8**	**60.8**	**86.3**	**85.5**	**93.5**	**63.2**	**85.7**	**64.4**	**90.9**	**89.0**	**88.9**	**86.8**	**57.2**	**85.1**	72.8	**88.4**	**75.9**

Tabla 13 - SSD: Precisión media de detección (%) en prueba VOC 2012

Method	data	Avg. Precision, IoU:			Avg. Precision, Area:			Avg. Recall, #Dets:			Avg. Recall, Area:		
		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast [6]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast [24]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster [2]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [24]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster [25]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**

Tabla 14 - SSD: Precisión media de detección (%) en MS COCO

La introducción del data augmentation también tiene su repercusión en los resultados del sistema. En la **Tabla 15** se puede observar que el mAP para los métodos con data augmentation del SSD (métodos marcados con * en la tabla), ha aumentado con respecto al que no utiliza esta herramienta.

Method	VOC2007 `test`		VOC2012 `test`		COCO `test-dev2015` trainval35k		
	07+12	07+12+COCO	07++12	07++12+COCO			
	0.5	0.5	0.5	0.5	0.5:0.95	0.5	0.75
SSD300	74.3	79.6	72.4	77.5	23.2	41.2	23.4
SSD512	76.8	81.6	74.9	80.0	26.8	46.5	27.8
SSD300*	77.2	81.2	75.8	79.3	25.1	43.1	25.8
SSD512*	**79.8**	**83.2**	**78.5**	**82.2**	**28.8**	**48.5**	**30.3**

Tabla 15 – Comparativa resultados SSD con data augmentation

2.2.3 YOLO9000: Better, Faster, Stronger

El YOLOv2 tenía como título: *YOLO9000: Better, Faster, Stronger*, haciendo alusión a las mejoras introducidas en todos los campos del algoritmo.

Primero, para resolver los errores que tenía en la anterior versión del YOLO, se optó por introducir una normalización por lotes (Batch Normalization) de los datos. Al añadir la normalización en todas las capas convolucionales se incrementó el rendimiento proporcionado en un **2%** de mAP, además, esto también normaliza el modelo y evita que se produzca sobreentrenamiento (overfitting).

Otra mejora vino del incremento de la resolución para la clasificación (High Resolution Classifier), la entrada se modificó a una mayor resolución de 448x448, consiguiendo una ganancia de mAP de al menos un **4%**.

A continuación, se sustituyeron los bounding boxes para predecir la localización, por el método de los anchors boxes, ya utilizado en el Faster R-CNN **[11]** y el SSD **[19]**, predice los desvíos (offsets) en vez de las coordenadas completas. Este

elemento no mejora el rendimiento del modelo en sí, pero si hay un incremento en el recall, por lo que proporciona datos que el modelo tiene margen de mejora.

Los anchors boxes anteriormente se introducían manualmente, en este caso, se decidió obtener los anchors automáticamente. Esto se hacía mediante la ejecución de un algoritmo de clustering, el *k-medias*, sobre el conjunto de entrenamiento, para obtener mejores dimensiones previas de los anchors boxes. La distancia estándar del algoritmo no era válida, ya que la distancia Euclídea en los rectángulos grandes producía más error que en los pequeños y se quería conseguir buenas dimensiones previas independientes del tamaño. Por lo que se modifica siguiendo la siguiente formula: *d(box, centroid) = 1 − IOU(box, centroid)*, donde *box* es el recuadro (anchor) y *centroid* el centroide. Un ejemplo se puede observar en la **Figura 12**.

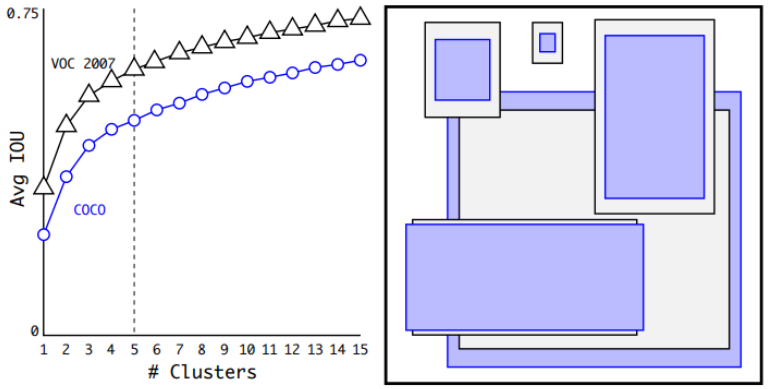

Figura 12 - Clustering de Dimensiones de bounding boxes en VOC y COCO

Posteriormente, se encontró otro problema relacionado con los anchors boxes, inestabilidad en el modelo. El método que se utilizaba para calcular el centro de coordenadas *(x, y)*, provocaba que el recuadro pudiera acabar en cualquier parte de la imagen. Por lo que, se volvió al método utilizado por YOLO **[16]**, para cada celda de la rejilla tenemos 5 bounding boxes con 5 coordenadas para cada bounding box. Esto provocaba que las coordenadas de la localización fueran relativas a la localización de la celda de la rejilla respecto de la esquina superior izquierda, solucionando el problema de los anchors boxes. La red predice 4 coordenadas para cada bounding box, *tx, ty, tw, th*. Si la celda está desplazada de la esquina superior izquierda de la imagen *por (cx, cy)* y la región previa tiene ancho y alto *pw, ph*, las predicciones irán acorde al esquema de la **Figura 13**. Este cambio junto con el clustering para obtener las dimensiones de los bounding boxes, proporciono una mejora del **5%** respecto de la versión con anchors boxes.

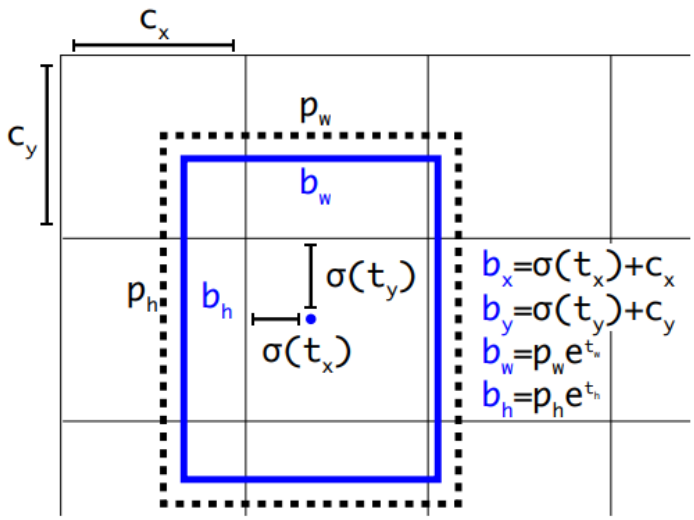

Figura 13 - Representación calculo dimensiones previas de los bounding boxes

El mapa de características de este YOLO era de tamaño 13x13, para detectar objetos grandes era suficiente, pero también podría beneficiar la localización de objetos más pequeños a través de las características más finas. Para ello, de una capa previa se extrae el mapa de características de tamaño 26x26 y apilando ambos mapas, se consigue tener acceso a las características más finas de la imagen, lo que provoca una mejora de **1%** de mAP.

Por último, como nueva mejora en el método, se introdujo el entrenamiento en múltiples escalas. Esto consistía en dotar al sistema de la capacidad de aprender y ejecutar sobre diferentes tamaños de imágenes. Entonces, en fase de entrenamiento, cada 10 batches (lotes) se cambiaba aleatoriamente las dimensiones de la imagen a un múltiplo de 32, siendo 320x320 el mínimo y el máximo 608x608. Esto provocaba que en la ejecución del YOLO9000, si se fijaba el tamaño de la imagen a valores bajos teníamos un menor rendimiento, pero más velocidad en el sistema y si fijábamos las dimensiones de la imagen a valores altos, el rendimiento incrementaba a costa de reducir la velocidad.

En la **Tabla 16** tenemos una muestra de cómo cada mejora introducida sobre la primera versión iba repercutiendo en el rendimiento del sistema.

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Tabla 16 - Tabla Diferencias YOLO y YOLOv2

El incremento en la velocidad viene dado por la nueva estructura de la red neuronal convolucional diseñada para este YOLOv2. El nombre de este nuevo modelo es Darknet19 (**Tabla 17**). Mientras que la VGG16 requería 30.69 billones de operaciones de coma flotante para hacer una única pasada sobre una imagen de 224x224, la primera versión de YOLO solo utilizada 8.52 billones de operaciones. El Darknet19, en cambio, va más allá y solo requiere de 5.58 billones para procesar una imagen, lo que le proporciona su mejora en velocidad.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Tabla 17 - Estructura Darknet19

En la **Figura 14** y la **Tabla 18** se puede observar el rendimiento adquirido con las anteriores mejoras en la prueba del PASCAL VOC 2007. El rendimiento adquirido logra alcanzar el **78.6%** de mAP manteniendo la detección en tiempo real, y la versión más rápida obtiene un **69%** de mAP con una velocidad de **91 fps**. En cuanto a

rendimiento, en esta prueba consigue posicionarse como el mejor detector de objetos hasta el momento.

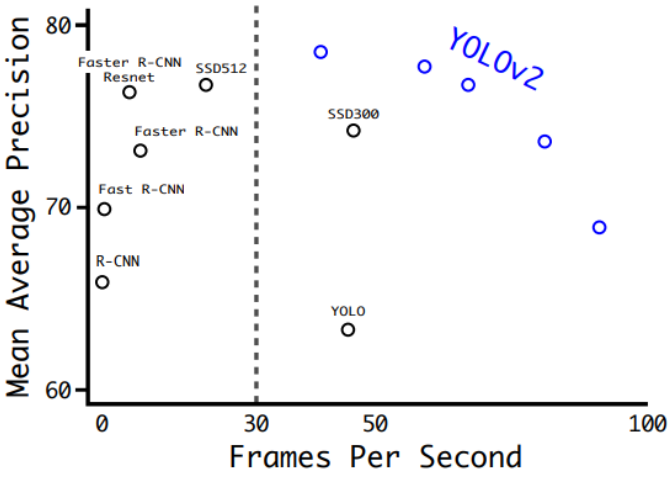

Figura 14 - Grafica Precision Velocidad VOC07

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Tabla 18 - Tabla de Rendimiento y Velocidad VOC07 - YOLOv2

Para el VOC12 (**Tabla 19**), los resultados no son los mejores, pero no están muy por debajo con un **73.4%** de mAP, se mantiene a 1.5% del mejor hasta el momento. La diferencia es que el YOLOv2 es entre 2 y 10 veces más rápido. Comparando el incremento en velocidad con la diferencia en rendimiento, se constata la mejoría en el sistema, además de colocarse entre los mejores hasta la fecha. Los resultados para el dataset COCO **0** tampoco son los más deseables, pero también consigue estar entre los mejores del momento con un **21.6%** de mAP en la prueba (**Tabla 20**).

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Tabla 19 - YOLOv2: Precisión media de detección (%) en prueba VOC 2012 ***[9]***

		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [5]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast R-CNN[1]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster R-CNN[15]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [1]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster R-CNN[10]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300 [11]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [11]	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**
YOLOv2 [11]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4

Tabla 20 - YOLOv2: Precisión media de detección (%) en MS COCO

Finalmente, para hacer más robusto el sistema, se introduce un mecanismo de entrenamiento conjunto de la clasificación y la detección. Las imágenes etiquetadas para detección, las utiliza para aprender sobre la información específica de detección, como la predicción de las coordenadas de los bounding boxes y si contiene objeto o no. Mientras que las imágenes con solo etiquetas de clase las utiliza para ampliar las categorías a detectar. En el proceso de entrenamiento se mezclan los datasets de detección y clasificación. Por lo tanto, las imágenes etiquetadas se propagan hacia atrás sobre la función de perdida (loss) completa del YOLOv2, y la función de perdida de las imágenes de clasificación se propaga hacia atrás solo para las partes específicas de clasificación de la arquitectura.

También genera un modelo jerárquico, para poder predecir clases que pueden pertenecer a un mismo grupo. Es decir, un perro de una raza 'Husky siberiano' podría incluirse en la clase 'perro' además de en la clase 'Husky siberiano', no implica que solo tenga que pertenecer a una sola clase en concreto. Para ello el modelo jerárquico llamado WordTree, cambia el método de utilizar una única función de softmax como distribución de probabilidades, por varias funciones de softmax asociadas a cada grupo (**Figura 15**). Por ejemplo, en el grupo vehículos, incluiría también a todos los tipos de vehículos, ya sean terrestres, acuáticos o aéreos (**Figura 16**).

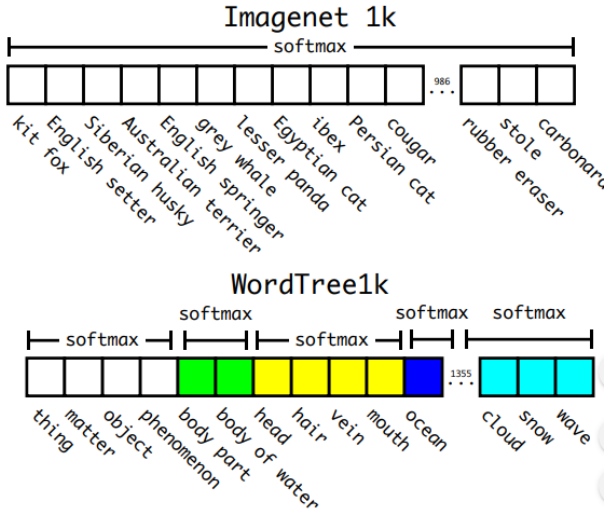

Figura 15 - Predicciones en Imagenet vs WordTree

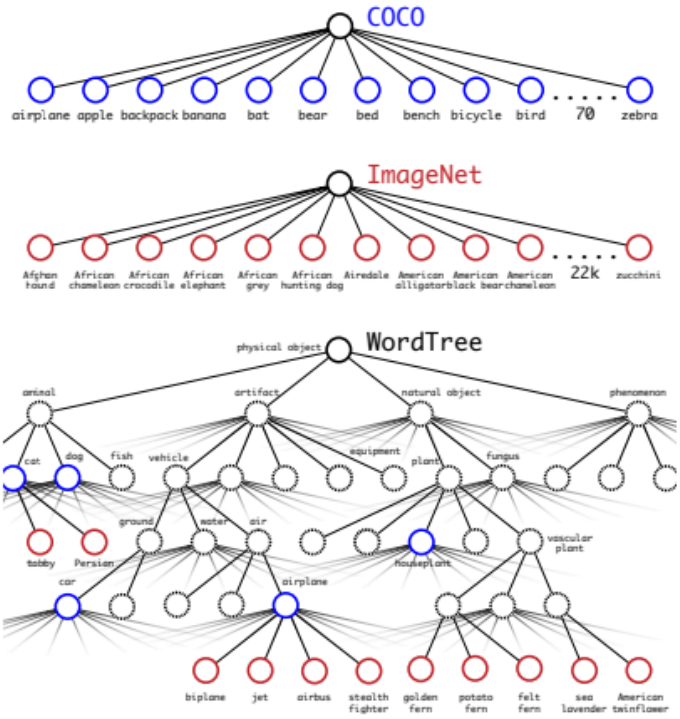

Figura 16 - Combinación de datasets utilizando la jerarquía WordTree

2.2.4 YOLOv3: An Incremental Improvement

En cuanto al sistema YOLO, apareció una siguiente mejora, la tercera versión del sistema, YOLOv3 **[20]**.

Esta versión, en la predicción de bounding boxes, sigue el mismo método comentado anteriormente para el YOLO9000 **[18]** (ver **Figura 17**).

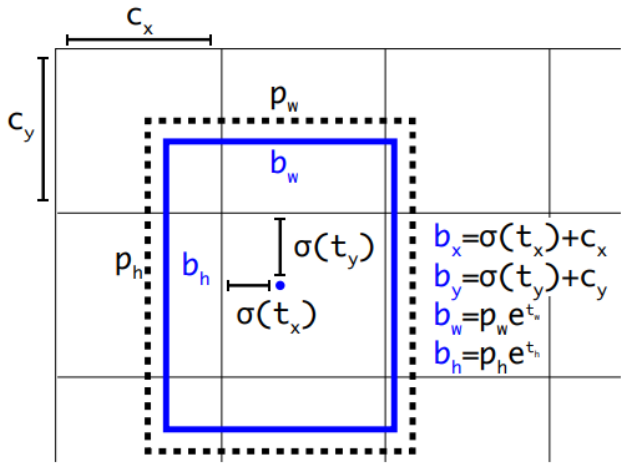

Figura 17 - Esquema dimensiones previas bounding boxes

El YOLOv3 por cada recuadro delimitador predice el score de la existencia de objeto, y si el bounding box definido a priori solapa al ground truth más que cualquier otro, obtiene el máximo como resultado (el máximo es 1). Por lo que el sistema solo asigna un bounding box definido a priori por cada ground truth.

Cada recuadro predice las clases que puede contener el bounding box utilizando clasificación multietiqueta. Por lo que se sustituye la función de softmax habitual, por clasificadores logísticos independientes. La softmax no es necesaria para un buen rendimiento, utilizar esta función implica que un bounding box solo puede contener una clase y esto a menudo no es así.

En esta versión del YOLO, para la predicción obtiene características a partir de 3 escalas diferentes. También se coge el mapa de características de una etapa temprana en la red y se concatena con las características sobremuestreadas posteriores, consiguiendo así información semántica significativa e información detallada más fina (finer-grained information). Además, se sigue utilizando el método de clustering de las *k-medias* para determinar los bounding boxes previos a aplicar con el ground truth.

En el entrenamiento no utilizan hard negative mining, pero si introducen el entrenamiento multiescala, mucho data augmentation y normalización por lotes de los datos (batch normalization).

En cuanto al modelo de extracción de características, surge la nueva versión del Darknet, la denominada Darknet53 (ver **Figura 18**). Es una mezcla entra el modelo Darknet19 del YOLO9000 y las nuevas redes residuales **[21]**. En este caso, la red se amplía a una cantidad de 53 capas convolucionales. A pesar de su longitud, es más eficiente que las ResNet, obteniendo una velocidad mayor con un rendimiento similar (ver **Tabla 21**).

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

Figura 18 - Estructura Darknet53

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Tabla 21 - Comparación de modelos. Acierto, billones de operaciones, billones de operaciones en coma flotante por segundo y FPS

Los resultados de esta nueva versión del sistema no son los que más rendimiento proporcionan, pero en comparación con la velocidad que obtienen, la mejora es considerable, a pesar de un rendimiento algo inferior. En el dataset de Microsoft COCO **0**, para la métrica general del COCO obtiene un mAP de **33%**. Para la métrica de mAP-50 el resultado obtenido es de un **57.9%** de mAP, como se aprecia en la **Tabla 22**. Si nos fijamos en las gráficas de la **Figura 19** y **Figura 20**, podemos observar que a pesar de tener un rendimiento menor que sus adversarios, la velocidad obtenida está bastante por encima, por lo que compensa, ya que, el rendimiento no es el mejor pero se mantiene cerca de los primeros.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Tabla 22 - YOLOv3: Precisión media de detección (%) en MS COCO

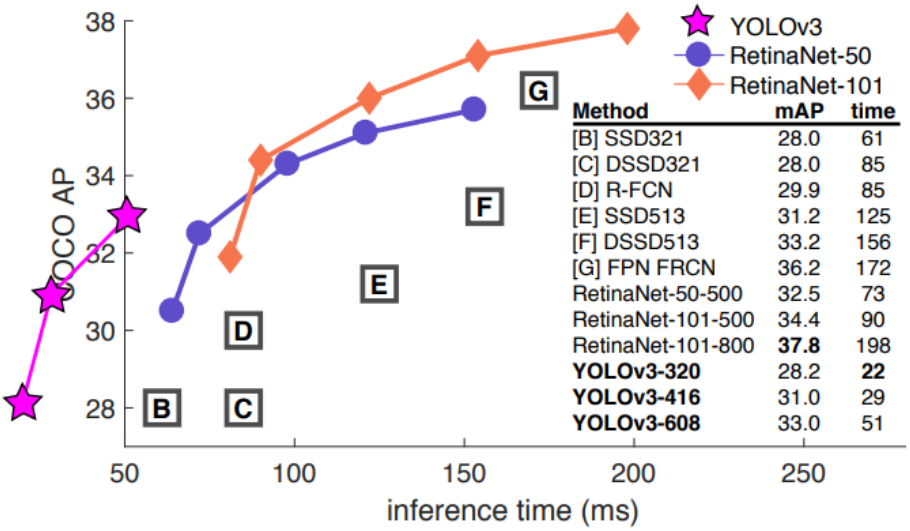

Figura 19 - YOLOv3: Grafica Comparativa Rendimiento y Velocidad COCO

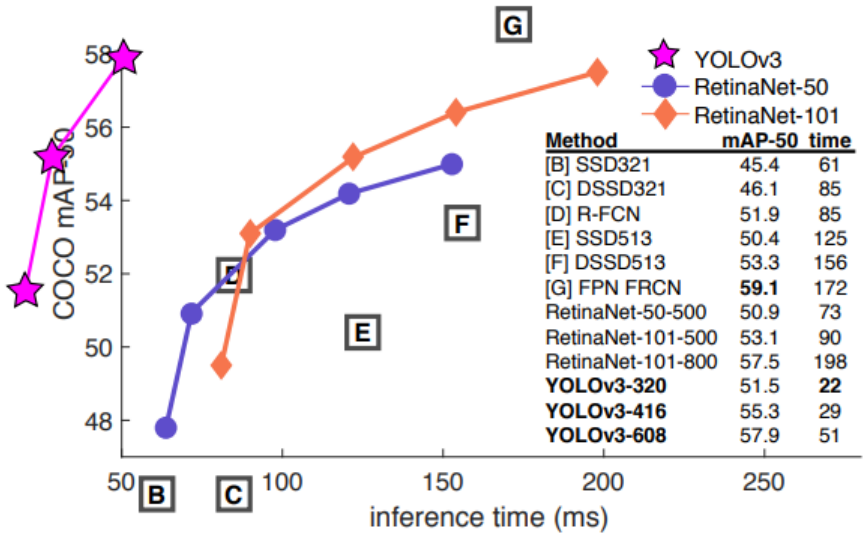

Figura 20 – YOLOv3: Grafica Comparativa Rendimiento y Velocidad COCO mAP-50

3. Microsoft CNTK – Detección de Objetos

El Microsoft Cognitive Toolkit (CNTK) **[22]** es un conjunto de herramientas de código abierto para el uso de aprendizaje profundo, con la opción de utilizarse para ámbito comercial. Como método de aprendizaje utiliza el descenso por gradiente estocástico (SGD, el error se retropropaga hacia atrás (error backpropagation)), con soporte para diferenciación y paralelización automáticas en múltiples GPUs o servidores. El CNTK puede ser incluido como librería en los lenguajes de programación Python, C# o C++.

El propio CNTK tiene implementados muchos tipos de sistemas de aprendizaje profundo. Desde ejemplos simples, como el MNIST, hasta algoritmos de Reinforcement Learning, reconocimiento del habla o reconocimiento de texto entre otros. En el enlace a pie de página se encuentra un video introductorio al Microsoft CNTK[1].

En nuestro caso, nos centraremos en la parte del Microsoft CNTK que tiene que ver con los métodos de visión artificial para la detección de objetos. Este sistema cuenta con dos detectores de objetos completos implementados, el Fast R-CNN **[10]** y el Faster R-CNN **[11]**. El Fast R-CNN tiene incluido como generador de propuestas de regiones el Selective Search **[4]**. Como se comentó previamente en esta memoria, este proyecto se llevó a cabo utilizando el método Faster R-CNN.

3.1 Faster R-CNN con Microsoft CNTK

El detector de objetos Faster R-CNN del Microsoft CNTK, cuenta con una guía de introducción al algoritmo en este entorno **[23]**. En el resumen inicial se muestran dos imágenes con las detecciones del algoritmo. Una imagen muestra las detecciones sobre el dataset Grocery, que contiene alimentos. Y en la otra imagen se muestran las detecciones sobre el dataset del PASCAL VOC **[9]** (ver **Figura 21**).

[1] Video introductorio CNTK: https://www.youtube.com/watch?v=9gDDO5ldT-4&feature=youtu.be

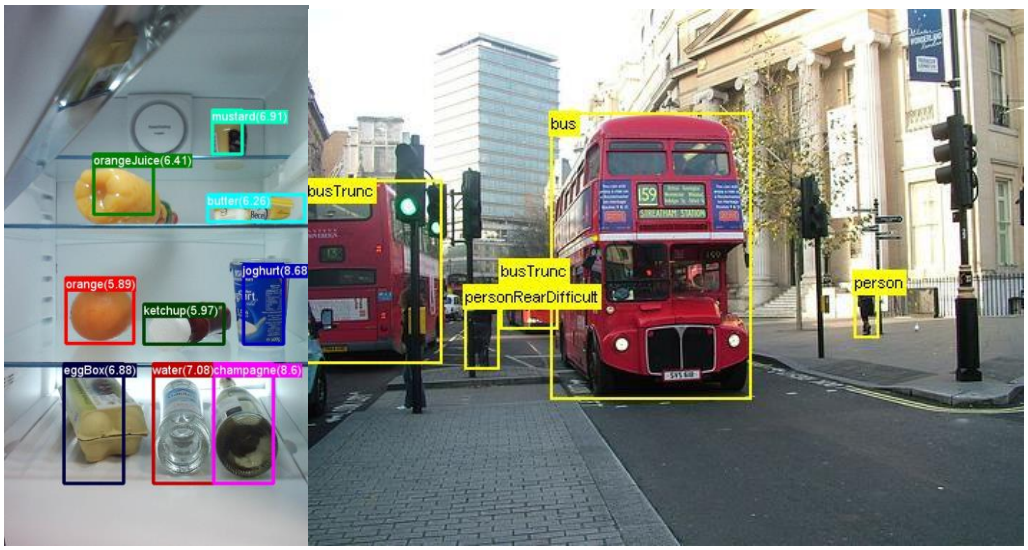

Figura 21 - Detecciones Faster R-CNN CNTK - Izquierda: Grocery dataset, Derecha: Pascal VOC

3.1.1 Configuración de Parámetros

Para comenzar con el proceso de entrenamiento de la red, lo primero que hay que tener en cuenta es la configuración. Los parámetros de configuración están divididos en tres partes:

- Parámetros del detector (Faster R-CNN)
- Parámetros del dataset (Grocery, Pascal VOC, ...)
- Parámetros del modelo base (AlexNet, VGG16, ...)

Cada uno de ellos tiene múltiples parámetros que necesitaremos configurar para ejecutar el sistema, y poder entrenar nuestro modelo, a continuación, veremos los parámetros de cada parte.

3.1.1.1 Parámetros del Detector

Los parámetros del detector son los que se utilizan para determinar la ejecución del Fast R-CNN o el Faster R-CNN. En nuestro caso, la configuración a modificar será la del Faster R-CNN. Los parámetros generales del CNTK son los siguientes:

- **MAKE_MODE**: Nos permite indicar si al entrenar queremos sobrescribir el modelo o saltar el entrenamiento en caso de que ya exista.
- **TRAIN_E2E**: Elección de entrenamiento end-to-end o el entrenamiento en 4 fases (4-stage).
- **FORCE_DETERMINISTIC**: Indica si queremos usar algoritmos deterministas o no.
- **FAST_MODE**: Si está activo ejecuta un solo epoch.
- **DEBUG_OUTPUT**: Genera información extra de depuración en la salida de la consola.
- **GRAPH_TYPE**: El formato del grafo, 'png' o 'pdf'.
- **STORE_EVAL_MODEL_WITH_NATIVE_UDF**: Activar para crear un modelo de evaluación UDF nativo, almacena un segundo modelo de evaluación sin capas Python, solo código nativo. Útil para usarse en C++ o C#.

A continuación, tenemos la configuración de los parámetros de aprendizaje:

- **L2_REG_WEIGHT**: El peso de la regularización L2.
- **MOMENTUM_PER_MB**: Valor del momentum por cada mini-batch.
- **BIAS_LR_MULT**: Multiplicador del learning rate para todos los pesos de los bias de la red.

En cambio, los parámetros de aprendizaje de las opciones de entrenamiento disponibles (end-to-end o 4-stage), van separadas cada una con sus respectivos parámetros. Los parámetros de ambos métodos indican lo mismo pero cada uno tiene su propia estructura. El método de 4-stage divide los parámetros de la RPN y del detector (Fast R-CNN).

Para el entrenamiento de extremo a extremo (end-to-end), tenemos los siguientes parámetros:

- **E2E_MAX_EPOCHS**: Número máximo de epochs de entrenamiento a realizar (método end-to-end).
- **E2E_LR_PER_SAMPLE**: Vector de valores del learning rate aplicados a cada muestra por epoch (método end-to-end).

En el entrenamiento de 4 fases (4-stage), tendríamos los parámetros a continuación:

- **RPN_EPOCHS**: Número máximo de epochs de entrenamiento a realizar en la RPN.
- **RPN_LR_PER_SAMPLE**: Vector de valores del learning rate aplicados a cada muestra por epoch en la RPN.

- **FRCN_EPOCHS**: Número máximo de epochs de entrenamiento a realizar por el Fast R-CNN.
- **FRCN_LR_PER_SAMPLE**: Vector de valores del learning rate aplicados a cada muestra por epoch en el Fast R-CNN.

La configuración de los datos de la entrada de la red viene asociada a los siguientes parámetros:

- **INPUT_ROIS_PER_IMAGE**: Número máximo de etiquetas (o ground truth) en cada imagen.
- **IMAGE_WIDTH**: Corresponde al ancho a redimensionar y rellenar sobre las imágenes de entrada
- **IMAGE_HEIGHT**: Corresponde al alto a redimensionar y rellenar sobre las imágenes de entrada

En estos parámetros está incluida la función de perdida smooth L1, la cual, se puede cambiar el valor de sigma (σ) para la RPN y del detector independientemente:

- **SIGMA_RPN_L1**: Valor de sigma de la función de perdida smooth L1 a aplicar en la RPN.
- **SIGMA_DET_L1**: Valor de sigma de la función de perdida smooth L1 a aplicar en el detector.

El CNTK también incluye el método de eliminación de bounding boxes non-maximum suppression (NMS) (Ejemplo en la **Figura 22**), cuyos umbrales se modifican con los parámetros a continuación:

- **RESULTS_NMS_THRESHOLD**: Umbral NMS usado para el descarte de las predicciones de bounding boxes solapados en la evaluación.
- **RESULTS_NMS_CONF_THRESHOLD**: Valor de umbral, si el bounding box se encuentra por debajo de este umbral se considera 'fondo'.

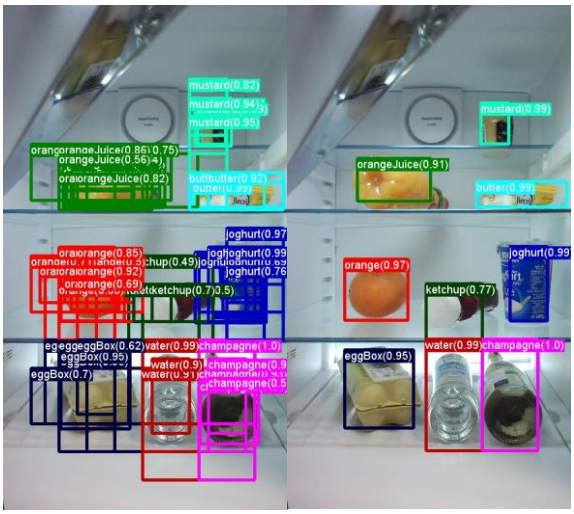

Figura 22 - Ejemplo Detecciones Sin Aplicar NMS (Izquierda) y Aplicando NMS (Derecha) ***[24]***

Los parámetros siguientes, son parámetros de visualización del sistema. Indica que queremos que se muestre en los resultados de las imágenes procesadas.

- **VISUALIZE_RESULTS**: Activar para ver las imágenes con los resultados proporcionados por el sistema.
- **DRAW_NEGATIVE_BOXES**: Activar para ver en las imágenes de resultado los bounding boxes negativos.
- **DRAW_UNREGRESSED_ROIS**: Activar para observar en las imágenes de resultado los rois no regresados.
- **RESULTS_BGR_PLOT_THRESHOLD**: Solo para la visualización de resultados en las imágenes, score mínimo para no considerar 'fondo' a las regiones devueltas.

Los parámetros de entrenamiento están divididos en tres secciones, parámetros generales de entrenamiento, parámetros de la RPN y parámetros del detector.

Los parámetros generales de entrenamiento incluyen los siguientes dos:

- **USE_FLIPPED**: Activar para, en fase de entrenamiento, también utilizar las muestras volteadas horizontalmente en el aprendizaje.
- **TRAIN_CONV_LAYERS**: Si está activo las capas convolucionales del modelo base pre-entrenado también serán entrenadas.

La RPN tiene sus propios parámetros en el CNTK, los cuales, se corresponden con los mostrados a continuación:

- **RPN_POSITIVE_OVERLAP**: Valor del umbral a superar para ser considerada muestra positiva.
- **RPN_NEGATIVE_OVERLAP**: Valor del umbral a superar para no ser considerada muestra negativa.
- **RPN_CLOBBER_POSITIVES**: Si está activo, un bounding box que cumple las condiciones de ser muestra positiva y negativa simultáneamente, se marca como negativo.
- **RPN_FG_FRACTION**: Valor decimal entre 0 y 1 inclusive, que indica el máximo número de muestras que no son 'fondo' (foreground examples)
- **RPN_BATCHSIZE**: Número máximo de muestras totales.
- **RPN_NMS_THRESH**: Umbral NMS para la eliminación de bounding boxes por debajo de ese umbral.
- **RPN_PRE_NMS_TOP_N**: Numero de regiones con score más importante a mantener de las propuestas proporcionadas por la RPN, previa aplicación de la NMS.
- **RPN_POST_NMS_TOP_N**: Numero de regiones con score más importante a mantener de las propuestas proporcionadas por la RPN, después de la aplicación de la NMS.
- **RPN_MIN_SIZE**: Tamaño mínimo de las regiones a detectar en la imagen. El ancho y el alto tiene que ser mayor que este valor a escala original de la imagen.

En cuanto a los parámetros del detector, tenemos la siguiente configuración de parámetros a indicar:

- **NUM_ROI_PROPOSALS**: Tamaño del mini-batch a entrenar, numero de imágenes a entrenar en cada paso del epoch.
- **FG_FRACTION**: Proporción del mini-batch que tiene muestras etiquetadas.
- **FG_THRESH**: Umbral mínimo de solape para que una región sea considerada positiva.
- **BG_THRESH_HI**: Umbral máximo de solape para ser considerada una muestra como 'fondo'.
- **BG_THRESH_LO**: Umbral mínimo de solape para ser considerada una muestra como 'fondo'.

Para los bounding boxes también obtenemos varios parámetros configurables para hacer la normalización:

- **BBOX_NORMALIZE_TARGETS**: Activar para normalizar los bounding boxes objetivos.
- **BBOX_NORMALIZE_MEANS**: Normalizar las medias de las regiones según los valores indicados.
- **BBOX_NORMALIZE_STDS**: Normalizar las desviaciones estándar de las regiones según los valores indicados.

Los parámetros de evaluación (o testing), definen varios parámetros para la fase de evaluación del modelo ya entrenado:

- **RPN_NMS_THRESH**: Umbral NMS para la eliminación de bounding boxes por debajo de ese umbral.
- **RPN_PRE_NMS_TOP_N**: Numero de regiones con score más importante a mantener de las propuestas proporcionadas por la RPN, previa aplicación de la NMS.
- **RPN_POST_NMS_TOP_N**: Numero de regiones con score más importante a mantener de las propuestas proporcionadas por la RPN, después de la aplicación de la NMS.
- **RPN_MIN_SIZE**: Tamaño mínimo de las regiones a detectar en la imagen. El ancho y el alto tiene que ser mayor que este valor a escala original de la imagen.

Finalmente, para los parámetros de configuración del detector de objetos, tenemos varios parámetros extra, que son los siguientes:

- **RND_SEED**: Semilla aleatoria para reproducibilidad.
- **USE_GPU_NMS**: Activar para utilizar la implementación en GPU de la non-maximum suppression (NMS).
- **GPU_ID**: Id de GPU por defecto.

Como se puede observar mediante los múltiples parámetros mencionados anteriormente, el CNTK es una buena herramienta, ya que, proporciona el detector de objetos Faster R-CNN con multitud de parámetros configurables. La integración de estos parámetros con el sistema, a la hora de querer crear, entrenar, validar o ejecutar

nuestro modelo, es de gran ayuda porque nos permite modificar multitud de opciones del sistema sin tener que reescribir código o similar.

Pero también tiene limitaciones, una de ellas es el reescalado de la imagen. Por defecto, el ancho y el alto de la imagen de entrada está marcado a 850x850 mediante los parámetros de **IMAGE_WIDTH** y **IMAGE_HEIGHT** respectivamente. En cambio, si decides modificar estos valores, se redimensiona la imagen, pero solo produce detecciones en una zona de la imagen, incluso llegando a recortar los bounding boxes que no entren en esa zona. Un ejemplo con un valor de **IMAGE_WIDTH** = 425 y **IMAGE_HEIGHT** = 425 se puede observar en la **Figura 23**.

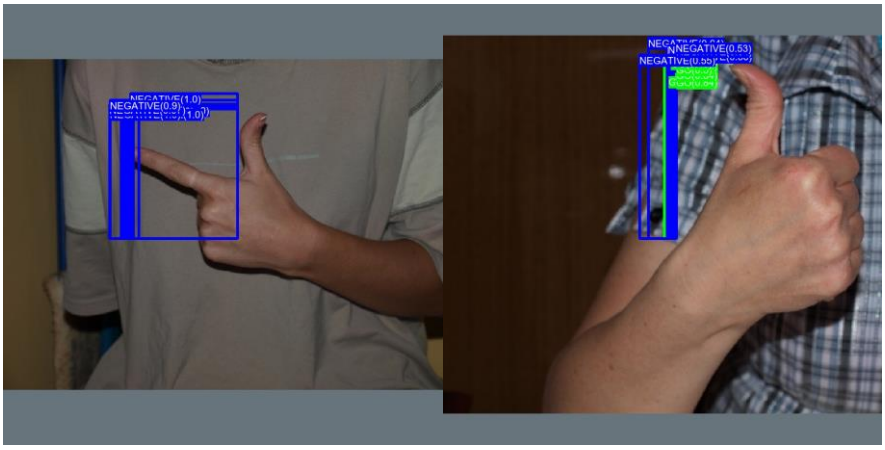

Figura 23 - Ejemplo Detecciones CNTK Imagen Reescalada 425x425

3.1.1.2 Parámetros del Dataset

El conjunto de datos a ejecutar también tiene un fichero de configuración asociado con varios parámetros a indicar, estos parámetros son los siguientes:

- **DATASET**: Nombre del dataset.
- **MAP_FILE_PATH**: Ruta a la carpeta que contiene los datos del dataset.
- **CLASS_MAP_FILE**: Nombre del mapa de clases del dataset. Contiene la correspondencia entre el nombre de la etiqueta en las imágenes y un id único.
- **TRAIN_MAP_FILE**: Nombre del archivo de texto con la ruta de las imágenes a entrenar.
- **TRAIN_ROI_FILE**: Nombre del archivo de texto con la ruta de las anotaciones o etiquetas de las imágenes a entrenar.
- **TEST_MAP_FILE**: Nombre del archivo de texto con la ruta de las imágenes a evaluar,
- **TEST_ROI_FILE**: Nombre del archivo de texto con la ruta de las anotaciones o etiquetas de las imágenes a evaluar.
- **NUM_TRAIN_IMAGES**: Numero de imágenes a entrenar.
- **NUM_TEST_IMAGES**: Numero de imágenes a evaluar.

- **PROPOSAL_LAYER_SCALES**: Vector de las escalas a aplicar para obtener los anchors de la red. Solo utilizada en la función generate_anchors() del CNTK.
- **TRAIN_PRECOMPUTED_PROPOSALS_FILE**: Nombre del archivo de texto con las propuestas precalculadas a entrenar (dataset Pascal VOC).
- **TEST_PRECOMPUTED_PROPOSALS_FILE**: Nombre del archivo de texto con las propuestas precalculadas a evaluar (dataset Pascal VOC).

3.1.1.3 Parámetros del Modelo Base

En cuanto al modelo base, para ejecutar los modelos se necesita configurar los parámetros asociados, los cuales, se detallan a continuación:

- **BASE_MODEL**: Nombre del modelo base.
- **BASE_MODEL_FILE**: Archivo con la estructura del modelo base.
- **IMG_PAD_COLOR**: Vector que define el color del margen añadido a las imágenes después del reescalado.
- **FEATURE_NODE_NAME**: Nombre del nodo de características.
- **LAST_CONV_NODE_NAME**: Nombre del nodo de la última convolución. Nodo utilizado como mapa de características para el procesamiento, incluyendo la RPN.
- **START_TRAIN_CONV_NODE_NAME**: Nombre del nodo convolucional desde donde comenzar el entrenamiento, los nodos anteriores permanecerán congelados.
- **POOL_NODE_NAME**: Nombre del nodo de pooling donde se empiezan a clonar las capas fully-connected.
- **LAST_HIDDEN_NODE_NAME**: Nombre del ultimo nodo oculto.
- **FEATURE_STRIDE**: Numero de saltos de las características.
- **RPN_NUM_CHANNELS**: Número de canales de la RPN.
- **ROI_DIM**: Dimensiones de la región de interés.

En teoría, podemos descargar varios modelos base para la ejecución con el CNTK. Y cambiando la configuración de los parámetros anteriormente detallados, siendo los correspondientes al nuevo modelo, conseguiríamos ejecutar el nuevo modelo base.

3.1.2 Entrenamiento

El entrenamiento del modelo conlleva varias partes, una es la configuración de los parámetros comentados en el apartado anterior. Pero para poder empezar con el entrenamiento, a parte de estas configuraciones, se necesita crear la estructura de carpetas con las que alimentar de datos a la red (dataset) y el modelo base que incluirá el Faster R-CNN. Además, el CNTK cuenta con su propio formato para los datos de entrada, por lo que debemos tener los datos tal cual los necesita.

Por tanto, una vez configurado el CNTK con los ficheros de configuración correspondientes, es necesario tener un modelo base, el cual, será el que utilice el Faster R-CNN. Para que el CNTK pueda utilizar estos modelos base, necesita que le proporcionemos un archivo con extensión '.model', que corresponda con alguno de los modelos compatibles, en la carpeta 'Pretrained Models' de la raíz del propio CNTK. Cabe destacar que el CNTK no admite los modelos ResNet **[23]**.

Una vez tenemos las configuraciones y el modelo base a ejecutar, necesitamos proporcionar el dataset a ejecutar en el formato adecuado del CNTK. El dataset a ejecutar debe tener una estructura de carpetas y archivos, la cual, consta de tres carpetas iniciales, 'negative', 'positive' y 'testImages', y cuatro ficheros de texto 'class_map.txt', 'train_img_file.txt', 'train_roi_file.txt', 'test_img_file.txt' y 'test_roi_file.txt'.

Las carpetas contienen las imágenes y etiquetas que se introducirán en el entrenamiento del sistema. Las etiquetas (ground truth) están formadas por dos ficheros de texto de extensión '.bboxes' y '.bboxes.labels'. Estos ficheros contienen información de los bounding boxes de las imágenes y las clases pertenecientes a esos bounding boxes, cada imagen tiene sus dos ficheros de etiquetas asociados. El archivo de extensión '.bboxes' organiza por filas las coordenadas (x_min, y_min, x_max, y_max) separadas en columnas de cada bounding box que aparece en la imagen. En cambio, el fichero de extensión 'bboxes.labels' organiza por filas las clases pertenecientes a los bounding boxes del fichero de extensión 'bboxes', creando una correspondencia por filas entre los dos ficheros. En la **Tabla 23** podemos observar la correspondencia de los archivos, la primera fila del archivo '.bboxes' correspondiente al primer bounding box de la imagen pertenece a la clase 'NEGATIVE', mientras que la segunda región pertenece a la clase 'GO'.

Archivo '.bboxes'				Archivo '.bboxes.labels'
1233	255	1273	327	NEGATIVE
1099	240	1130	294	GO

Tabla 23 - Ejemplo Ficheros '.bboxes' y '.bboxes.labels'

La carpeta 'negative' contiene todas las fotos que no hayan sido etiquetadas, es una carpeta de descarte que no se tiene en cuenta en la fase de entrenamiento. La carpeta 'positive' contiene todas las imágenes a entrenar, junto con sus etiquetas correspondientes. Por último, la carpeta 'testImages', igual que la carpeta 'positive', contiene las imágenes con sus etiquetas asociadas, con la excepción de que estas imágenes no se usaran en la fase de entrenamiento, se utilizaran a posteriori para evaluar el rendimiento del sistema entrenado.

Una vez tengamos las carpetas con las imágenes y etiquetas de entrenamiento y evaluación, necesitamos crear los ficheros de texto 'class_map.txt', 'train_img_file.txt', 'train_roi_file.txt', 'test_img_file.txt' y 'test_roi_file.txt'. Para ello, el CNTK te proporciona

una herramienta llamada 'annotations_helper.py' que indicándole la ruta del dataset, te crea esos ficheros. Estos ficheros son los que lee el sistema a la hora de obtener las imágenes y las etiquetas. El 'class_map.txt' asigna un id a cada clase vista en el dataset, por defecto el id 0 siempre corresponde a la clase '__background__'. Un ejemplo del contenido del archivo se puede ver en la **Tabla 24**.

Archivo 'class_map.txt'	
Clase	**ID**
__background__	0
NEGATIVE	1
GO	2
READY	3
STORE	4

Tabla 24 - Ejemplo Fichero 'class_map.txt'

Los ficheros 'train_img_file.txt' y 'test_img_file.txt' contienen la ruta donde se encuentran las imágenes de entrenamiento y evaluación respectivamente. Y los archivos 'train_roi_file.txt' y 'test_roi_file.txt' contienen las coordenadas y el id de clase de todos los bounding boxes de la imagen. Un ejemplo de estos ficheros se puede ver en la **Tabla 25** y **Tabla 26**.

'train_img_file.txt'			'train_roi_file.txt'						
ID	**Ruta**	**ID2**	**ID**	**Descripción**	**x_min**	**y_min**	**x_max**	**y_max**	**ID Clase**
0	positive/train_img_1.jpg	0	0	|roiAndLabel	1233.0	255.0	1273.0	327.0	1.0
1	positive/train_img_2.jpg	0	1	|roiAndLabel	745.0	341.0	791.0	428.0	2.0
2	positive/train_img_3.jpg	0	2	|roiAndLabel	691.0	328.0	734.0	399.0	2.0

Tabla 25 - Ejemplo Formato Ficheros 'train' y 'test' de entrenamiento

‘test_img_file.txt’			‘test_roi_file.txt’						
ID	**Ruta**	**ID2**	**ID**	**Descripción**	**x_min**	**y_min**	**x_max**	**y_max**	**ID Clase**
0	testImages/test_img_1.jpg	0	0	|roiAndLabel	1289.0	326.0	1335.0	403.0	1.0
1	testImages/test_img_2.jpg	0	1	|roiAndLabel	1258.0	478.0	1309.0	565.0	3.0
2	testImages/test_img_3.jpg	0	2	|roiAndLabel	920.0	403.0	961.0	469.0	4.0

Tabla 26 - Ejemplo Formato Ficheros 'train' y 'test' de entrenamiento

Una vez tengamos hechos estos pasos, al ejecutar el archivo ‘run_faster_rcnn.py’, comenzará el entrenamiento del Faster R-CNN con la configuración indicada (ver **Figura 24**).

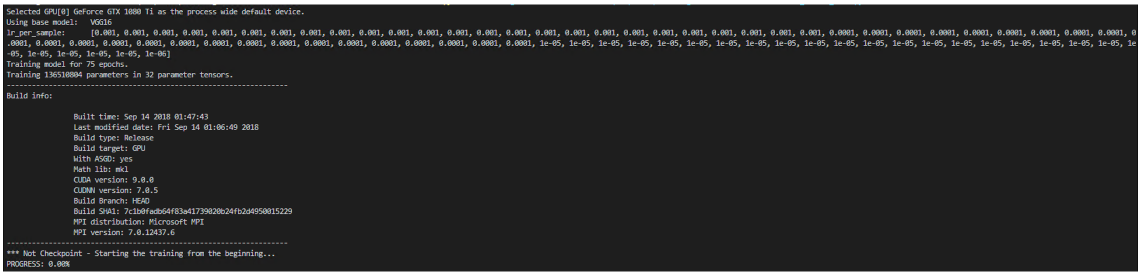

Figura 24 - CNTK Faster R-CNN: Entrenamiento Iniciado

3.1.3 Ejecución

El CNTK proporciona un script en Python para la ejecución del modelo entrenado. Este programa carga los ficheros de configuración y el modelo entrenado. En caso de no existir un modelo entrenado comienza el entrenamiento. Una vez acabado el entrenamiento o en el caso de que ya exista un modelo entrenado, a partir de una ruta a la imagen a procesar indicada en el código, ejecuta el modelo sobre la imagen y los resultados de la detección los imprime en la imagen y guarda una copia en la ruta de salida proporcionada.

Estas detecciones se muestran en la imagen con un recuadro alrededor del objeto detectado de color único para cada clase. También incluye un texto con la predicción de la correspondiente clase en la esquina superior izquierda del bounding box. Además, entre paréntesis, junto al texto de la clase, aparece el nivel de confianza (score) asociado a dicha clase (ver **Figura 25**).

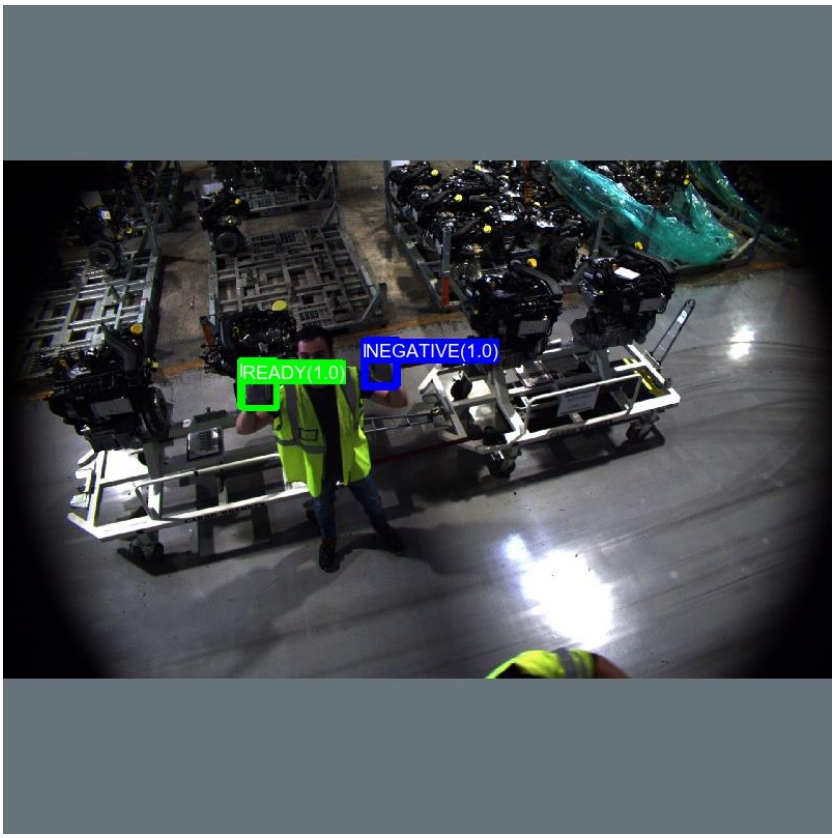

Figura 25 - Ejemplo Detección Faster R-CNN CNTK

3.1.4 Cambios Realizados

A modo de base el CNTK está bastante bien, incluye múltiples herramientas y estructuras para entrenar un modelo con mayor facilidad. Pero a la hora de desarrollar un sistema surgen diferentes necesidades, las cuales, hay varias que el CNTK no las contempla. Por ese motivo, se deben añadir desde cero estas lagunas que contiene previamente este entorno.

3.1.4.1 Visualización Progreso y Tiempo Restante de Entrenamiento

El primer cambio realizado fue la visualización del progreso y tiempo restante de entrenamiento. Cuando entrenas por primera vez el Faster R-CNN, la librería del CNTK muestra en la línea de comandos el texto **'PROGRESS: 0.00%'**. Si indagas por el código puedes encontrar que la función que se utiliza para mostrar el progreso de entrenamiento ejecuta una línea de código para mostrar ese texto fijo continuamente, cada 10 segundos aproximadamente. Por lo que, esto imposibilitaba la tarea de conocer cuánto tiempo faltaba para terminar el entrenamiento y que porcentaje de entrenamiento había completado.

Entonces, para solucionar esto, se implementaron los cálculos correspondientes para en base al número de muestras a ejecutar en cada batch, el número de epochs y el estado actual, se visualizará el porcentaje de progreso completado del entrenamiento, el tiempo aproximado en horas restante, el epoch actual del total y el progreso completado del epoch actual. En la **Figura 26** se puede observar un ejemplo en la salida de la consola con la visualización actualizada.

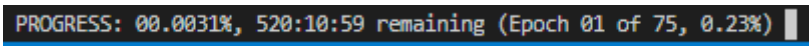

Figura 26 - ProgressPrinter Actualizado

3.1.4.2 Carga de Imágenes en Memoria

El segundo cambio se efectuó para la mejora en la velocidad del sistema. El Faster R-CNN del CNTK, en fase de ejecución del algoritmo entrenado, obtenía la ruta de cada imagen y las cargaba de disco una a una. Eso implicaba muchas llamadas a disco, con su correspondiente pérdida de tiempo de acceso a disco.

Con el objetivo de eliminar el tiempo empleado en los múltiples accesos a disco, se cambió la estructura interna del CNTK para almacenar todas las imágenes en un vector inicialmente y solo acceder a disco en la carga de las imágenes la primera vez. Con esto conseguíamos aumentar la velocidad de detección, eliminando este tiempo del proceso de detección de cada imagen.

3.1.4.3 Puntos de Restauración del Entrenamiento, Modelo Intermedio y Almacenamiento de Resultados de Evaluación.

El CNTK tampoco disponía de un guardado de seguridad del modelo en fase de entrenamiento o de un punto de restauración (checkpoint). Esto suponía un problema, ya que, el proceso de entrenamiento del algoritmo, en el caso de contener una cantidad de imágenes importante en el dataset, puede llegar a tardar días en terminar. Por lo que, un corte de luz, una parada inesperada del entrenamiento o la necesidad de la utilización de la gráfica del equipo por otro sistema provocaba la perdida de todo el entrenamiento realizado hasta el momento, con la consiguiente necesidad del reinicio del entrenamiento desde cero y la pérdida del tiempo empleado en el entrenamiento.

Para solucionar este problema, se implementaron dos herramientas: Creación de puntos de restauración del entrenamiento y guardado del modelo entrenado en el estado actual, denominado modelo intermedio. Además, el almacenamiento de los resultados de la evaluación en un archivo externo y la posibilidad de guardar las imágenes con las detecciones de los modelos intermedios. Para ello, se tuvieron que crear varios parámetros extra en el fichero de configuración del detector, los cuales, son los siguientes:

- **RESTORE_CHECKPOINT_NAME**: Nombre del checkpoint a restaurar, si el nombre existe, se restaura el entrenamiento. En caso contrario el entrenamiento se inicia desde cero.
- **EPOCHS_TO_SAVE_CHECKPOINT**: Cada cuantos epochs se creará un punto de restauración y un modelo entrenado intermedio.
- **TEST_MODE_IN_INTERMEDIATE_MODELS**: Activar si en la creación de los modelos intermedios se quiere almacenar las imágenes con las detecciones devueltas por el modelo. Esto permite observar directamente las predicciones realizadas por el modelo para el estado actual.

Estos parámetros se implementaron en el código en el proceso de entrenamiento para que tuvieran efecto al ser utilizados. El checkpoint, se creó codificando el epoch actual en el nombre del fichero para restaurarlo automáticamente en caso de reinicio del entrenamiento. También se incluyó el almacenamiento de los datos de evaluación producidos por el modelo, para tener constancia del rendimiento del modelo en el estado actual (ver **Figura 27**).

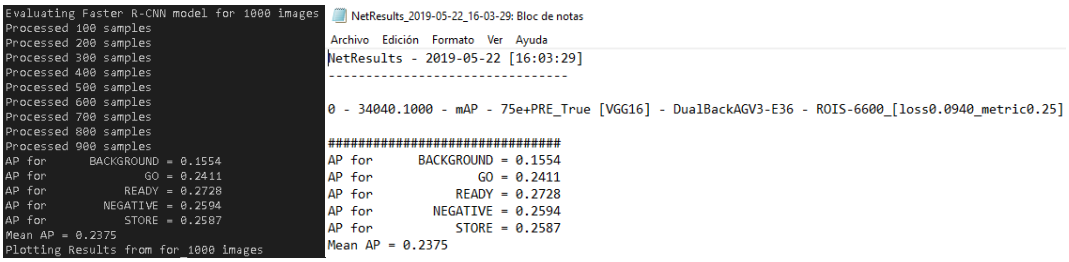

Figura 27 - Izquierda: Evaluación en consola, Derecha: Evaluacion almacenada en fichero

3.1.4.4 Script Python de Segundo Filtrado de Detecciones

El sistema producía varios falsos positivos y muchas detecciones sobre el mismo objeto, con el objetivo de paliar esto se realizó un segundo filtro escrito en Python que obtenía solamente la cantidad indicada de gestos de cada clase. Es decir, se le indicó que filtrase todas las detecciones y solo mostrase las detecciones con más score de cada clase. En nuestro caso, este valor se fijó en 1 para obtener como mucho un gesto de cada clase en la imagen y el de mayor puntuación (ver **Figura 28**).

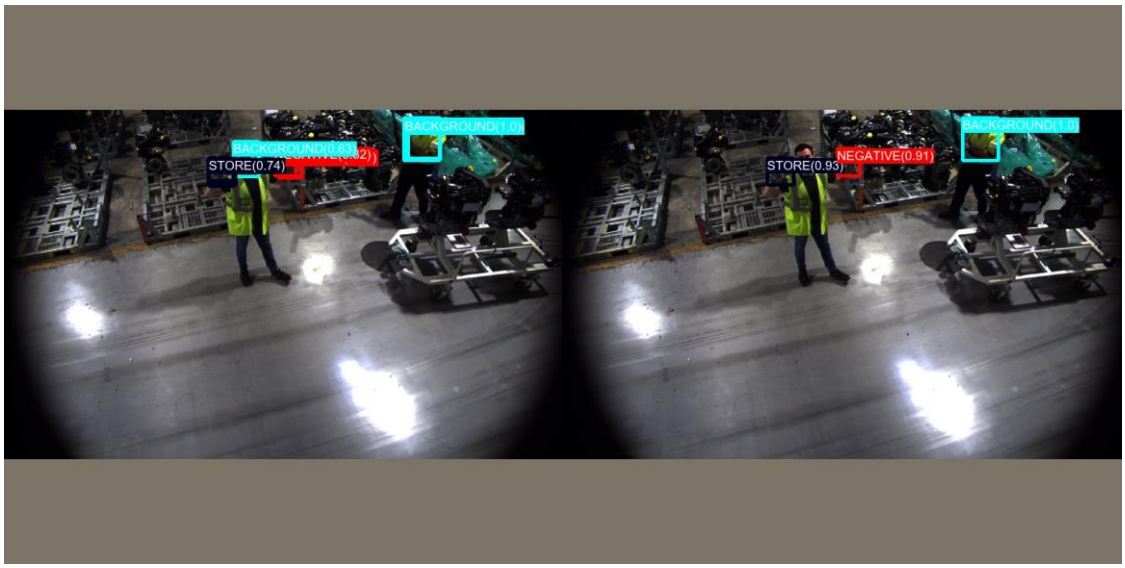

Figura 28 - Pre-Aplicación Segundo Filtro (Izquierda) y Post-Aplicación Segundo Filtro (Derecha)

3.1.4.5 Cambio Modelo Base

El CNTK viene configurado para usarse con los modelos base AlexNet **[31]** y VGG16 **[14]**. En teoría, según indican en la documentación **[23]** se pueden descargar diferentes modelos y modificando el script de configuración se podrían aplicar como modelos base. Pero a la hora de configurarlos, no es capaz de reconocerlos y no es posible ejecutarlos.

Se hicieron multitud de pruebas, con varias herramientas diferentes que extraen los nombres de los nodos de los modelos base, pero a la hora de introducir los nombres en el fichero de configuración del modelo base no los reconoce por alguna razón y no se pueden utilizar. Tanto la herramienta del propio sistema CNTK y como la herramienta externa Netron, nos permiten ver la información interna de los nodos de los modelos base, como se muestra en la **Figura 29** y **Figura 30**. A pesar de ello, después de aplicar los datos internos de los modelos en el fichero de configuración con sus nombres, seguía sin funcionar.

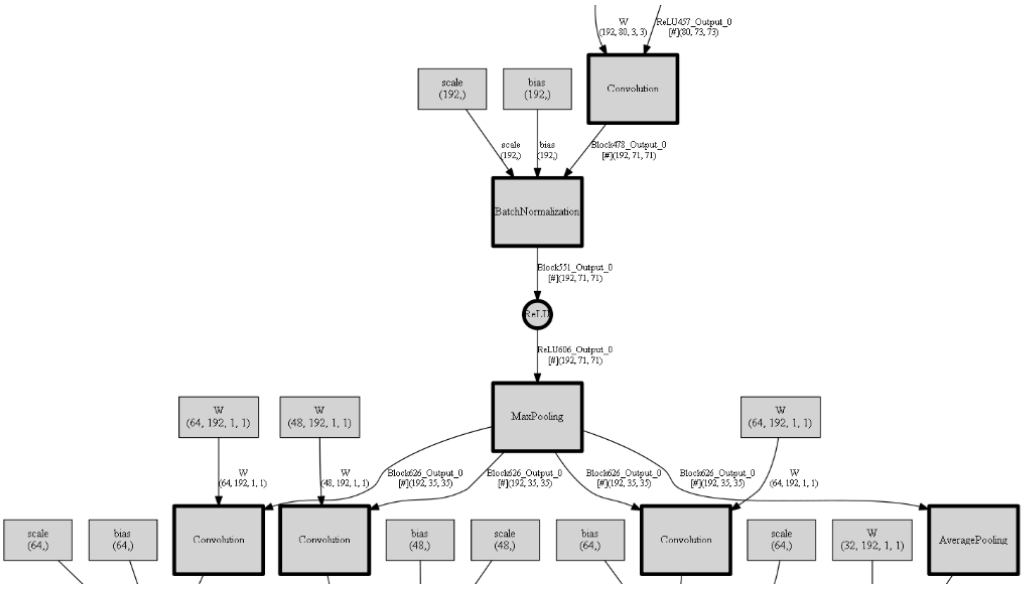

Figura 29 - Imagen Nodos Internos Modelo InceptionV3 con CNTK

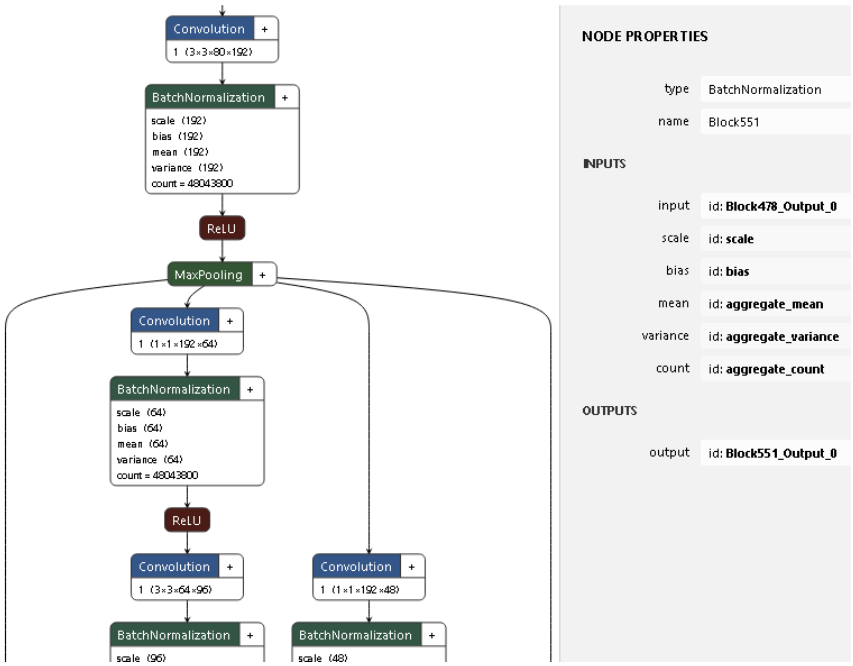

Figura 30 - Imagen Nodos Internos InceptionV3 con Netron

Cabe destacar que la herramienta de Netron para la visualización de las redes neuronales, es mucho más completa. Esta herramienta te permite navegar entre todos los nodos de la red, seleccionarlos y ver sus propiedades, además, de los valores que contienen estas propiedades.

3.1.4.6 Programas de Ejecución mediante Carpeta Virtual y Cámara

El programa de ejecución por defecto del CNTK se queda bastante corto para realizar pruebas más relevantes. Además, en este caso, necesitamos poder ejecutar el modelo entrenado para una cámara conectada directamente al ordenador y una carpeta virtual como proveedor de imágenes a detectar.

Para ello, se generó un script en Python con la implementación de las dos opciones. Para la conexión con la cámara, se utilizó OpenCV, el cual, a través de la función *VideoCapture* consigue obtener las imágenes directamente desde la cámara conectada al ordenador. Para poder identificar la cámara a la que conectarse le tenemos que indicar el id de la cámara, con el id 0 por defecto coge la primera. También se configuró para que pudiese ser compatible con una cámara IP, obtenía las imágenes a través de la IP asociado a una cámara concreta. Una vez implementado lo anterior, las imágenes se procesan con el modelo entrenado, se obtienen los resultados y se muestran por pantalla y/o se almacenan en disco las imágenes con las detecciones.

Para el caso de asignar una carpeta virtual como proveedor de imágenes, se tuvo que crear un programa que previamente cargara las imágenes en memoria, y según iban procesándose, mostrara por pantalla y/o almacenara las imágenes con sus respectivas detecciones de objetos. Este método de ejecución nos facilitó hacer pruebas en un entorno más real o con imágenes de las instalaciones finales donde iba a estar ubicado el sistema, ya que, contábamos con una base de datos de imágenes del emplazamiento final. Lo que nos daba información de cómo podría funcionar el modelo entrenado una vez estuviera integrado en las instalaciones finales.

3.1.4.7 Ampliación Limite de Clases a Detectar

Otro fallo que tenía el CNTK era la asignación de colores a las detecciones de las imágenes. El CNTK proporciona un color a cada clase de los bounding boxes, pero el número de colores no era suficiente para un valor elevado de clases (27, por ejemplo), y el programa producía un error. La solución a esto fue modificar la función que generaba los colores que se asociaban a las clases, e incluir un abanico más amplio para poder mostrar más clases a la hora de dibujar las detecciones en las imágenes.

3.1.4.8 Evolución de Parámetros en TensorBoard

También se incluyeron varios parámetros para poder visualizarse desde la herramienta TensorBoard. En cuanto a la precisión del modelo y la función de pérdida del modelo en fase de entrenamiento, era conveniente saber cómo evolucionaba y ver los resultados proporcionados. Por lo que, se modificó el código original del CNTK para que estos valores los devolviera en tiempo de ejecución del entrenamiento. De esta manera, podíamos verificar continuamente si la configuración del detector de objetos era adecuada o había que realizar algún cambio (ver **Figura 31**).

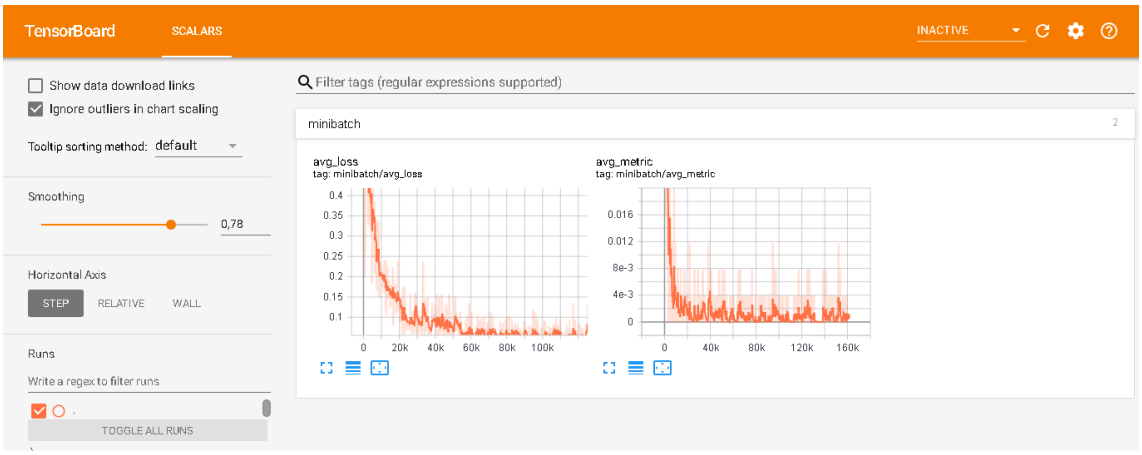

Figura 31 - Evolución Variables avg_loss y avg_metric TensorBoard

3.1.4.9 Profiler de Velocidad de Detección

El profiler es una herramienta del CNTK que por defecto no está incluida en el sistema. Es una herramienta muy útil para observar el tiempo que tarda en procesar una imagen el modelo, proporciona los datos calculando la media de varias ejecuciones. Por esta razón, se decidió implementar dicha herramienta en el código original del CNTK.

Para ello, debemos escribir en el código las siguientes funciones predefinidas de la librería 'cntk': *start_profiler(), enable_profiler()* y *stop_profiler().* El *start_profiler()* indica cuando va a empezar el profiler a ejecutarse, en este caso se ha posicionado justo antes del comienzo del entrenamiento. El *enable_profiler()* es la función que tenemos que ejecutar en el momento que queramos que se ejecute el profiler, se ejecuta después del primer epoch y en nuestro caso lo ejecutamos al final de cada epoch. Por último, el *stop_profiler()* se utiliza para indicar al profiler que no siga ejecutándose, en nuestro caso se colocó al final del entrenamiento. En caso de que el entrenamiento sea muy largo el profiler se desactivará automáticamente.

4. Hardware Empleado

El desarrollo de este proyecto conlleva múltiples elementos de hardware y herramientas. En este capítulo se detallarán las herramientas empleadas y adquisición de las imágenes de entrenamiento y evaluación del modelo mediante las herramientas detalladas.

4.1 Herramientas

Estos sistemas de redes neuronales requieren una cantidad de computo muy elevada. Antes de la aparición de las tarjetas gráficas (GPUs), era inviable diseñar un sistema basado en redes neuronales, ya que, los cálculos necesarios para completar el entrenamiento no se podían satisfacer con la tecnología disponible. La llegada de las tarjetas gráficas supuso un cambio en esta área, debido a que estas eran capaces de acelerar el proceso gracias a la cantidad de computo que podían proporcionar.

Además, entrenar estos métodos de redes neuronales convolucionales conlleva la necesidad de disponer de multitud de imágenes. Para ello, se necesitan cámaras para adquirir dichas imágenes.

Por lo demás, con un ordenador capaz de integrar estos elementos, se puede comenzar a trabajar en crear un sistema de este tipo.

4.1.1 Cámaras

Las cámaras son un elemento importante en este proyecto, ya que sin ellas no podríamos desarrollar el modelo entrenado ni integrar el reconocimiento de gestos final. Para este proyecto, se han utilizado las siguientes:

- Cámara IPhone
- Basler acA1920-48gc

La segunda cámara de la lista es la que finalmente se instalará en la ubicación final, Basler (ver **Figura 32**).

Figura 32 - Cámara Basler acA1920-48gc

Las cámaras Basler son ideales para nuestro objetivo final, ya que, están diseñadas para integrarse en entornos industriales. Son cámaras monofocales que se pueden conseguir en diferentes resoluciones y con diferentes monturas. Las ópticas son intercambiables y existen multitud de variantes a elegir. Dispone de dos versiones diferentes de adquisición de imágenes, la que produce imágenes a color y la monocromática que obtiene imágenes en blanco y negro, la cual, tiene un precio menor.

4.1.2 Tarjetas Graficas

A lo largo del desarrollo del reconocedor gestual, se utilizaron varias tarjetas gráficas. Estas tarjetas gráficas son las siguientes:

- NVIDIA Quadro P5000
- NVIDIA GTX 1070ti
- NVIDIA GTX 1080ti

Se puede observar que la tecnología predominante es la de la marca NVIDIA. Esto es debido a que, para el proceso de aceleración de estos sistemas, esta marca cuenta con su propia plataforma de computación que permite resolver cálculos de forma paralela, denominada CUDA **[25]**. Además, sus tarjetas gráficas cuentan con múltiples CUDA cores, los cuales, actúan como los núcleos de los procesadores, pero para las tarjetas gráficas, con la salvedad de que son más pequeños y se puede aumentar la cantidad de ellos en menos espacio **[26]**.

NVIDIA para el desarrollo de sistemas basados en redes neuronales también dispone de una librería específica, NVIDIA CUDA Deep Neural Network library (cuDNN) **[32]**. La cuDNN es una librería para la aceleración de la GPU para la ejecución de elementos de redes neuronales profundas, como las capas de activación, las capas de pooling, y las convoluciones forward y backward. Acepta múltiples entornos de deep learning como TensorFlow, PyTorch o Theano entre otros. El CNTK utilizado para este proyecto también tiene la necesidad de tener .

Estas librerías y herramientas tienen el inconveniente de al ser desarrolladas por NVIDIA solo son compatibles con hardware de la marca. Por ejemplo, las tarjetas

gráficas de AMD no pueden ejecutar dichos programas. Por lo que, la mayoría de los sistemas relacionados con redes neuronales trabajan con tarjetas gráficas NVIDIA para poder utilizar esta opción de desarrollo.

De las anteriores GPUs mencionadas, la que mejor rendimiento y velocidad proporcionaba es la NVIDIA GTX 1080ti. El emplazamiento final donde se ubicará el sistema de reconocimiento gestual posee la segunda opción de la lista, una NVIDIA GTX 1070ti (ver **Figura 33**).

Figura 33 - Tarjeta Gráfica NVIDIA: PNY GeForce GTX 1070ti

4.2 Adquisición de Fotos

La adquisición de fotos en el proyecto se llevó a cabo en dos fases y entornos diferentes.

La primera fase corresponde a la adquisición de imágenes con diferentes condiciones ambientales de personas efectuando gestos, en este caso consta de 3 personas distintas. Este conjunto de imágenes se adquirió mediante la cámara de un IPhone 6 disponible en la empresa. Un ejemplo de las imágenes obtenidas bajo las condiciones mencionadas se puede observar en la **Figura 34**.

El objetivo de estas imágenes era recopilar fotos muy diferentes entre sí de los gestos a detectar para el entrenamiento. Esto pretendía lograr una alta variación de los gestos y suministrar la capacidad al sistema que consiguiera hacer detecciones en todo tipo de ambientes y bajo condiciones muy diferentes.

Las imágenes se realizaron en 5 entornos ambientales diferentes. El primer entorno corresponde a dentro de la oficina, en varios lugares con diferente iluminación. El siguiente ambiente se encuentra en los alrededores de la oficina, perteneciente a la calle donde está ubicada. El tercero de los escenarios se localiza en los talleres internos de la oficina. Estos talleres están divididos en 3 zonas distintas. Cada zona

cuenta con sus iluminación y características propias. Las imágenes obtenidas en taller era lo más parecido que podíamos conseguir al resultado final donde se iba a implantar el sistema de detección de objetos. Esto era debido a que en el taller el fondo consta de mucho ruido y las condiciones lumínicas son más bajas con cierta similitud al emplazamiento final.

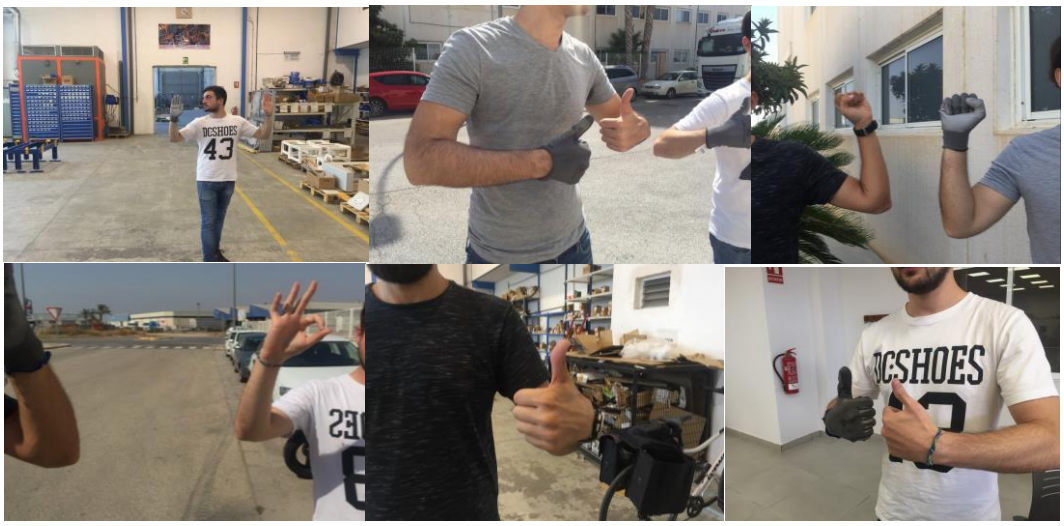

Figura 34 - Ejemplo Dataset Propio

La segunda fase de adquisición de imágenes fue ya en la planta de producción del cliente. La idea detrás de esto era dotar al sistema de detección de objetos de las características y condiciones reales que se iba a encontrar una vez estuviera en funcionamiento. Para ello, las cámaras ya instaladas en planta y diferentes personas realizando gestos, nos proporcionaron las imágenes del entorno real final. Un ejemplo de las fotos adquiridas se puede observar en la **Figura 35**. Estas imágenes fueron adquiridas mediante la cámara industrial Basler acA1920-48gc.

Las cámaras instaladas estaban a una altura superior a 1.80 metros, por lo que para enfocar a la zona de trabajo donde se realizará el gesto se tuvo que dar inclinación a las cámaras. Esta inclinación no era del todo adecuada para la detección. Dado que los usuarios que utilizasen el sistema tenían que realizar el gesto enfocando directamente el gesto hacia la dirección de la cámara. Esto provocaba que los gestos no siempre se hicieran perpendiculares a la cámara y por consiguiente tuvieran cierta inclinación.

Otro problema añadido es que el tamaño de los gestos en las imágenes era bastante reducido. Los pixeles correspondientes al gesto eran un porcentaje muy pequeño de la imagen, por lo que dificultaba la detección y aumentaba las probabilidades de detección de falsos positivos. Pero debido a especificaciones del proyecto se tuvo utilizar estas cámaras ya instaladas en el cliente y con dicha ubicación.

El área de movimiento donde se realizan los gestos para la ejecución de la adquisición de las imágenes concierne a todo el alrededor del carro. Además, se aprovecharon todos los límites de la imagen para poder proporcionar imágenes desde todas las perspectivas posibles.

Figura 35 - Ejemplo Dataset Propio en Planta

La adquisición de las fotos era un trabajo largo y costoso, ya que, se necesitaba mucho tiempo para recopilar todas las fotos necesarias. Además, sobre todo en esta segunda fase de adquisición de imágenes, estaba el coste añadido del desplazamiento al cliente para realizar las fotos del entorno real para después crear el dataset.

Las imágenes recopiladas en el entorno real resultaron ser imprescindibles para nuestro modelo entrenado. La mejoría más notoria que se pudo observar fue gracias a la adición de estas imágenes al conjunto de datos de entrenamiento del sistema. El obtener las características del ambiente final es trascendental para el sistema poder aprender a diferenciar el fondo particular del emplazamiento final.

5. Preproceso y Datasets

En este capítulo se detallará todo el trabajo realizado para el preproceso de las imágenes, como de la generación de los datasets posteriores creados mediante las imágenes

5.1 Preproceso

El preproceso de las imágenes es un trabajo costoso y conlleva una gran parte del tiempo del desarrollo de este tipo de sistemas. En los siguientes apartados se explica el trabajo de preproceso realizado correspondiente al etiquetado manual de las imágenes y el data augmentation posterior para aumentar la cantidad de imágenes a entrenar.

5.1.1 Etiquetado

El proceso de etiquetado de las imágenes es el más costoso en cuanto a tiempo de dedicación. Para crear los datasets de los que se alimentará el sistema de detección de objetos, es un paso previo indispensable. Necesitamos proporcionar el ground truth mediante el marcado de la localización del gesto y la clase a la que pertenece. Los gestos a detectar pertenecen a 4 clases diferentes (ver **Figura 36**):

- **READY**: Gesto con la palma de la mano en posición vertical. El objetivo de este gesto es empezar el proceso de salida.
- **GO**: Gesto con la mano mostrando el pulgar hacia arriba. Este gesto se utiliza posterior al 'READY' para confirmar la señal de salida.
- **STORE**: Gesto con los dedos índice y pulgar de la mano formando una 'L'. Gesto para añadir funcionalidad en el futuro si fuera necesario.
- **NEGATIVE**: Cualquier otro gesto realizado con las manos. El objetivo de este gesto es dotar al sistema de la capacidad de distinguir mejor los gestos positivos (los tres mencionados anteriormente).

Figura 36 - Clases Gestos: READY (Izquierda), GO (Centro), STORE (Derecha)

El CNTK proporciona dos herramientas simples para realizar este proceso. La primera es una herramienta que te permite crear los bounding boxes de los objetos de la imagen a entrenar. La segunda, a través de los bounding boxes marcados con la herramienta anterior, recorría todos los bounding boxes de las imágenes y te pedía indicar la clase a la que pertenecía cada uno. Esto a priori era útil si no disponías de más medios para hacer el etiquetado. El inconveniente estaba cuando la cantidad de imágenes a etiquetar era muy elevada. Este sistema implicaba recorrer todas las imágenes para marcar las regiones y después obligaba a volverlo a repasar para indicar las clases. Esto suponía un coste elevado de tiempo, ya que, el repaso de un dataset grande suponía una inversión de tiempo muy grande, y realizar una segunda pasada para indicar las clases elevaba este tiempo invertido.

Posteriormente, se descubrió el software de etiquetado Visual Object Tagging Tool (VOTT). Este software es más completo y con más funcionalidades. La mayor ventaja respecto del software del CNTK, es la reducción en tiempo de etiquetado gracias a poder etiquetar los bounding boxes y la clase a la que pertenecen al mismo tiempo. También incluye diferentes formatos de exportación aparte del CNTK, como Tensorflow PASCAL VOC o YOLO. Para ejecutar el programa tendremos que indicarle la ruta con las imágenes a aumentar y después nos aparecerá una ventana como la de la **Figura 37**, en la cual, tendremos que indicarle la forma de los bounding boxes a marcar y las clases a etiquetar.

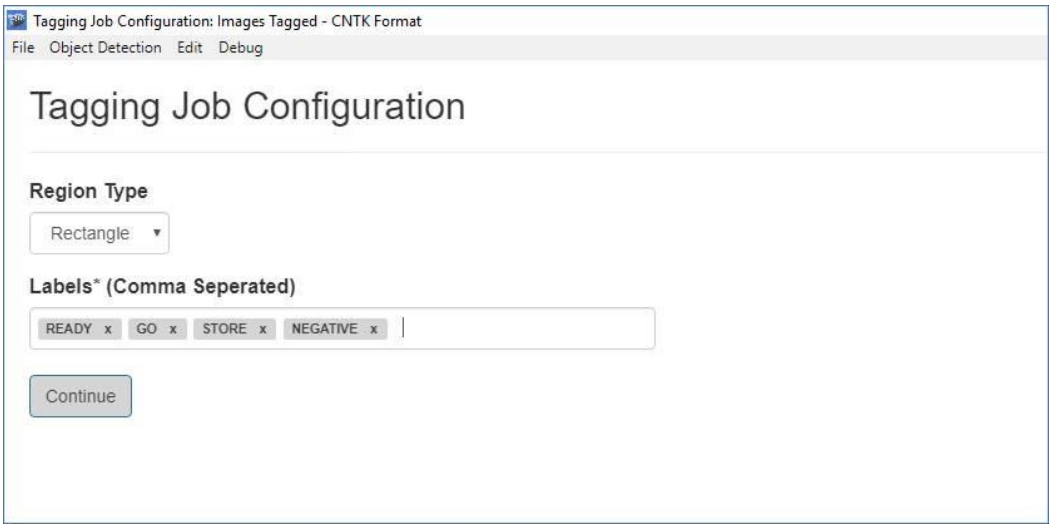

Figura 37 - VOTT: Configuración Etiquetas

Una vez tengamos esto, aparecerá la pantalla de etiquetado con la imagen a etiquetar. A continuación, recuadramos los objetos de la imagen, les asignamos su clase y pasamos a la siguiente imagen para guardar el resultado. En el ejemplo de la **Figura 38** se puede observar la imagen etiquetada con los bounding boxes y su clase 'GO' identificada con el color azul.

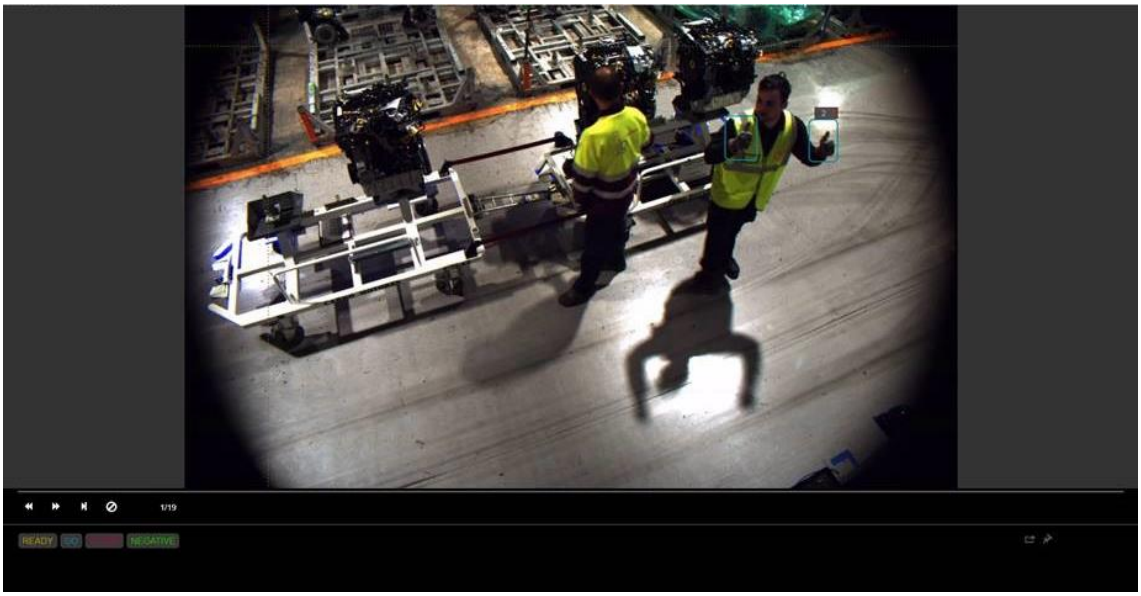

Figura 38 - Etiquetado VOTT: Ejemplo Imagen Etiquetada

Cuando hayamos finalizado el proceso de etiquetado tendremos la opción de exportar las etiquetas en los diferentes formatos. Este programa tiene la ventaja añadida de poder etiquetar un conjunto de imágenes y exportar hasta los datos

etiquetados. Además, las etiquetas se guardan automáticamente por lo que podemos reanudar el etiquetado en otro momento.

El software de etiquetado VOTT dispone de una nueva versión más completa con una nueva interfaz. Esta versión permite el etiquetado directamente en la nube, proporciona más formatos de exportación del etiquetado, atajos de teclado para una mayor velocidad de etiquetado y demás funcionalidades. **[27]**

5.1.2 Data Augmentation

El Data Augmentation se le conoce como al proceso de aumentar las fotos mediante la aplicación de transformaciones en la imagen original, consiguiendo así más variantes de una única foto. Este proceso es importante, debido a que el proceso de entrenamiento de una red neuronal convolucional requiere de muchas muestras de entrenamiento, pero generar todas esas muestras manualmente resulta un trabajo muy complicado. Por lo que en la creación de un sistema de este tipo siempre se recurre a este método de aumentación de los datos.

El problema surge que este método solo está implementado para utilizarse con problemas de clasificación. En nuestro caso, la aumentación de los datos incluye la transformación de los bounding boxes, por tanto, si modificamos la imagen también deberemos reajustar los bounding boxes. Para ello, se implementó un script en Python, el cual, nos permitía realizar dicho proceso. Este script finalmente se le otorgo la funcionalidad de transformar la imagen de varias formas distintas. Se le incluyeron las siguientes transformaciones:

- Rotación
- Inclinación
- Traslación
- Brillo
- Desenfoque
- Ruido Gaussiano
- Ruido Sal y Pimienta

El script funcionaba de la siguiente manera, a través de una ruta le indicábamos la carpeta con las imágenes y las etiquetas a aumentar. A continuación, debíamos introducir el nombre de la carpeta de destino donde se guardará el resultado de la aumentación y el número de veces que se aumentaría cada foto, es decir, el número de imágenes a generar de cada imagen en la carpeta de origen. Después, aparecen las opciones de aumentación una a una, las cuales, tendremos que indicar un valor máximo de aumentación para cada una de ellas, o un 0 si no se quiere utilizar ese tipo de aumentación en la ejecución (ver **Figura 39**). Finalmente, el programa empieza a generar las imágenes y las etiquetas aumentadas. Los datos de aumentación aplicados a la imagen se codifican en el nombre.

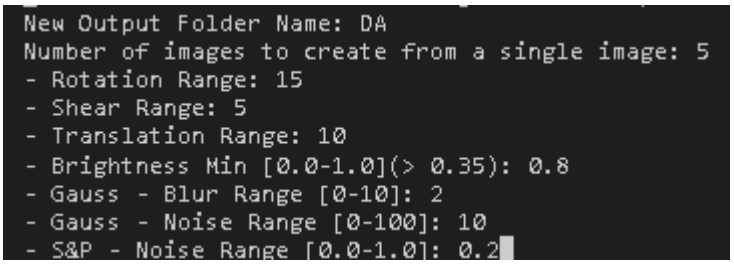

Figura 39 - Salida Consola: Ejemplo Ejecución Programa Data Augmentation

El funcionamiento interno del programa selecciona uno de los bounding boxes que aparecen en la imagen aleatoriamente como centro de transformación. A partir de este bounding box se aplicarán las transformaciones a la imagen. Se generaba una matriz del tamaño de la imagen y se indicaban en la matriz las coordenadas de los bounding boxes de la imagen. En un vector se almacenaba una matriz por imagen, ya que, cada imagen tiene su matriz con las coordenadas de los bounding boxes. Esta matriz de coordenadas se utiliza para aplicar las transformaciones a los bounding boxes. Entonces, se crea una copia de la imagen original y se le van aplicando las transformaciones correspondientes a la imagen al igual que a la matriz de coordenadas. Finalmente, la imagen transformada y los bounding boxes transformados con sus clases se exportaban en la carpeta de destino, y obteníamos la aumentación de los datos de las imágenes correspondientes. También existía la opción de generar una copia de la imagen con los recuadros aumentados superpuestos y observar el resultado del data augmentation (ver **Figura 40)**. Si la imagen no contenía bounding boxes se consideraba negativa y se almacenaba en una carpeta aparte con el nombre ‘negative’.

Figura 40 - Ejemplo Resultado Data Augmentation con Bounding Boxes Dibujados

Este programa presentaba algunos fallos que hubo que corregir. El primer error se producía cuando se hacia el ajuste de los bounding boxes. Si el desplazamiento, rotación o inclinación era muy pronunciado o coincidía en los márgenes de la imagen, las coordenadas de los bounding boxes caían fuera de la imagen y se generaban

recuadros erróneos. Para solucionar esto hubo que tener en cuenta las dimensiones de la foto, y si alguna de las nuevas coordenadas era mayor que la dimensión de la foto se descartaba dicho bounding box por no encontrarse dentro de la imagen. Otro de los fallos presentados era perdida de información en la imagen. Esto era un comportamiento que se había producido en alguna ejecución que provocaba que la imagen estuviera incompleta. Por tanto, se generó un código aparte para verificar las imágenes, el cual, cargaba las imágenes y compraba que estuvieran correctamente, en caso contrario, indicaba que imagen era la incorrecta.

Posteriormente a realizar este proyecto, se descubrió una nueva herramienta de aumentación de datos llamada 'imgaug' **[28]**. Esta herramienta se utilizaba como una librería de Python, y constaba de varias funcionalidades para a través de código modificar las imágenes aplicando multitud de transformaciones disponibles. A diferencia de otras, esta herramienta permitía la aumentación de los bounding boxes junto con la imagen (ver **Figura 41**).

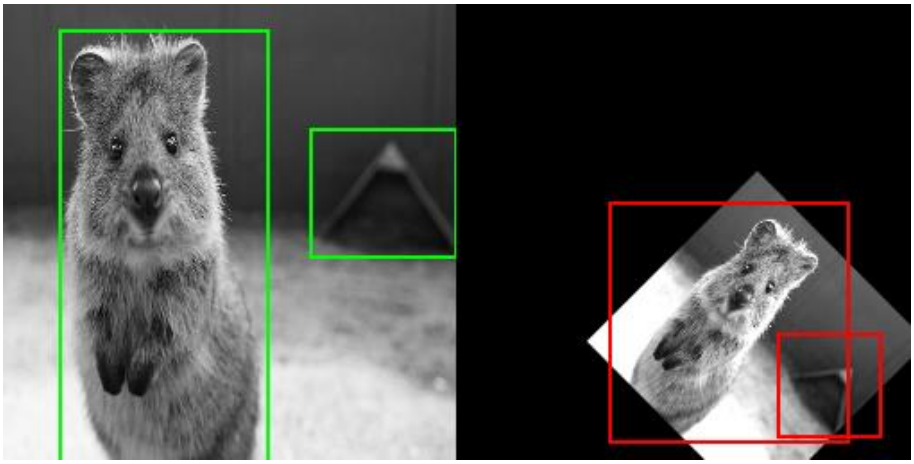

Figura 41 - Imgaug: Ejemplo Data Augmentation, Original (Izquierda) y Aumentación (Derecha)

Además, incluye el control de los límites de la imagen con los bounding boxes. Existe la opción de eliminar las regiones que caen fuera de la imagen y una segunda opción que rectifica la región al tamaño adecuado (ver **Figura 42**).

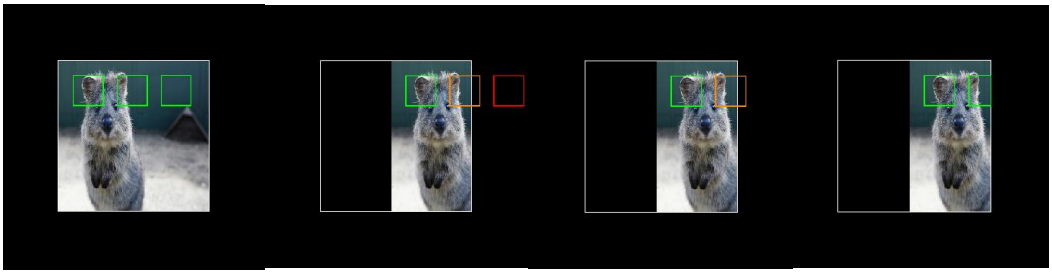

Figura 42 - Imgaug: Ejemplo Eliminación y Rectificado de Bounding Boxes

5.2 Datasets

En este proyecto finalmente se crearon múltiples datasets con el objetivo de mejorar los resultados proporcionados por el sistema de detección de objetos. Cada dataset tiene sus características y modificaciones en comparación con los demás.

5.2.1 HandGesture (HG)

Este dataset fue el inicial, el primer conjunto de datos que se probó para el reconocimiento gestual. La primera prueba se generó con un dataset de gestos con las manos descargado de internet. En este dataset había 27 clases diferentes de gestos, pero para nuestro sistema utilizaríamos 3 como etiquetas de clase (ver **Figura 43**). Con el fin de ir mejorando los resultados se llegaron a crear 6 versiones y 7 entrenamientos diferentes de este dataset.

Figura 43 - HandGesture: READY (Izquierda), GO (Centro) y STORE (Derecha)

5.2.2 HandGesturePropias (HGP)

El segundo dataset creado era una ampliación del anterior. Este conjunto de datos incluye fotos propias de personas haciendo gestos. Este dataset incluye por primera vez las imágenes de gestos con guante, debido a que los usuarios finales llevarán guantes. Las imágenes contienen bastante variabilidad, se realizan en distintas zonas, condiciones diferentes y distancia al objeto cambiante con el objetivo de fomentar el aprendizaje del sistema. Este dataset es el que más versiones y entrenamientos de pruebas conllevó, llegando a las 12 versiones y 23 entrenamientos. Un ejemplo de las nuevas imágenes incluidas en el dataset se puede observar en la **Figura 44**.

Figura 44 - HandGesturePropias: READY (Izquierda), GO (Centro) y STORE (Derecha)

5.2.3 NewHGP (HGP, Mano Izquierda y Derecha)

Este dataset es una copia del anterior con una diferencia importante. En este caso, tanto la mano derecha como la mano izquierda se consideran gestos positivos mientras el gesto realizado sea correcto. Los anteriores conjuntos de datos solo reconocían la mano derecha haciendo el gesto como gesto positivo. Este dataset tuvo 2 versiones y 2 entrenamientos.

Debido a que los resultados no mejoraron se optó por continuar con la mejora de los datasets que no incluían la mano izquierda como objeto positivo.

5.2.4 DataAGV

Las pruebas finales realizadas de los anteriores datasets a priori arrojaban buenos resultados, por lo que se instaló una primera versión en el emplazamiento final del cliente. Los resultados obtenidos fueron malos debido a que las condiciones y el entorno eran muy dispares a lo aprendido con los anteriores datasets. Esto provocaba muchos falsos positivos, además de no obtener un buen resultado de detección de gestos.

El objetivo de este dataset era paliar los errores producidos por los anteriores conjuntos de datos. Para ello, se creó este dataset desde cero con imágenes de las instalaciones finales exclusivamente (ver **Figura 45**). Este dataset llevó 5 versiones y 6 entrenamientos.

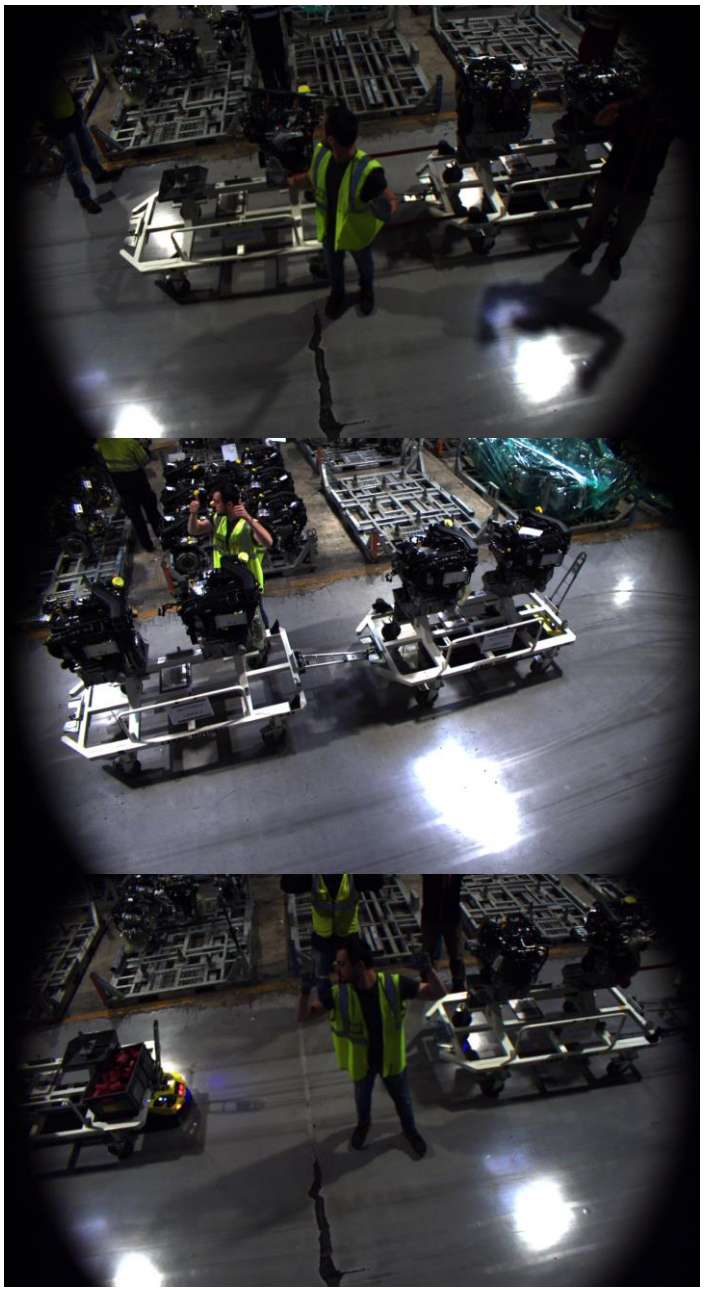

Figura 45 - DataAGV: READY (Arriba), GO (Centro) y STORE (Abajo)

5.2.5 BackAGV

El anterior dataset con la inclusión de las imágenes en la planta de producción, las detecciones mejoraron y se redujeron los falsos positivos. A pesar de incluir las fotos en las instalaciones finales, los resultados seguían sin ser adecuados. Las detecciones seguían detectando gestos en zonas donde no los había, como en los AGV en movimiento, arrugas en las camisetas o en las caras. Para intentar mejorar los resultados obtenidos se generó un nuevo dataset incluyendo una nueva etiqueta para identificar los elementos del fondo y aumentar la capacidad de discernir del sistema. De este conjunto de datos se produjo una única versión y entrenamiento.

5.2.6 NoisyAGV

La inclusión de ruido en las imágenes se ha probado que en otros casos ha mejorado los resultados de los sistemas de redes neuronales, dotando de la capacidad de detectar clases en imágenes incompletas o en mal estado. Por ello, a este conjunto de datos se le agregaron las imágenes con ruido, en la **Figura 46** se puede ver un ejemplo. Para este dataset se hicieron 2 entrenamientos de 2 versiones, una con la nueva clase generada en el anterior dataset y otra versión sin ella. Los resultados obtenidos finalmente fueron similares a los conseguidos con el dataset anterior BackAGV.

Figura 46 - NoisyAGV: Ejemplo Imagen con Ruido Agregado

5.2.7 DualBackAGV

Las especificaciones del cliente y un artilugio para leer códigos que los operarios llevaban en las manos forzaron la introducción de las dos manos como gesto positivo. Por lo que, este dataset es similar al dataset BackAGV con la diferencia que hubo que re-etiquetar el conjunto de datos con la mano izquierda como gesto positivo. Para este dataset final se crearon 3 versiones y entrenamientos diferentes.

6. Resultados

Este capítulo incluye los resultados obtenidos durante el desarrollo del sistema de reconocimiento gestual a lo largo del proyecto. Las pruebas se han realizado para multitud de datasets diferentes con ciertas diferencias o cambios entre ellos con la finalidad de mejorar el rendimiento final de la aplicación. Los resultados reflejados pertenecen a la aplicación Python que incluye el sistema de detección de objetos con el Faster R-CNN **[11]**.

La primera columna indica el nombre del dataset entrenado que corresponde a **'Nombre Dataset'**.

Debido a los problemas del CNTK para introducir otro tipo de modelos base en el detector de objetos, la mayoría de los resultados del entrenamiento se ejecutan sobre el modelo base VGG16 **[14]**, y en menor medida, el modelo base AlexNet **[31]**. El modelo base se indica en la tabla de resultados por la columna **'Modelo Base'**.

En cuanto a los datos del dataset, la columna **'Núm. Clases'** indica cuantas clases de objetos diferentes se encuentran en el dataset. Mientras que la columna de nombre **'Tamaño'** proporciona el tamaño del dataset. La columna **'Data Augmentation'** proporciona la cantidad de data augmentation o número de imágenes con transformaciones generadas a partir de las demás imágenes del dataset.

Los datos del entrenamiento se dividen en 4 columnas. La primera columna **'Tamaño Train'** nos indica el número de imágenes utilizadas para entrenamiento. La segunda columna **'# Epochs'** corresponde al número de epochs realizados en el entrenamiento. La tercera **'Pre-Entrenamiento'** nos indica si se ha entrenado toda la red o solo las ultimas capas, esta columna puede contener el valor SI o NO respectivamente. Por último, la columna final **'Duración min'** nos proporciona la duración del entrenamiento en minutos.

Los datos de la validación también están incluidos en la tabla. La columna **'Tamaño Test'** corresponde al número de imágenes utilizadas en la validación del entrenamiento. El Faster R-CNN del CNTK **[23]** la bondad del sistema nos la devuelve con un porcentaje de mAP. Este valor se identifica en la tabla mediante la columna de nombre **'% mAP'**. El problema es que este valor devuelto no se corresponde bien con los datos de validación. En el caso de los datasets que incluyen elementos del fondo para obtener mejor discriminación de los gestos con el fondo, al no detectar todos los elementos marcados del fondo esto perjudica gravemente al valor del mAP. Por tanto, para este proyecto la precisión del sistema para detectar gestos se ha calculado manualmente. La precisión del sistema se obtiene mediante la visualización de las imágenes de validación del entrenamiento una a una, y comprobando que los gestos etiquetados con anterioridad corresponden con las detecciones realizadas por el sistema. Este cálculo es un procedimiento costoso si el número de datos de validación es muy alto, pero nos proporciona el porcentaje de acierto obtenido por el sistema en el reconocimiento gestual. En la tabla el porcentaje de acierto corresponde a la columna **'% Acierto'**.

	Red Neuronal	DataSet			Train				Test		
Nombre Dataset	Modelo Base	Núm. Clases	Tamaño	Data Augmentation	Tamaño Train	# Epochs	Pre-Train	Duración min.	Tamaño Test	% mAP	% Acierto
HandGesture (HG)	VGG16	3	2425	2329	2200	20	NO	135	225	10,78%	98,66%
HandGesture (HG)	VGG16	3	1683	1587	1487	20	NO	120	196	12,65%	98,47%
HandGesture (HG)	VGG16	3	9263	9167	9263	20	NO	630	225	???	98,66%
HandGesturePropias (HGP)	VGG16	4	22575	3321	21475	75	SI	7500	1100	9,29%	98,73%
HandGesturePropias (HGP)	VGG16	4	25431	26954	24331	75	SI	7800	1100	8,48%	99,00%
NewHGP	VGG16	4	8440	4615	7790	75	SI	2160	650	10,00%	94,92%
NewHGP	VGG16	4	21519	23042	20419	75	SI	7200	1100	8,01%	97,55%
DataAGV	VGG16	4	22570	20646	21570	75	SI	7800	1000	10,95%	93,50%
DataAGV	VGG16	4	19574	16875	18574	75	SI	6600	1000	10,14%	94,20%
DataAGV	VGG16	4	19574	16875	18574	75	SI	6600	1000	10,24%	94,30%
BackAGV	VGG16	5	19658	16875	18658	75	SI	7200	1000	26,71%	96,00%
NoisyAGV	VGG16	4	23067	20368	22067	75	SI	8700	1000	24,72%	95,90%
NoisyAGV	VGG16	5	23151	20368	22151	75	SI	8700	1000	20,01%	94,50%
DualBackAGV	VGG16	5	32923	30620	31923	75	SI	15000	1000	23,14%	96,40%
DualBackAGV	VGG16	5	35040	16496	34040	75	SI	14400	1000	24,49%	97,70%

Tabla 27 - Tabla de Mejores Resultados

La **Tabla 27** previa solo recopila algunas de las pruebas realizadas a lo largo del proyecto. Para poder observar el abanico completo de pruebas realizadas se debería acceder al **Anexo 10.2**, donde se ubica la tabla completa de resultados obtenidos para este sistema.

En los primeros datasets (**HG**) ya se consiguen valores de acierto altos, rondando el **98%** de acierto para varias de las versiones. Las versiones de este dataset con un 100% de acierto que se encuentran en el **Anexo 10.2** no son muy relevantes debido a su bajo volumen de imágenes de validación o la inclusión de imágenes de entrenamiento en la validación a modo de prueba. El problema es que a la hora de hacer detecciones en imágenes en entornos que no tiene entrenados o a distancias más lejanas los resultados no son buenos.

Para intentar resolver este problema, se creó el dataset HandGesturePropias (HGP). Este dataset al incluir imágenes de diferentes entornos, y gestos a distancia y escala variables, consiguió mejorar la detección bajo diferentes circunstancias. Además, de una nueva etiqueta 'NEGATIVE' correspondiente a cualquier gesto que no fuera los otros 3 etiquetados, con el fin de diferenciar mejor entre gestos. La última versión de este conjunto de datos (**HGP**), que incluía el número de imágenes más elevado hasta el momento, consiguió obtener un porcentaje de acierto de las imágenes de validación de un **99%**, siendo el mejor resultado obtenido. En este caso, al instalar este dataset en las instalaciones del cliente producía muchos falsos positivos. Las condiciones del entorno de las líneas de producción contienen mucho ruido y provocaba múltiples falsos positivos en las detecciones.

A modo de prueba, se creó un dataset con las mismas características que el HGP, pero en este caso etiquetando las dos manos como gesto positivo (**NewHGP**). El porcentaje de acierto para este dataset alcanzó un notable resultado del **97.55%** de acierto. Pero debido a que no conseguía superar el 99% de acierto del anterior conjunto de datos, se decidió no utilizarlo.

Con el objetivo de paliar estos falsos positivos producidos por el entrenamiento de los anteriores datasets, se generó uno nuevo (DataAGV) con las imágenes de las instalaciones finales. El dotar al sistema de imágenes en el entorno final fue un acierto, debido a que el porcentaje de falsos positivos se redujo considerablemente. La detección en este dataset era más complicada por las variaciones de las condiciones en las imágenes. A pesar de esto, los resultados de acierto obtenidos para la última versión de este dataset consiguieron llegar al **94,30%**, siendo un porcentaje alto de acierto, con el añadido de la reducción de detección de falsos positivos. Los falsos positivos se habían reducido, pero aun así seguían apareciendo en las imágenes.

A los dos siguientes datasets (BackAGV y NoisyAGV) se les añadió dos diferencias. El primero se hizo con el objetivo de que el sistema fuera capaz de discriminar mejor los gestos del fondo. Para ello, se etiquetaron las diferentes zonas de la imagen que detectaba falsos positivos como una nueva clase 'BACKGROUND'. Estas regiones solían ser los carros, los motores o las caras de la gente que aparecía en la imagen. El segundo dataset introduce como diferencia imágenes con ruido gaussiano, sal y pimienta, y un ligero desenfoque en la fase de data augmentation. Esto, en otros casos, ha ayudado a que las redes neuronales convolucionales

ganasen conocimiento para detectar incluso cuando el objeto en si no es visible en su totalidad. Las dos pruebas incrementaron los resultados obtenidos. El **BackAGV** consiguió un porcentaje de acierto de **96,00%** mientras que el **NoisyAGV** un **95,90%**. Por lo que finalmente se introdujo el sistema entrenado mediante el BackAGV1 como detector de gestos en las instalaciones del cliente.

El dataset **NoisyAGV** su segunda versión era una combinación de ambos conjuntos de datos, pero los resultados obtenidos (**94,50%**) no mejoraban los resultados de ninguno de ellos, por lo que se descartó esta prueba.

Finalmente, el ultimo conjunto de datasets generados pertenece a los DualBackAGV. Este dataset se creó debido a que los operarios que trabajan en las líneas de producción (los usuarios finales de este sistema), añadieron un lector de códigos en los dedos, el cual, perjudicaba las detecciones de los gestos. Para ello, se introdujo la opción de realizar el gesto con cualquiera de las dos manos, izquierda o derecha. Para ello, se partió de las mismas pautas realizadas para el dataset instalado (BackAGV) y se re-etiquetaron las imágenes para incluir la mano izquierda como gesto positivo. Esto permitió a los usuarios del sistema realizar el gesto con cualquiera de las dos manos, pudiendo efectuar los gestos con la mano que no incluyese el lector de códigos. A pesar de estos cambios, en cuanto al porcentaje de acierto para el conjunto de imágenes de validación, los resultados seguían siendo buenos con un **97,70%** para la versión más reciente del **DualBackAGV**.

Además, a lo largo de la **Tabla 27** se puede observar, cuantos más datos de entrenamiento contienen los datasets, tiende a obtener mejores resultados de acierto, independientemente de ser imágenes reales o generadas mediante data augmentation.

7. Aplicación

La aplicación final cuenta con muchos elementos de desarrollo diferentes, desde la aplicación desarrollada en Python con la estructura TCP integrada para la comunicación con la aplicación de C#, hasta las diferentes herramientas y funcionalidades integradas a lo largo del proyecto. En este capítulo se detallará el desarrollo correspondiente a la aplicación final.

7.1 Aplicación Python

La parte que integra todo el desarrollo hecho alrededor del detector de objetos del Faster R-CNN, se incluye en la aplicación Python. Esta aplicación mediante un sistema de comunicación TCP se comunica con la aplicación C# que realiza las gestiones de conexión con el PLC y las visualizaciones de las detecciones.

7.1.1 Comunicación TCP

La aplicación Pyhton es la que contiene el sistema completo de reconocimiento gestual, pero para que el resultado del procesamiento de las imágenes tenga efecto en el proceso de transporte de los carros, la aplicación de C# tiene que obtener esos resultados. Esto es, debido a que la aplicación de C# es la que puede indicar acciones a los sistemas AGV, ya que, es el sistema que tiene conexión con el PLC. Por tanto, dependiendo de los valores de los resultados obtenidos realizará una acción u otra.

Después de realizar diferentes pruebas, finalmente, se diseñó un sistema de comunicación TCP para transmitir los resultados de la detección gestual a la aplicación C#. La estructura de la comunicación comienza desde el lado del servidor (en este caso la aplicación Python de reconocimiento gestual). Mediante una IP y un puerto crea un servidor de escucha, el cual, espera peticiones de conexión. Cuando un cliente (en este caso la aplicación C#) quiere enviar una imagen para obtener los resultados del reconocimiento gestual, hace una petición de conexión al servidor, y este le genera un socket único donde el cliente enviará las imágenes a procesar y el servidor le responderá con el resultado de las imágenes recibidas por ese socket (ver **Figura 47**).

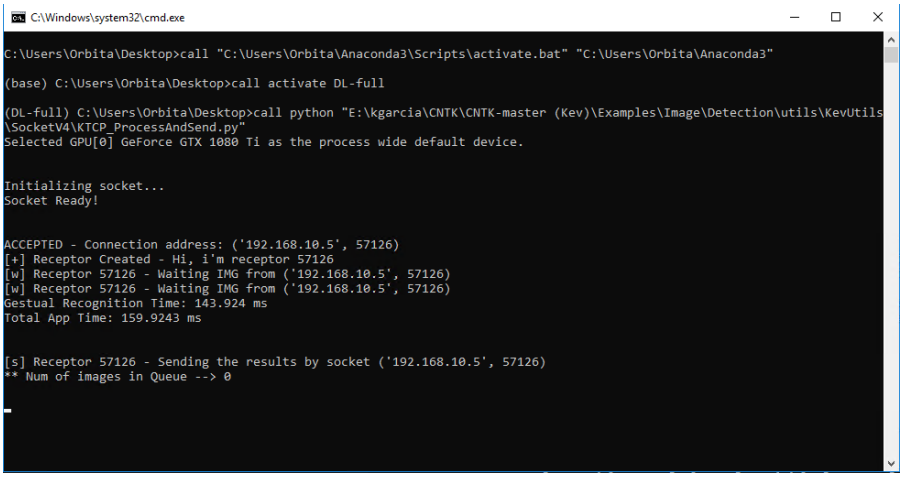

Figura 47 - Consola Socket TCP en Ejecución

El flujo es el siguiente, el cliente hace una petición al servidor, el servidor crea un socket de comunicación, mediante el socket de comunicación el cliente le envía las imágenes a procesar, el servidor las procesa, y envía los resultados. El ejemplo de la estructura se puede ver en la **Figura 48**.

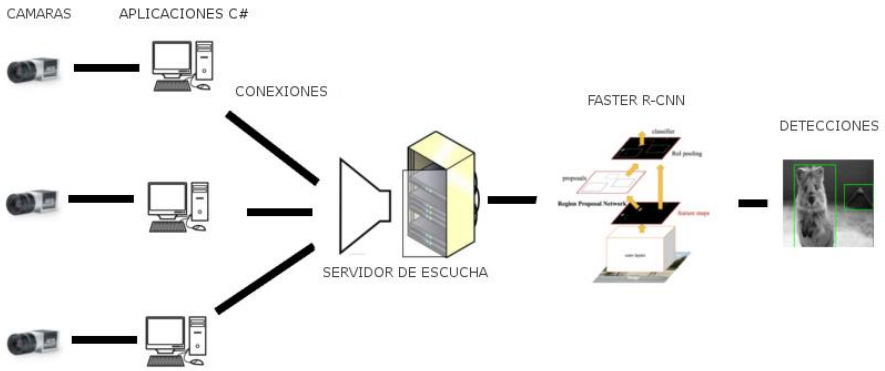

Figura 48 – Estructura Simplificada TCP

El servidor admite múltiples conexiones simultaneas, a cada conexión le genera un socket único, las imágenes se van encolando desde esos sockets por orden de llegada con un id correspondiente al socket al que pertenece y se va procesando la cola de imágenes para devolver los resultados codificados en un archivo JSON al mismo cliente que envío la imagen. El servidor es un hilo paralelo que siempre está en ejecución escuchando peticiones de nuevas conexiones. Para cada conexión se crea otro hilo, el cual, espera imágenes de la conexión establecida por el cliente, y cuando obtiene una imagen la introduce en la cola de procesamiento.

Por último, el hilo principal contiene la cola de imágenes, donde se van encolando las imágenes enviadas por las diferentes conexiones. La cola está siempre a la espera de contener algún elemento y cuando le llega alguna imagen, se obtiene esa imagen, se procesa mediante el Faster R-CNN entrenado para reconocimiento gestual y se obtienen los resultados de las detecciones. Estos resultados se codifican y se envían a través del socket con el id correspondiente a la imagen procesada, es decir, el cliente que envió la imagen a la cola.

Este método, permite a la aplicación C# descodificar los resultados del JSON obtenido, y realizar las acciones correspondientes a los valores obtenidos. La cantidad máxima de la cola es de 10, en caso de que entren nuevas imágenes sin que la cola se haya vaciado se irán eliminando las imágenes más antiguas sustituidas por las recién encoladas. Esta longitud de la cola es un parámetro que podemos modificar libremente.

7.1.2 Aplicación Final

La aplicación final es una combinación de la aplicación Python y la aplicación C# enlazadas mediante el sistema de comunicación TCP detallado en el apartado anterior.

La aplicación de Python que ejecuta el Faster R-CNN del CNTK entrenado para la detección de gestos, se ejecuta en segundo plano mediante una consola que inicia la comunicación TCP para proceder con el proceso de esperar peticiones de conexión para procesar imágenes mediante el sistema de redes neuronales convolucionales (ver **Figura 49**).

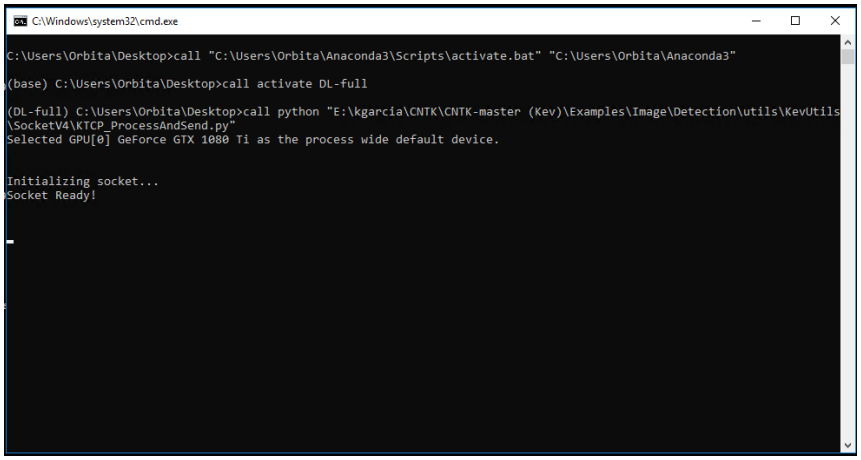

Figura 49 - Consola Gestual TCP

Una vez está en ejecución la aplicación Python, se inician las aplicaciones de C#. Hay una aplicación C# asociada por cada línea de producción. En este caso tenemos

3 cámaras correspondientes a las líneas de producción 1, 2 y 3. Cada aplicación C# muestra las imágenes de la cámara que tiene conectada y los resultados obtenidos del sistema de reconocimiento gestual. Además, cada aplicación gestiona el AGV de su línea de producción indicándole las acciones correspondientes a llevar a cabo. Un ejemplo de las 3 aplicaciones de C# se puede observar en la **Figura 50**.

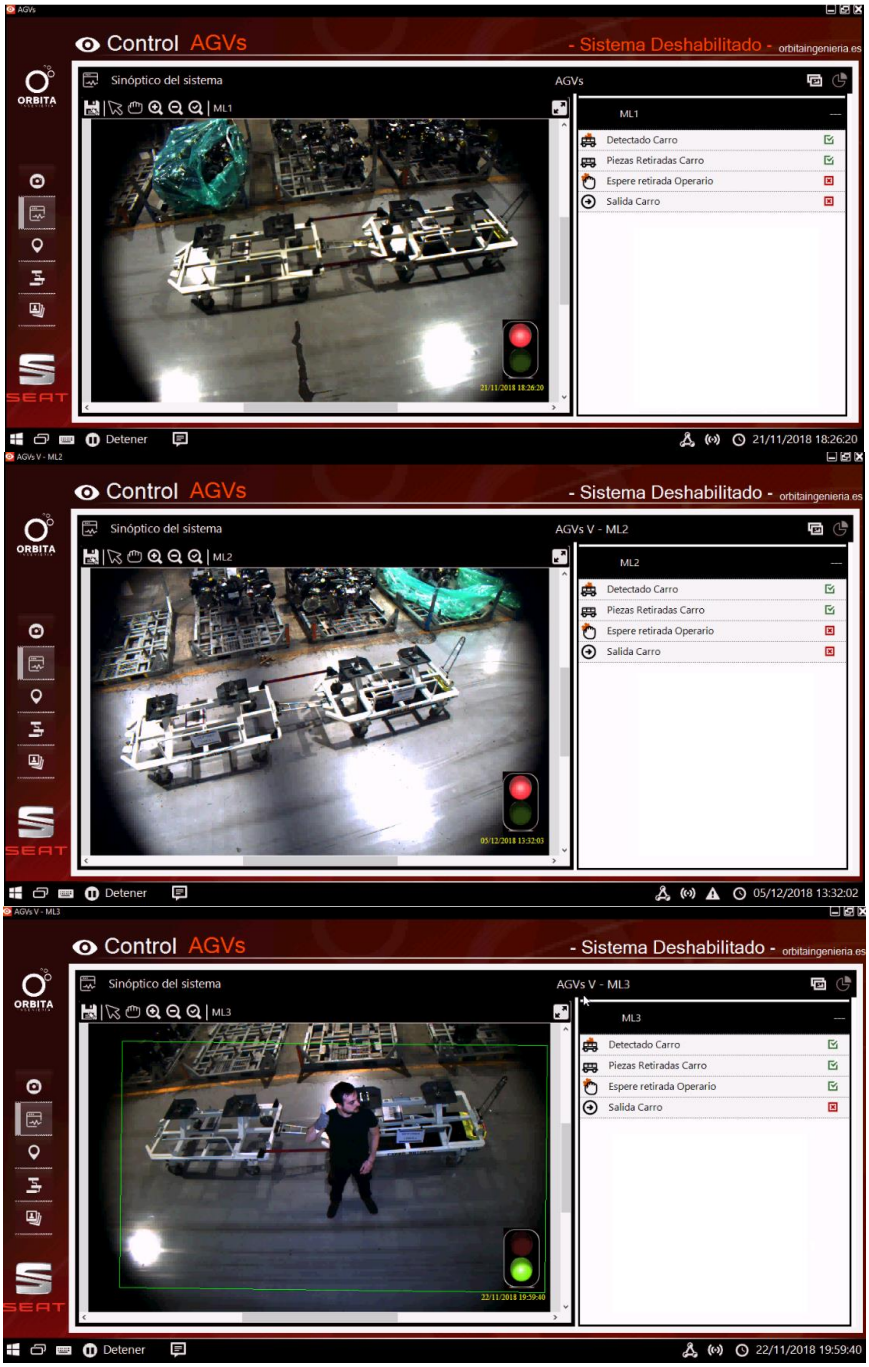

Figura 50 - Aplicaciones C#: Líneas de Producción ML1 (Arriba), ML2 (Centro) y ML3 (Abajo)

Cada aplicación C# cuando se reciben los resultados del reconocimiento gestual, muestra en pantalla el bounding box del gesto detectado con su clase. Si no existe

gesto en la imagen no se muestra detección alguna. Las detecciones en las aplicaciones se muestran como en la **Figura 51**.

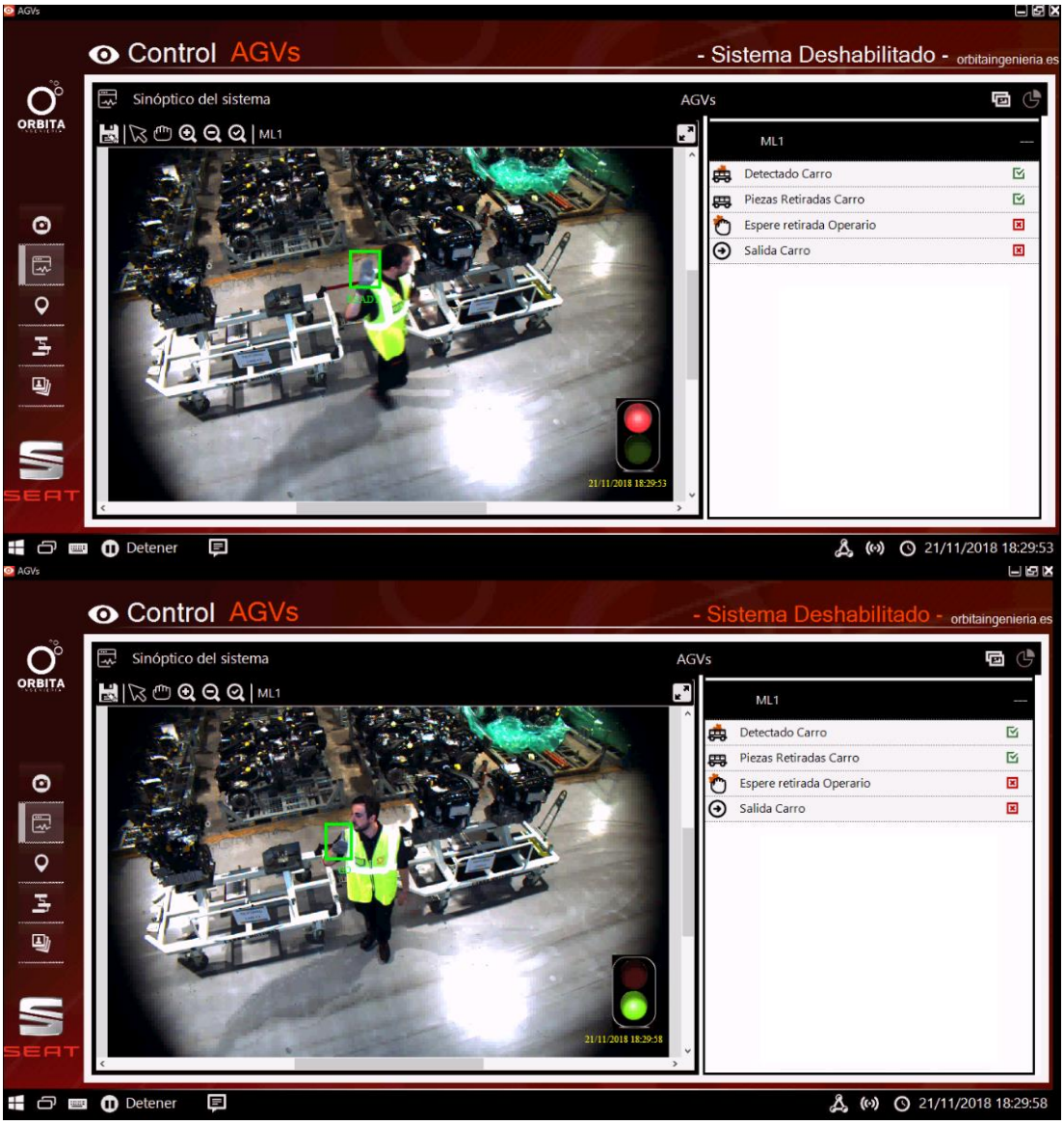

Figura 51 - Aplicación C#: Detección de Gestos READY (Arriba) y GO (Abajo)

Estas aplicaciones actualmente están configuradas para gestionar los gestos de READY y GO. Por el momento, el gesto STORE está en desuso a falta de ser integrado en caso de necesidad de añadir nueva funcionalidad futura al sistema.

La aplicación en el momento que tenga un carro en la imagen y esté lleno, entra en el estado de búsqueda de gestos. En este estado la secuencia para indicar al AGV la salida a la siguiente estación consta de un primer gesto READY, seguido de un GO. En caso de que no se detecte el gesto GO pasado un cierto tiempo (parámetro modificable fijado a 10 segundos), se vuelve al estado anterior y se vuelve a pasar a buscar el gesto READY. Además, si en un intervalo de tiempo sigue sin detectar el

gesto READY, se sale automáticamente del estado de búsqueda de gestos. Hasta no detectar que el carro está dentro de la imagen y el carro está lleno por completo, no se vuelve al estado de detección de gestos. Este reinicio del estado del sistema se realiza para controlar que la aplicación no se quede en ese estado si no está en funcionamiento la línea de producción y esté continuamente tomando imágenes. Cabe destacar que cada gesto se ha de detectar dos veces en las últimas 10 imágenes para ser considerado como gesto valido para el sistema en el orden apropiado. Esto implica que, para que el sistema de la salida al AGV, se tienen que detectar en 2 imágenes el gesto READY para pasar a buscar el gesto GO y después en otras 2 imágenes el gesto GO, en las últimas 10 imágenes obtenidas.

Detrás de las cámaras que obtienen las imágenes, al realizar los gestos se encuentra una serie de luces que indican si el gesto ha sido detectado. La luz roja indica que la aplicación ha entrado en el estado de detección de gestos. Esta luz se cambia al color naranja cuando se detecta el gesto READY. Y finalmente, cuando se detecta el gesto GO se vuelve verde (ver **Figura 52**).

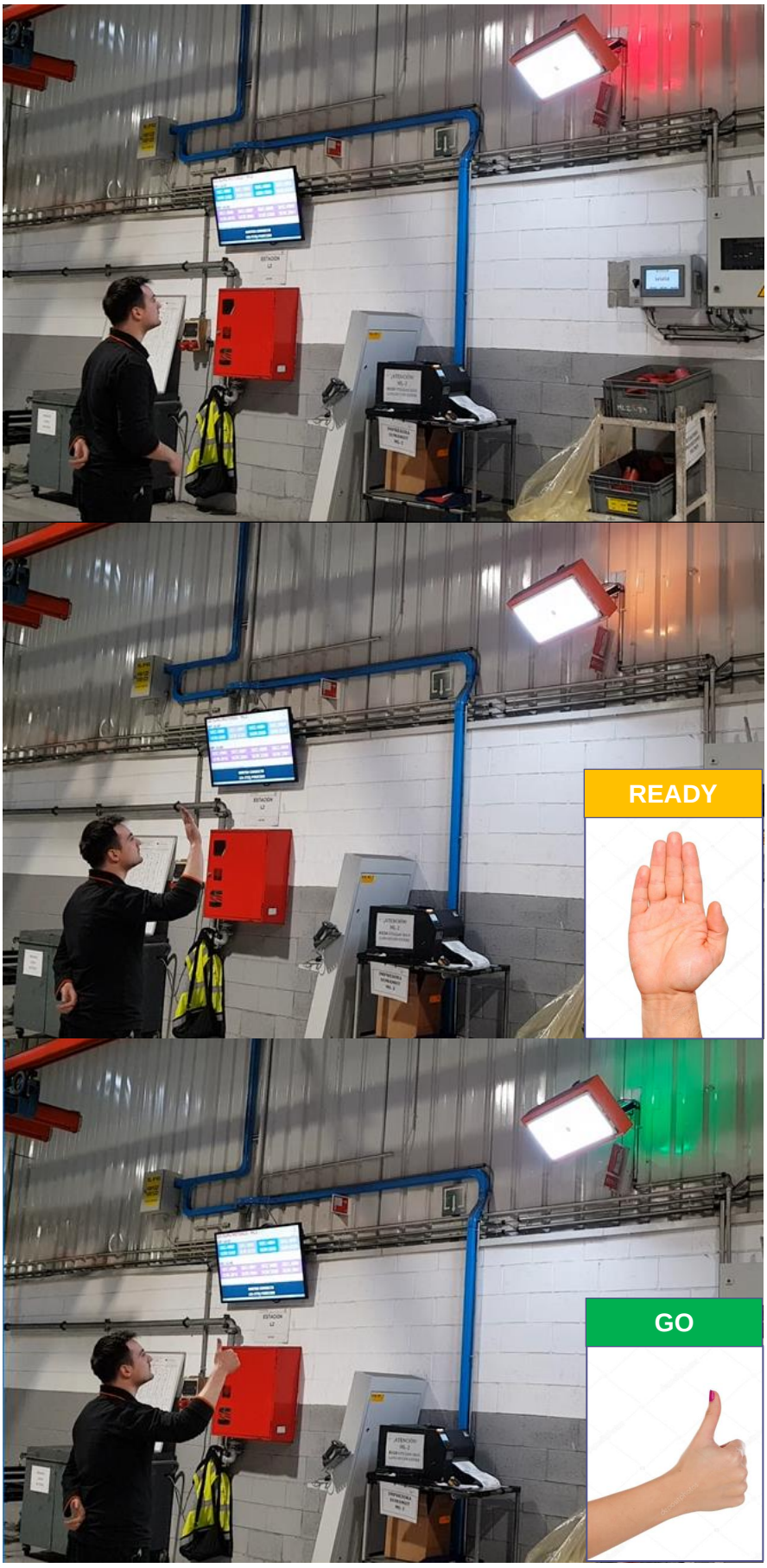

Figura 52 - Cambio de Luces Pared: Secuencia de Gestos

Una vez realizada la secuencia, la propia aplicación envía una señal al PLC, que comunica con el AGV de la línea y este continua la marcha.

7.2 Herramientas Descartadas

7.2.1 Facenet

A modo de seguridad extra, se integró un reconocedor de caras ya implementado **[29]**. El nombre de este sistema era Facenet. Estaba constituido por 3 redes pequeñas que trabajaban en conjunto, para reconocer las caras frontalmente como de perfil y a diferentes escalas y distancias (ver **Figura 53**).

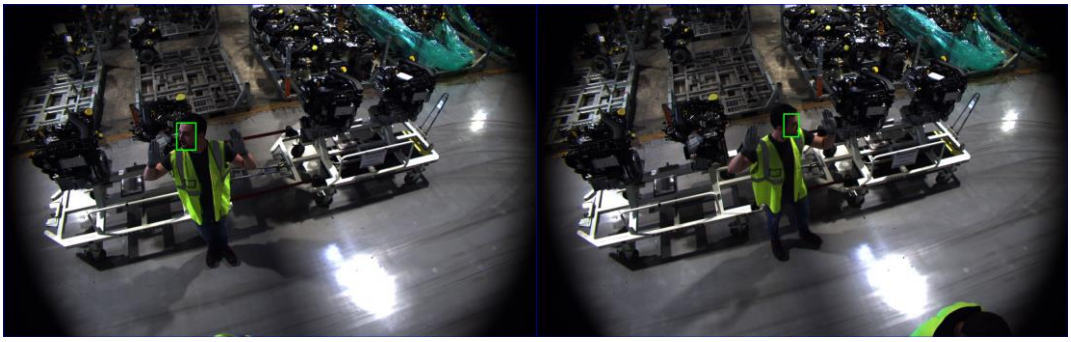

Figura 53 - Ejemplo Resultados Facenet

Al sistema de reconocimiento gestual se le hicieron varias modificaciones para integrar este método. Primero, la entrada de la red ya no sería la imagen completa, con el objetivo de reducir fondo no deseable se creaba un recorte del área de la mano respecto del rostro detectado y este recorte final es el que se le pasaría al sistema de redes neuronales convolucionales.

El método del Facenet funcionaba bien por si solo era capaz de reconocer las caras con un buen resultado. El problema es que al proporcionar una imagen de menor escala el CNTK por defecto reescala la imagen a un tamaño de 850x850. Esto provoca que la imagen se vea borrosa, pixelada y con pérdida de nitidez (ver **Figura 54**). Por ello, el sistema no es capaz de realizar las detecciones correctamente.

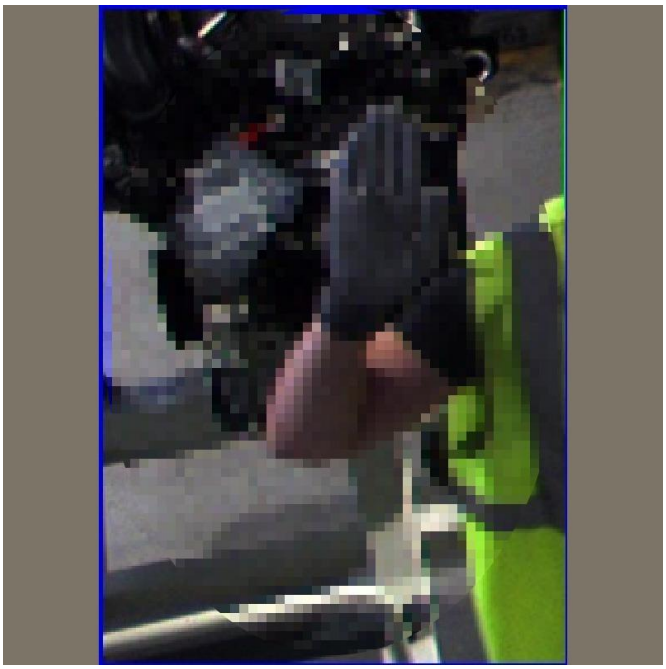

Figura 54 - Recorte Facenet Pixelado

Además, este algoritmo estaba diseñado en tensorflow, y el sistema de reconocimiento gestual estaba desarrollado en CNTK. Esto producía un tiempo de detección elevado, ya que, cada entorno reserva parte de la tarjeta gráfica independientemente. Se modificaron los parámetros de asignación de la memoria de la GPU de los dos entornos, pero los resultados no fueron buenos igualmente (~500ms la ejecución conjunta), por lo que finalmente se descartó esta idea.

7.2.2 RabbitMQ

Al principio se integró el software de mensajería RabbitMQ **[30]**. Este software, además de otras funcionalidades, proporciona una plataforma de gestión de colas. El RabbitMQ genera una clave, indicando la clave a través de código junto con la conexión, podremos encolar las imágenes o recogerlas de la misma forma. Proporciona facilidades para obtener conexiones entre diferentes plataformas.

En nuestro caso, las tres líneas de producción proporcionan imágenes que iremos acumulando en una cola. Las imágenes de está cola se irán procesando por el sistema de reconocimiento gestual y se enviarán los resultados obtenidos a la aplicación C#.

Este software la gestión de la cola la realizaba adecuadamente y proporcionaba visualización del contenido de las colas, conexiones y otros parámetros. El problema es que el tiempo que empleaba para la transferencia de los datos no era suficientemente rápida, y el sistema de reconocimiento gestual no iba a la velocidad necesaria. Por lo que finalmente descartamos este software para la gestión de la cola de imágenes.

8. Conclusiones

El sistema de reconocimiento gestual se encuentra en funcionamiento actualmente en el cliente. Los resultados no son malos, sigue habiendo algún falso positivo, pero se han reducido considerablemente y las detecciones de los gestos se efectúan adecuadamente. La baja tasa de falsos positivos no suelen ser un problema debido a los demás procedimientos de seguridad añadidos. Utilizar solo las mejores detecciones obtenidas de cada clase en la imagen, además de la seguridad de tener que detectar cada gesto en 2 imágenes de entre las ultimas 10 obtenidas en el orden adecuado, evita que produzcan problemas los falsos positivos.

Haciendo análisis del proyecto, debido a las especificaciones asignadas, tener que utilizar el CNTK y el detector de objetos Faster R-CNN no ha sido la mejor opción disponible para este trabajo. La detección de gestos requiere una velocidad alta de detección, el Faster R-CNN no destaca precisamente por ser rápido en realizar las detecciones, está alrededor de unos 150ms en nuestro caso. Esto supone un retraso desde que el usuario realiza el gesto hasta que se detecta dicho gesto en las imágenes. Este retraso no es muy adecuado ya que a veces da lugar a confusión si el sistema está detectando bien el gesto o simplemente es cuestión de la demora producida. Además, el CNTK no tiene demasiada documentación detrás en la que puedas respaldarte a la hora de intentar corregir errores o realizar las configuraciones del sistema adecuadamente. Añadido a esto, otro problema que contiene está relacionado a los modelos base. No tiene soporte para modelos base basados en ResNet y las configuraciones para ejecutar otros modelos base diferentes a AlexNet y VGG16 no es sencilla y no acaba de funcionar. Para proyectos de menor envergadura o que no dependan demasiado de la velocidad del sistema, es una buena opción gracias a su facilidad de utilización y configuración.

También hay que destacar que las imágenes proporcionadas por las cámaras no eran las más adecuadas para este sistema, debido a las características de las imágenes. El posicionamiento de la cámara, la iluminación y el contraste de las instalaciones, no eran los más adecuados. Unas mejores condiciones de las imágenes podrían haber ahorrado varios problemas. El tener que lidiar con estos problemas en las imágenes confirma aún más la opción de utilizar un sistema basado en redes neuronales convolucionales. Los métodos de visión artificial tradicional acusan mucho más estas variaciones de las condiciones en las imágenes.

Como trabajo futuro, con el objetivo de mejorar el sistema y resolver los problemas comentados anteriormente, se procedería a cambiar el CNTK y el Faster R-CNN por un sistema que incluyera el YOLO3 sobre el entorno de TensorFlow con Keras. La velocidad del YOLO3 es muy superior a la del Faster R-CNN y los resultados de detección también son buenos, por lo que nos proporcionaría mejor rendimiento de velocidad en el sistema. Este cambio también nos concede mucho soporte y documentación para Keras y TensorFlow, son dos de las herramientas más utilizadas actualmente.

9. Bibliografía

[1] J. M. Vicente. Sistema de Reconocimiento de Comandos por Voz Basado en Redes de Neuronas LSTM. Junio 2018. 13

[2] D. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 2004. 19

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In CVPR, 2005. 19

[4] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013. 19, 20, 25, 43

[5] V. Vapnik. The Nature of Statistical Learning Theory, Springer, 1995. 19

[6] Cortes C. and Vapnik V. Support vector networks. Machine Learning 20: 273-297, 1995. 19

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. IEEE Computer Vision and Pattern Recognition (CVPR), 2014. 20, 20

[8] mAP (mean Average Precision) for Object Detection. Medium. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173 20

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) Challenge, IJCV. 2010. 20, 21, 24, 27, 29, 31, 37, 43

[10] R. Girshick. Fast R-CNN. IEEE International Conference on Computer Vision (ICCV). 2015. 20, 22, 25, 28, 43

[11] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Neural Information Processing Systems (NIPS). 2015. 22, 24, 25, 32, 43, 77

[12] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2015. 22

[13] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft COCO: Common Objects in Context. European Conference on Computer Vision (ECCV). 2014. 24, 31, 36, 40

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. International Conference on Learning Representations (ICLR). 2015. 25, 27, 57, 77

[15] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. European Conference on Computer Vision (ECCV). 2014. 25

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640. 2015. 25, 27, 30, 33

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842. 2014. 27

[18] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 6517–6525, IEEE. 2017. 29, 38

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.- Y. Fu, and A. C. Berg. SSD: Single Shot Multibox Detector. European conference on computer vision, pages 21–37. Springer. 2016. 29, 32

[20] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv. 2018. 38

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778. 2016. 39

[22] The Microsoft Cognitive Toolkit. https://docs.microsoft.com/en-us/cognitive-toolkit/ 43

[23] Object Detection using Faster R-CNN. Microsoft CNTK. https://docs.microsoft.com/en-us/cognitive-toolkit/object-detection-using-faster-r-cnn 43, 51, 57, 77

[24] Object detection using Fast R-CNN. Microsoft CNTK. https://docs.microsoft.com/en-us/cognitive-toolkit/Object-Detection-using-Fast-R-CNN 46

[25] CUDA.Wikipedia. https://es.wikipedia.org/wiki/CUDA 62

[26] Que son los Nvidia CUDA cores y cuál es su importancia. Profesional Review. https://www.profesionalreview.com/2018/10/09/que-son-nvidia-cuda-core/ 62

[27] Visual Object Tagging Tool: An electron app for building end to end Object Detection Models from Images and Videos. GitHub. https://github.com/microsoft/VoTT 70

[28] Image augmentation for machine learning experiments. GitHub. https://github.com/aleju/imgaug 72

[29] Face recognition using Tensorflow. GitHub. https://github.com/davidsandberg/facenet 88

[30] RabbitMQ https://www.rabbitmq.com/ 89

[31] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. NIPS, pp. 1106–1114. 2012. 57, 77

[32] NVIDIA.cuDNN https://developer.nvidia.com/cudnn 62

10. Anexos

10.1 Glosario

Neural Network: Una red neuronal, es un modelo computacional que pretende simular la estructura de una red neuronal biológica a través de neuronas artificiales conectadas. Los datos de entrada atraviesan la red neuronal efectuando diferentes operaciones y finalmente se obtienen unos valores de salida.

Convolutional Neural Network (CNN): Red neuronal convolucional, es un tipo de redes neuronales profundas mayormente utilizadas en la visión artificial o visión por computador.

Automatic Guided Vehicle (AGV): Se trata de un vehículo guiado y automatizado para el transporte de mercancías.

Data Augmentation: Termino correspondiente al proceso de aumentar el número de muestras mediante transformaciones y/o perturbaciones de la imagen.

Bounding Box: Recuadro que tiene como objetivo contener el objeto a buscar dentro de la imagen.

Recall: Es el cálculo del número de verdaderos positivos respecto de los propios verdaderos positivos más el número de falsos negativos obtenidos, proporciona un valor de cuantos verdaderos positivos se han detectado frente a todos los casos positivos existentes.

Precision: Hace referencia al número de verdaderos positivos respecto de los propios verdaderos positivos más el número de falsos positivos, por lo que te devuelve un valor de la precisión de los verdaderos positivos encontrados frente al total de los positivos detectados.

Average Precision (AP): El termino AP viene de average precision, precisión media, esto se obtiene del cálculo conjunto del recall y la precisión obtenidos de los resultados de validación del modelo entrenado.

Mean Average Precision (mAP): Precisión media de entre todas las clases a detectar, es la media de las precisiones medias de cada clase.

Region of Interest (RoI): Es la región de interés de una imagen, se representa mediante un recuadro delimitador que encuadra el elemento de interés de la imagen.

Max Pooling: Es un tipo de pooling (combinación de cosas), el cual, reduce el tamaño del mapa de características en proporción al tamaño indicado, quedándose con las características más representativas del mapa de características y consiguiendo así un mapa de menor tamaño con las características concentradas.

Average Pooling: Otro tipo de pooling, este en vez de quedarse con las características más significativas, combina las características con las medias de estas.

Intersection over Union (IoU): También conocido como Jaccard Index, es la métrica utilizada para calcular el solape entre dos recuadros delimitadores o bounding boxes. Es el resultado de dividir la intersección de las regiones entre la unión de estas.

Ground Truth: Se le denomina ground truth, al recuadro delimitador etiquetado, es decir, es el bounding box que se sabe que corresponde con un objeto debido a que ha sido marcado manualmente como objeto.

Tensor: Se conoce como tensor a una unidad de datos, que puede tener múltiples dimensiones.

Anchors Boxes: Las cajas de anclajes, son recuadros con diferentes relaciones de aspecto y escalas, y están centrados sobre unas coordenadas proporcionadas.

Inception: Hace referencia a los módulos de las capas de las redes neuronales convolucionales diseñadas por Google, que tienen la peculiaridad de poder ejecutarse en paralelo, para posteriormente concatenarse y obtener la salida.

Clustering: Algoritmo de agrupación para dividir datos en diferentes grupos, los cuales, están formados por elementos del mismo tipo.

Loss Function: Es la función de perdida en inglés, se hace referencia a ella como loss, y relaciona un evento con un número real y mide el coste asociado al evento.

Non-Maximum Suppression (NMS): Es el proceso de descartar múltiples detecciones o bounding boxes similares obtenidos mediante la detección de objetos. El resultado de una detección de objetos suele tener muchos bounding boxes que se solapan entre sí para el mismo objeto. La idea de este método es obtener los bounding boxes que recuadran al mismo objeto, quedarse con el de mejor puntuación (score) y descartar los demás.

Reinforcement Learning: Aprendizaje basado en recompensas, las redes neuronales ejecutan acciones y en base a esas acciones se les recompensa o castiga, y adquieren conocimiento en base a eso.

Graphics Processing Unit (GPU): La unidad de procesamiento gráfico, es el elemento dedicado al procesamiento de los gráficos u operaciones en coma flotante de un sistema.

Epoch: Época en castellano, en la fase de entrenamiento del modelo, se conoce como una iteración completa de aprendizaje de todo el dataset.

Batch: Se conoce como lote, conjunto de datos de tamaño finito.

Momentum: Es un factor que acelera la convergencia del algoritmo, pero tiene en cuenta la información de la iteración anterior para que los cambios no sean bruscos.

Learning Rate: Tasa de aprendizaje o velocidad de aprendizaje. Cuanto más alta es, el aprendizaje convergerá más rápido, pero los saltos serán mayores pudiendo

saltarse los mínimos. Y cuanto más baja, el aprendizaje se realiza más lentamente y puede quedar atascado en un mínimo local.

Bias: Es un valor de sesgo que se añade a la red neuronal.

Programmable Logic Controller (PLC): Es un ordenador programable utilizado en las líneas de montaje o producción de plantas industriales generalmente, para automatizar procesos como movimientos de transportes o vehículos guiados.

Forward: Hace referencia al procesado hacia adelante de una imagen por una red neuronal para obtener las características finales.

Backward: Es la acción que se realiza cuando la imagen ha sido procesada. El error se retropropaga hacia atrás en la red neuronal para modificar los pesos obtenidos mediante las características obtenidas de la imagen.

10.2 Tabla de Resultados Completa

	Red Neuronal	DataSet			Train				Test		
Nombre Dataset	Modelo Base	Núm. Clases	Tamaño	Data Augmentation	Tamaño Train	# Epochs	Pre-Train	Duración min.	Tamaño Test	% mAP	% Acierto
HandGesture (HG)	AlexNet	27	899	0	874	20	NO	25	25	76,55%	44,00%
HandGesture (HG)	VGG16	27	899	0	874	20	NO	50	25	77,78%	52,00%
HG	VGG16	3	99	0	96	20	NO	5	3	100,00%	100,00%
HG	VGG16	3	996	900	96	20	NO	5	900	10,39%	50,11%
HG	VGG16	3	2425	2329	2200	20	NO	135	225	10,78%	98,66%
HG	VGG16	3	201	105	201	20	NO	15	40	24,72%	100,00%
HG	VGG16	3	1683	1587	1487	20	NO	120	196	12,65%	98,47%
HG	VGG16	3	9263	9167	9263	20	NO	630	225	???	98,66%
HG	VGG16	3	9263	9167	9263	20	NO	630	40	10,32%	60,00%
HandGesturePropias (HGP)	VGG16	4	432	216	400	20	NO	25	32	36,54%	81,25%
HGP	VGG16	4	864	432	777	20	NO	50	87	29,65%	74,13%
HGP	VGG16	4	1664	832	1497	20	NO	95	167	32,03%	74,85%
HGP	VGG16	4	1763	832	1593	20	NO	105	170	29,73%	75,29%
HGP	VGG16	4	1763	832	1593	75	SI	600	170	19,85%	95,88%
HGP	VGG16	4	1763	832	1593	20	NO	100	170	21,14%	82,35%
HGP	VGG16	4	1664	832	1497	75	SI	600	167	18,08%	91,03%
HGP	AlexNet	4	1763	832	1593	20	NO	50	170	42,29%	72,36%
HGP	AlexNet	4	1763	832	1593	75	SI	300	170	19,64%	83,53%
HGP	AlexNet	4	11649	10348	11279	20	NO	330	370	17,75%	70,54%
HGP	AlexNet	4	11649	10348	11279	75	SI	1800	370	10,84%	85,41%
HGP	VGG16	4	1763	832	1593	75	SI	600	370	10,87%	95,14%
HGP	VGG16	4	11649	10348	11279	75	SI	3900	370	14,69%	94,59%

	Red Neuronal	DataSet			Train				Test		
Nombre Dataset	Modelo Base	Núm. Clases	Tamaño	Data Augmentation	Tamaño Train	# Epochs	Pre-Train	Duración min.	Tamaño Test	% mAP	% Acierto
HGP	AlexNet	4	1763	832	1593	75	SI	300	370	16,03%	90,27%
HGP	VGG16	4	3563	1632	3193	75	SI	600	370	14,15%	91,35%
HGP	VGG16	4	4217	1890	3762	75	SI	1260	455	9,76%	92,31%
HGP	VGG16	4	5654	2109	5054	75	SI	1680	600	8,45%	97,50%
HGP	VGG16	4	7651	3826	7001	75	SI	2130	650	14,26%	97,08%
HGP	VGG16	4	8440	4615	7790	75	SI	2160	650	14,60%	97,69%
HGP	VGG16	4	22575	3321	21475	75	SI	7500	1100	9,29%	98,73%
HGP	VGG16	4	25431	26954	24331	75	SI	7800	1100	8,48%	99,00%
NewHGP	VGG16	4	8440	4615	7790	75	SI	2160	650	10,00%	94,92%
NewHGP	VGG16	4	21519	23042	20419	75	SI	7200	1100	8,01%	97,55%
DataAGV	VGG16	4	1243	0	1012	75	SI	420	231	11,69%	79,65%
DataAGV	VGG16	4	4857	3614	4357	75	SI	2000	500	15,86%	86,60%
DataAGV	VGG16	4	12446	8832	11446	75	SI	4200	1000	16,69%	87,30%
DataAGV	VGG16	4	22570	20646	21570	75	SI	7800	1000	10,95%	93,50%
DataAGV	VGG16	4	19574	16875	18574	75	SI	6600	1000	10,14%	94,20%
DataAGV	VGG16	4	19574	16875	18574	75	SI	6600	1000	10,24%	94,30%
BackAGV	VGG16	5	19658	16875	18658	75	SI	7200	1000	26,71%	96,00%
NoisyAGV	VGG16	4	23067	20368	22067	75	SI	8700	1000	24,72%	95,90%
NoisyAGV	VGG16	5	23151	20368	22151	75	SI	8700	1000	20,01%	94,50%
DualBackAGV	VGG16	5	27499	24376	26499	75	SI	12900	1000	26,09%	96,70%
DualBackAGV	VGG16	5	32923	30620	31923	75	SI	15000	1000	23,14%	96,40%
DualBackAGV	VGG16	5	35040	16496	34040	75	SI	14400	1000	24,49%	97,70%

Tabla 28 - Tabla de Resultados Completa