Universidad Internacional de La Rioja (UNIR)

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Inteligencia Artificial

Desarrollo e implementación IoT de un sistema de reconocimiento de imágenes a nivel industrial.

Trabajo Fin de Máster

Presentado por: Sanz Cabreros, David

Director/a: Medina Quero, Javier

Ciudad: Valladolid
Fecha: 17/02/2019

Índice de contenido:

Índice de ilustraciones:

Índice de tablas:

Resumen

El objetivo de este proyecto es diseñar un reconocedor de objetos basado en la API “Object Detection” de TensorFlow con la finalidad de aplicarlo a nivel industrial en una planta de fabricación de luminarias para mejorar uno de sus procesos de fabricación. La mejora que se propone con respecto a los modelos actuales de detección de objetos del mercado a nivel industrial es la generación de dicho reconocedor con técnicas de aumentación de datos, donde a partir de unas pocas imágenes de entrada de las diferentes piezas a detectar y de un video del fondo donde estarán localizadas. Esto permitirá crear un modelo suficientemente potente como para detectar los objetos deseados, suprimiendo la parte de generación de cantidades enormes de datos de entrada y facilitando así su desarrollo y actualización. Tras su implementación física, la transmisión de información para su visualización se realizará mediante dispositivos IoT.

Palabras Clave: Object Detection, TensorFlow, Aumentación de Datos, Vision artificial industrial, Dispositivos IoT.

Abstract

The objective of this project is to design an object recognizer based on the “Object Detection” API of TensorFlow with the purpose of applying it to an industrial level in a luminaire manufacturing plant to improve one of its manufacturing processes. The improvement proposed with respect to the current models for detection objects in the market at the industrial level is the generation of said recognizer with data augmentation techniques, where from a few input images of the different pieces to be detected and a video of the background where they will be located. This will allow creating a sufficiently powerful model to detect the desired objects, suppressing the generation part of enormous amounts of input data and facilitating its development and updating. After its physical implementation, the transmission of information for its visualization will be done through IoT devices.

Keywords: Object detection, TensorFlow, Data Augmentation, Industrial artificial vision, IoT Devices.

1. Introducción:

Dentro de la visión artificial, el reconocimiento de objetos en imágenes y su identificación es una de las áreas más atractivas y que mayor desarrollo han tenido en los últimos años. Gracias a diferentes tecnologías desarrolladas en la última década, el reconocimiento de objetos en imágenes se ha convertido casi en un problema abordable y configurable mediante técnicas de Machine Learning. El gran desafío actual es el de mejorar estos sistemas de reconocimientos de imágenes, puliendo ciertos detalles como su requerimiento de grandes datos, capacidad de cómputo y su precisión de reconocimiento, con la finalidad de que esta tecnología mejore lo necesario como para formar parte de nuestro día a día, ayudándonos en nuestras vidas hasta tal punto de volverse imprescindible.

Estos grandes avances que ha sufrido el área de la visión artificial en los últimos años ya han permitido la creación e implementación de diversas aplicaciones basadas en el reconocimiento de imágenes para mejorar o ayudar a procesos actualmente realizados de forma manual en diferentes áreas de la sociedad como en la medicina, la seguridad de las ciudades, los automóviles, el comercio, la industria…

Y es en esa área de la industria en el cual se centra este trabajo final de máster. La industria está sufriendo un proceso evolutivo hacia lo que se denomina la 4ª revolución industria, la cual tiene como principales objetivos lograr una digitalización total de sus procesos, con la finalidad de obtener la mayor cantidad de datos posibles para mejorar sus procesos productivos.

Mejorar la calidad de los productos y la eficiencia de los procesos, son 2 de los aspectos clave que toda empresa quiere mejorar y optimizar, y la tecnología es una buena compañera de camino para conseguirlo. Gracias a los últimos avances tecnológicos, son muchas las empresas que desde hace años buscan la forma de implementar en sus factorías sistemas inteligentes que aporten un gran beneficio en sus procesos, y entre estos sistemas inteligentes, los sistemas de visión artificial son sin duda uno de los elementos primordiales.

Este trabajo final de máster tiene como finalidad desarrollar un sistema de visión artificial, basado en el reconocimiento de imágenes, con el objetivo de ser implementado en el proceso productivo de pintura de la fábrica Signify Manufacturing Spain S.L., localizada en Valladolid, con la finalidad de mejorar dicho proceso, generando nueva información para la planta sobre el material producido, y aportando datos necesarios para el cálculo de aprovechamiento, eficiencia y calidad del proceso en el que será implementado. En trabajos

futuros, se podrá emplear estos datos con otras finalidades como dotar a la zona de producción de una mayor automatización, pudiendo incluso transmitir la información de las piezas detectadas a las cabinas de pintado para eliminar así los procesos manuales actuales y aumentar la disponibilidad de la planta.

La parte innovadora que pretende este proyecto es la de diseñar un sistema de reconocimiento de imágenes implementado directamente en un entorno industrial, sin grandes gastos económicos de herramientas comerciales, y de una forma ágil y asequible siendo capaz de diseñar un sistema suficientemente potente de reconocimiento de imágenes a partir de una pequeña cantidad de imágenes de entrada. Además, propone la creación de un entorno inteligente dentro de la factoría a través de dispositivos IoT obteniendo la información generada por el clasificador y visualizándolo en la parte deseada de la instalación vía internet sin necesidad de cableados.

2. Estado del arte:

Antes de empezar con la propuesta y desarrollo del TFM, es conveniente hacer una breve introducción sobre el área tecnológica en el que se encuentran enmarcadas las técnicas empleadas, los conceptos principales en los que se apoyará el sistema de percepción y que serán mencionados a lo largo del desarrollo del TFM; así como describir los sistemas de detección o técnicas existentes en la actualidad para resolver casos similares al propuesto en este trabajo. El objetivo de este apartado no es explicar en detalle las diferentes etapas históricas de la inteligencia artificial y de la visión computacional, pero si es conveniente hacer un breve repaso de los conceptos clave.

2.1. Inteligencia artificial:

De manera muy sencilla, se podría definir inteligencia artificial como una tecnología basada en sistemas que muestran inteligencia exhibida por máquinas (Gross, 1992).

Según la Real Academia Española, la inteligencia artificial se podría definir como “la disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como aprendizaje o el razonamiento lógico”.

En definitiva, la inteligencia artificial es aquella área tecnológica que se encarga de dotar a las máquinas de cierta “inteligencia”, entendiendo por inteligencia como la capacidad de resolver problemas o de replicar aquellos procesos u operaciones de una forma similar a como lo haría un ser humano (Pastor, 2018).

Dentro de la inteligencia artificial (cuya definición exhaustiva no forma parte de este proyecto), existen diferentes áreas o casos de uso, las cuales, de forma más o menos completa, se podrían dividir de la siguiente forma:

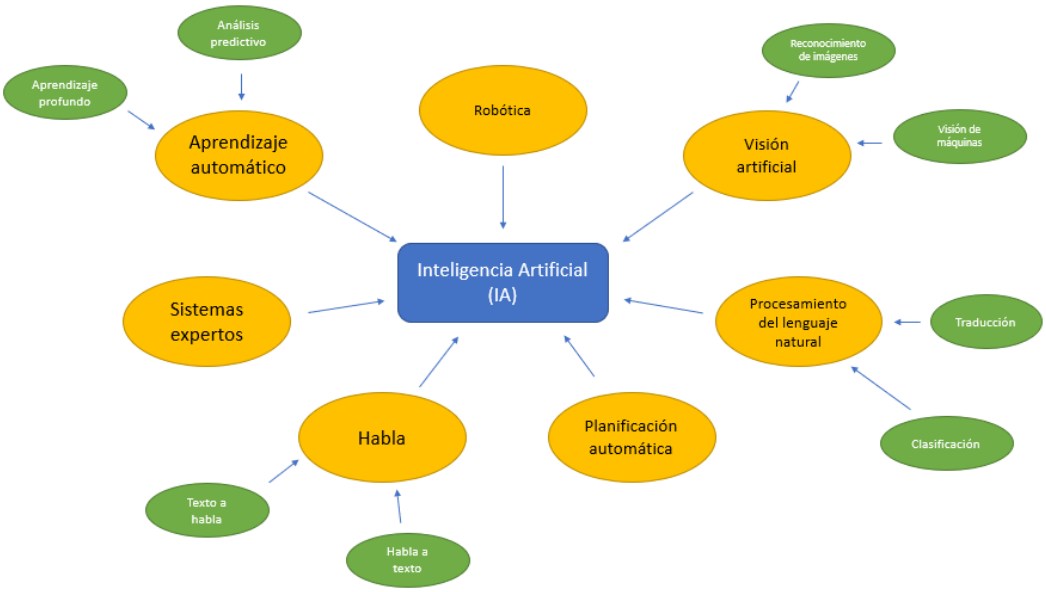

ILUSTRACIÓN 1: APLICACIONES MODERNAS DE LA INTELIGENCIA ARTIFICIAL

De entre todas estas disciplinas situadas dentro de la inteligencia artificial (Galipienso, 2003), la aplicación objetivo de este trabajo se encuentra enmarcada dentro del área de la visión artificial (Davies, 2004). Por eso mismo, será la técnica que será explicada más en detalle en los siguientes apartados.

Es importante recalcar, que la clasificación o distinción mostrada anteriormente en grupos de tecnologías no es fija, sino que las diferentes tecnologías o áreas de la inteligencia artificial pueden combinarse, y de hecho se combinan. En el desarrollo de esta aplicación se usarán técnicas de visión, basadas en Deep Learning con la finalidad de completar el desarrollo total del sistema.

2.2. Visión artificial:

a) Descripción:

Como se ha comentado anteriormente, el área de la visión artificial, o visión por ordenador ("*Computer vision"*) es una disciplina englobada dentro del área de la inteligencia artificial, la cual, como define la Asociación de Imágenes Automatizadas (AIA), "incluye todas las aplicaciones industriales y no industriales donde una combinación de hardware y software brindan una guía operativa a dispositivos en la ejecución de sus funciones en base a la captura y el procesamiento de imágenes".

Dentro de las diferentes áreas de aplicación de la visión industrial, de forma simple, se podría hacer una división en 2 categorías como la siguiente:

- Visión artificial académico/científica.
- Visión artificial industrial.

Usualmente en el ámbito académico o educativo, se emplean dispositivos sin un gran alto nivel económico y computacional, cuya finalidad es la de ser empleados en la enseñanza con el fin de crear sistemas de enseñanza asistida por ordenadores o tutores inteligentes, más que el de obtener una gran precisión (Gross, 1992). La creación de sistemas inteligentes de enseñanza-aprendizaje entre los alumnos y los docentes (Pollo Cattaneo, 2016, May) son áreas donde la visión artificial tiene y tendrá un gran impacto durante los futuros años, gracias a técnicas como la realidad aumentada (Basogain, 2007).

Los dispositivos empleados en el sector industrial suelen ser mucho más potentes, y se basan, como comenta una de las mayores empresas del sector industrial de la visión artificial como Cognex, "en sensores digitales protegidos dentro de cámaras industriales con ópticas especializadas en adquirir imágenes, para que el hardware y el software puedan procesas, analizar y medir diferentes características para tomar decisiones" (Cognex, 21/02/2019). Estos dispositivos, normalmente llevan incorporado en su interior sistemas de procesamiento capaces de realizar todo el procesado de la imagen, así como sacar su información. La mayoría de estos proyectos industriales que emplean visión artificial se basan en la detección de problemas de calidad para mejorar procesos como el perforado de PCB (Cárdenas, 2015), o para la clasificación de diferentes elementos dentro de los procesos productivos, como minerales presentes en placas metálicas (Castroviejo, 2008), alimentos como frambuesas (Constante, 2016) u otros tipos de materiales.

Disponer de sistemas de bajo coste en áreas donde se requiere una mayor precisión y para los que usualmente se requieren sistemas computacionalmente más costosos, podría ser un gran avance para la industria, que encontraría en estos sistemas una forma fácil y barata de disponer de sistemas tan potentes y útiles para mejorar sus procesos. Como ejemplo se presenta un artículo cuya finalidad era la de diseñar un sistema embebido de bajo costo para realizar diferentes aplicaciones de visión artificial (Aguirre, 2014).

b) Herramientas principales:

Con el fin de desarrollar un sistema de visión artificial industrial más económico de lo disponible actualmente, se van a necesitar una serie de herramientas fundamentales para su desarrollo.

En este apartado se pretende hacer una breve descripción de las herramientas o conceptos empleados a lo largo de la memoria, incluyendo tanto lenguajes de programación empleados, como *"frameworks"* o algoritmos de *"Machine Learning"* usados, así como diferentes dispositivos o técnicas empleadas en el desarrollo de la aplicación que no necesariamente están incluidos dentro de la visión artificial, sino más enfocados en el apartado IoT que serán empleados en la implementación del sistema de forma física.

- **Object Detection API:**

  Para el desarrollo de aplicaciones típicas como es el caso del reconocimiento de objetos en imágenes, reconocimiento de voz, clasificación de texto… TensorFlow (Abadi, 2016, November) aporta una serie de API's (interfaz de programación de aplicaciones) (GitHub: Tensorflow Models, 21/02/2019) que permiten abstraerse del diseño de los diferentes modelos necesarios para las diferentes aplicaciones, y usar modelos ya existentes a partir de los cuales, a partir de pequeñas modificaciones, adaptar dichos modelos a los casos particulares.

  "Object Detection" (Huang, 2017) es una de estas API's que como menciona en la página oficial (GitHub: Tensorflow Object Detection API, 21/02/2019) permite "localizar e identificar múltiples objetos en una sola imagen". Los archivos contenidos en esta API serán los empleados a lo largo del proyecto.

- **OpenCV**:

  Si hablamos de visión artificial, esta biblioteca desarrollada por Intel es sin ninguna duda una de las más importantes a nivel mundial (Bradski, 2008). Desarrollada tanto

para Windows, Mac OS como Linux, incluye multitud de funciones de visión artificial, tanto para el reconocimiento de imágenes, como para su tratamiento.

Programado inicialmente en C++, esta librería incluye versiones de Python lo que la convierte en una librería fundamental para el desarrollo del trabajo.

- **Aumentación de datos:**

Una de las técnicas más empleadas en los últimos años en el mundo del Deep Learning, en especial para las aplicaciones de visión artificial es la aumentación de datos. Los modelos de visión artificial, así como otros modelos que utilicen grandes cantidades de datos, requieren de gran multitud de información, imágenes de los objetos a predecir en el caso de la visión artificial, para servir de entradas al modelo durante el entrenamiento. Esto provoca que el tiempo de recolección de estos sea muy elevado, lo cual causa serios problemas cuando o no se dispone de tiempo suficiente como para recopilar toda esta información, o directamente no se puede disponer de esta.

Para solucionarlo, surgió la técnica denominada en ingles *Data Augmentation* o aumentación de datos la cual consiste en “agregar valor a los datos básicos al agregar información derivada de fuentes internas y externa” (Techopedia, 21/02/2019). Cualquier modificación realizada sobre los datos de forma artificial con el fin de generar nuevos datos de entrada con ligeras modificaciones se podría considerar aumentación de datos.

En el caso de la visión artificial, esto es muy utilizado debido a la necesidad de cantidades elevadas de imágenes de las que normalmente no se dispone. Agregando pequeñas modificaciones a las imágenes como traslaciones, rotaciones, “flips”, escalados, recortes, adición de ruido Gaussiano u otra serie de alteraciones, se pueden generar muchas imágenes nuevas a partir de una imagen original.

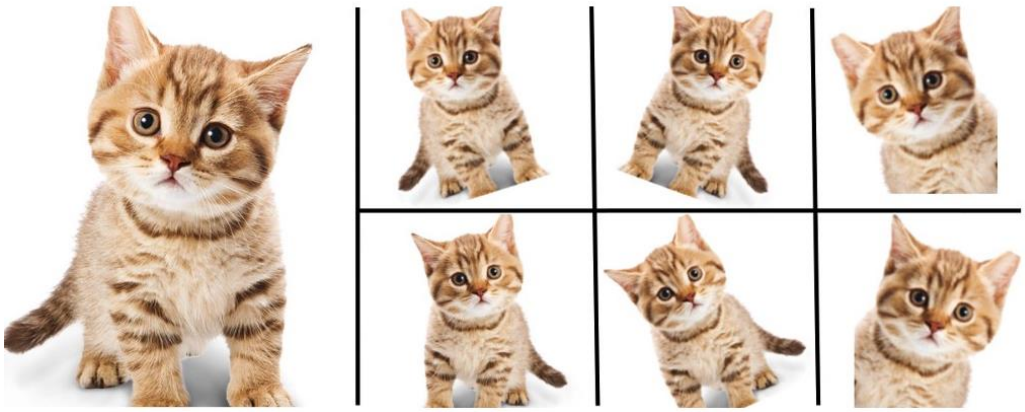

ILUSTRACIÓN 2: EJEMPLO DE DATA AUGMENTATION

En artículos como el presentado por Bharath Raj (Raj, B., 2018) de donde se ha sacado la ilustración anterior, o en libros como el de McLaughlin, N, (McLaughlin, 2015, August), el de Fawzi A. (Fawzi, 2016) o en el de Zhu, X (Zhu, 2017) explican el funcionamiento de esta técnica mostrando casos de uso en los cuales se ha empleado aumentación de datos para diferentes objetivos y aplicaciones.

El objetivo es aportar a la red neuronal, diferentes imágenes de un mismo objeto con pequeñas modificaciones, que obliguen a la red a ser capaz de generalizar conceptos en vez de memorizarlos:

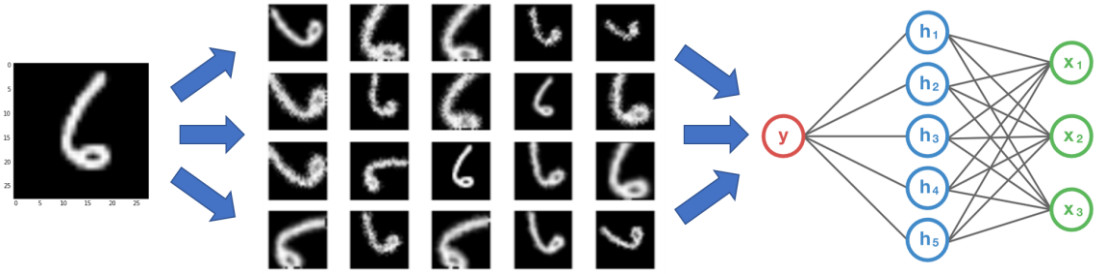

ILUSTRACIÓN 3: RED NEURONAL CON DATA AUGMENTATION

Esto será de gran utilidad durante el proyecto, puesto que la idea principal que se desarrollará más adelante es la de diseñar un clasificador a partir de unas imágenes de entrada generadas a partir de realizar modificaciones sobre unas pocas imágenes originales de las piezas a detectar. Para ello, se generarán imágenes de cada una de las piezas a las cuales se las eliminará el fondo, y posteriormente se sobrepondrán sobre el fondo original sobre el cual estarán cuando la aplicación este instalada.

Como se menciona en el artículo de investigación (Medina Quero, 2018), la aumentación de datos es una potente solución empleada para disminuir el riesgo de overfitting (Krizhevsky, 2012), debido a que cuanta más variedad de información sea aportada al modelo durante el entrenamiento, menos capacidad de memorización tendrá, así como reducir el problema de la necesidad de gran cantidad de datos para aportar el modelo (Ciresan, Meier, Masci, Gambardella, & Schmidhuber, 2011). En estudios recientes se ha abordado estas soluciones propuestas a través de aumentación de datos (Georgakis, 2017) (Dwibedi, 2017, October), en los cuales se ha mostrado la posibilidad de emplear pequeñas imágenes anotadas de los objetos superpuestas sobre fondos ambientales para generar nuevos ejemplos de entrada empleando transformaciones y operaciones de datos como las explicadas en este apartado. En el caso de la "*Activity Recognition*" AR, tradicionalmente se han empleado videos

etiquetados desarrollando herramientas específicas para dichas anotaciones. Sin embargo, herramientas más simples pueden usarse con la misma finalidad, como se propone en el artículo mencionado al principio de este párrafo (Medina Quero, 2018) y como se desarrollará en este trabajo.

De entre las diferentes librerías de las que dispone OpenCV, una de las más importantes a tener en cuenta durante el desarrollo de este trabajo es GrabCut. El algoritmo GrabCut, diseñado por C. Rother, V. Kolmogorov y A. Blake para *Microsoft Research Cambridge*, UK (Rother, 2004) es una librería que permite eliminar el fondo de las imágenes a través de la selección de diferentes áreas de esta. Esta tecnología será la empleada durante el desarrollo del proyecto, en cierta medida gracias a la flexibilidad aportada por la herramienta al permitir utilizar la librería en códigos propios agilizando el proceso de extracción del fondo. En la parte de elaboración del proyecto se mostrará detalladamente los pasos necesarios para su utilización.

- **Transfer Learning:**

El Transfer Learning o aprendizaje por transferencia, es un método de aprendizaje automático en el cual se emplean modelos pre-entrenados para una tarea, reutilizándolo para nuevas tareas, con la finalidad de reducir el tiempo de cómputo requerido para desarrollar modelos de redes neuronales (Brownlee, J., 2017, December).

Transfer Learning es una técnica de Machine Learning en la cual un modelo entrenado en una tarea se emplea para otra (Goodfellow, 2016), para lo cual, a los modelos de redes neuronales ya entrenados, se les pasan imágenes o datos de entrada del nuevo modelo deseado con el fin de modificar las últimas capas encargadas de identificar objetos concretos, manteniendo las primeras y las capas intermedias las cuales son las encargadas de reconocer partes o conceptos más genéricos de las imágenes (como bordes o zonas de contorno) (Olivas, 2009).

En la imagen siguiente se muestra la diferencia de precisión obtenida empleando técnicas de aprendizaje por transferencia comparadas con modelos creados desde cero, en los que se ven las ventajas de emplear modelos pre-entrenados con redes neuronales diseñadas anteriormente, cambiando simplemente el entorno de objetos a detectar (Brownlee, J., 2018).

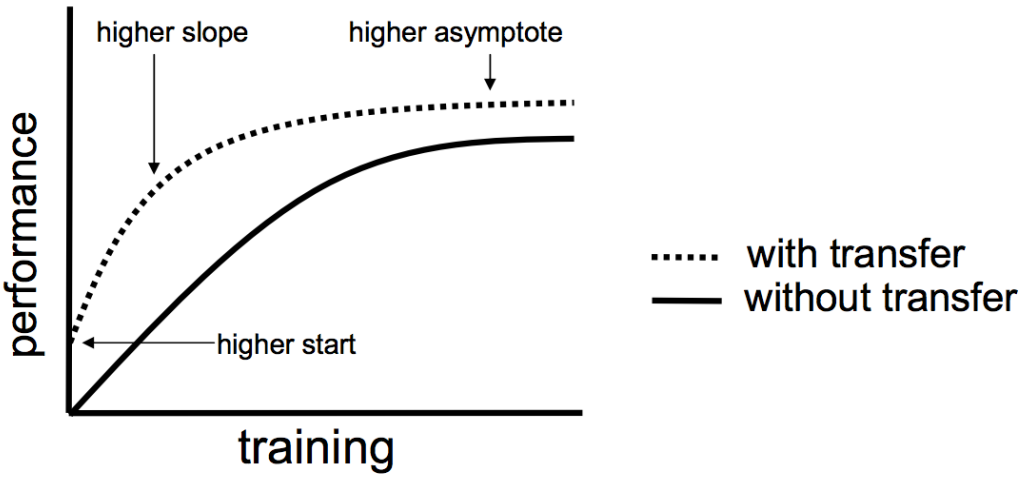

ILUSTRACIÓN 4: COMPARACIÓN DE ENTRENAMIENTO CON Y SIN TRANSFER LEARNING

- **Dispositivos IoT:**

Los dispositivos IoT (internet de las cosas) son dispositivos que se basan el concepto IoT, el cual se refiere a una interconexión digital de objetos cotidianos con internet. El uso de este tipo de dispositivos y su conexión a través de internet es una de las tecnologías más en auge en la actualidad. Como se define en el libro "*Internet of Things (IoT): A vision, architectural elements, and future directions*." (Gubbi, 2013), "La detección ubicua habilitada por las tecnologías de la Red de sensores inalámbricos (WSN) atraviesa muchas áreas de la vida moderna. Esto ofrece la capacidad de medir, inferir y comprender indicadores ambientales, desde ecologías delicadas y recursos naturales hasta entornos urbanos. La proliferación de estos dispositivos en una red de comunicación y activación crea la Internet de las cosas (IoT), en la que los sensores y los actuadores se combinan a la perfección con el entorno que nos rodea, y la información se comparte entre plataformas para desarrollar una imagen operativa común (COP)".

Crear ambientes inteligentes donde los dispositivos se comunican entre sí es uno de los avances más potentes de los últimos años y uno de los retos más importantes que tendrán que afrontar las empresas en los próximos años para mantenerse a la vanguardia del sector industrial (Lee, 2015).

Durante estos años han aparecido gran cantidad de proyectos relacionados con dispositivos y la tecnología IoT para soluciones como información sobre vehículos y su situación física para solucionar problemas de transporte (He, 2014), monitorización de acciones cotidianas domesticas relacionadas con la domótica (Kelly, 2013), ciudades

inteligentes donde se puedan recoger datos de la situación actual de las mismas en tiempo real (Sanchez, 2014), etc.

Dentro de la parte de IoT tanto para domótica como para ciudades inteligentes, es importante destacar que Signify, compañía para la que se desarrollará el proyecto, es una de las pioneras en el mundo de los dispositivos IoT con la plataforma IoT Interact (Signify, La plataforma de IoT Interact, 2018), la cual "permite recopilar información valiosa de la iluminación LED, los sensores integrados, los dispositivos IoT y las aplicaciones software Interact para ayudar a tomar decisiones más acertadas y eficientes".

Tanto en el mundo de la domótica con dispositivos como Philips Hue, en el mundo de la iluminación profesional urbana con sus sistemas Citytouch (Signify, CityTouch: Sistema de gestión de la iluminación, 21/02/2019), como en el desarrollo de nuevas formas de comunicación entre dispositivos IoT como la tecnología LiFi (Signify, Introducing LiFi, 21/02/2019), dicha compañía ha demostrado apostar fuertemente por las tecnologías IoT emergentes, haciendo prácticamente imprescindible y necesario la creación de un entorno dentro de sus factoría lo más inteligente posible.

Para lograr la interconexión de estos dispositivos de una manera económica y crear un ambiente inteligente, en el proyecto se propondrá el uso de dispositivos como Arduino (Banzi, 2014) o Raspberry Pi (Richardson, 2012), implementando el sistema de reconocimiento de objetos, almacenando los datos obtenidos en la red de la fábrica, y haciendo así que los datos generados aporten el valor necesario para que la compañía pueda usarlo para futuros proyectos.

2.3. Detección de objetos (“*Object Detection”):*

Este apartado profundizara en aplicaciones de visión artificial concretas centradas en el reconocimiento, detección y clasificación de objetos, haciendo un repaso de las diferentes técnicas empleadas desde que se planteó el objetivo de ser capaces de detectar y clasificar objetos, hasta las técnicas actuales más empleadas para dicho propósito.

Para ello, es conveniente diferencia entre las versiones más clásicas de detectores de objetos en imágenes basadas en algoritmos clásicos de aprendizaje automático, y las nuevas tendencias de algoritmos de aprendizaje automático basadas en redes de neuronas convolucionales (CNN) o aprendizaje profundo (“Deep Learning”).

a) Modelos previos usados para el reconocimiento de objetos:

- **Máquinas de vector soporte (SVM):**

En los sistemas de visión artificial clásicos, las Máquinas de Vector soporte han sido una de las tecnologías más empleadas para ser capaces de reconocer objetos en las imágenes. Las SVM, desarrolladas por V. Vapnik son un tipo de algoritmos de aprendizaje supervisado, centrados en el área de clasificación o regresión que presentaron una gran revolución conceptual en su época, por su idea de separar los diferentes ejemplos de entrenamiento en hiperplanos para diferenciar entre clases.

Son diversos los proyectos que se han centrado en el uso de estas tecnologías para el reconocimiento de imágenes, como es el caso de la tesis de grado de Condori Arias y Elvis Franks, publicado por la Universidad Nacional de Ingeniería, en la cual “se desarrolla un sistema de reconocimiento y clasificación de objetos, el cual trata de emular la forma como los humanos percibimos la información visual mediante uso de dos cámaras ópticas, que actúan como nuestros ojos” (Condori Arias, 2013). El problema que presenta esta tecnología reside en que no son capaces de extraer características o patrones pese a que son muy buenos clasificadores.

- **SIFT:**

Otro de los algoritmos de aprendizaje automáticos clásicos empleados en el área de la visión artificial clásica para la extracción de características ha sido SIFT (“Scale-invariant feature transform”), cuyo objetivo o finalidad es la de encontrar puntos relevantes en las imágenes de entrada basándose sobre todo en la detección de esquinas o bordes.

Un ejemplo de proyecto de clasificación de objetos basado en esta técnica es el proyecto de Villasenor, M. M. D., y Fernández, L. C. (2015), el cual “se propone el uso de las características SIFT para resolver problemas de reconocimiento de clases de objetos en imágenes” (Vilasenor, 2015).

- **SURF:**

Siguiendo con las diversas tecnologías existentes, una más avanzada que las anteriores es SURF (Speeded Up Robust Features), el cual es un detector empleado en tareas de reconocimiento de objetos, registro de imágenes o clasificación, el cual está basado en el descriptor SIFT mencionado anteriormente. En el artículo *Speeded-up robust features (SURF)* (Bay, 2008) se explica en detalle el funcionamiento de esta tecnología y las ventajas que presentan ante sus técnicas predecesoras.

- **Histogramas de gradientes orientados (HOG):**

Otra de las técnicas empleadas tradicionalmente en el área de la visión artificial y el procesamiento de imágenes para la detección de objetos son los histogramas de gradientes orientados, la cual se basa en la cuenta de ocurrencia de gradientes orientados presentes en porciones localizadas de una imagen. Es una técnica parecida a SVM, pero difiere en que se calcula en un conjunto de celdas específicas a través de cuadriculas densas.

Un ejemplo de caso de aplicación de estas técnicas en el reconocimiento de imágenes es el artículo de Merchán, F., Galeano, S., & Poveda, H. el cual “aborda aspectos de entrenamiento de la máquina de aprendizaje AdaBoost con modelos de reconocimiento de objetos basados en características de apariencia tales como: Patrones Binarios Locales (LBP), Histograma de Gradientes Orientados (HOG) y características tipo Haar para la detección de sonrisas” (Merchán, 2016)

b) Aprendizaje profundo (“*Deep learning*”):

En este apartado se hará un repaso por los diferentes modelos presentados en los últimos años en el área del aprendizaje profundo, mencionando algunos artículos, pero sin entrar en detalle en cada uno de ellos, puesto que el objetivo de este apartado es solo dar una idea de los diferentes modelos, no de entrar en detalle en cada uno de ellos.

- **Redes neuronales:**

Las redes de neuronas son un tipo de modelo computacional muy empleado actualmente, el cual se basa en el funcionamiento de los axones y las neuronas de los cerebros biológicos, para tratar de replicar este funcionamiento de forma similar artificialmente (Schalkoff, 1997).

Cada una de las neuronas artificiales recibe una señal de entrada, y en el caso de que se supere el valor definido internamente en la neurona a través de su función de activación, es capaz de generar una salida hasta otra neurona.

La capacidad de estos sistemas de aprender y el buen desempeño que han mostrado en los últimos años para la resolución de problemas de visión por computador, provocan que hoy en día sean una de las principales tecnologías empleadas para resolver estos tipos de problemas. Por eso mismo, será una de las bases del desarrollo de la aplicación de reconocimiento de imágenes del trabajo.

- **Redes convolucionales (CNN):**

Una de las tecnologías que revolucionó el mundo del Deep Learning, y más concretamente de la visión artificial y el área del reconocimiento de objetos, fue la aparición de las redes convolucionales ("convolutional neural network"). Este tipo de red de neuronas profundas, al igual que el resto de las redes de neuronas, tienen su inspiración biológica basado en las neuronas del cerebro humano.

Este tipo de redes tienen sus fundamentos en las ideas mostradas por Kunihiko Fukushima con el Neocognitron (Fukushima, 1982), el entrenamiento usando *Backpropagation* propuesto por Yann LeCun (LeCun, 1998), y los avances logrados por Dan Ciresan y otros al emplear unidades de procesamiento gráfico para su entrenamiento (Ciresan D. C., 2011, July).

La característica principal de estas redes es su diferente estructura de capas en comparación con las redes *feedforward* tradicionales. Estas redes están formadas por una serie de capas especiales para esta arquitectura explicadas a continuación con el apoyo de la información recabada de la web Intelligent (Intelligent, 2018), de la documentación aportada en el artículo "CS231n Convolutional Neural Networks for Visual Recognition" (Karpathy, 2016) y de los libros "Imagenet classification with deep convolutional neural networks" (Krizhevsky, Imagenet classification with deep convolutional neural networks, 2012) y "Visualizing and understanding convolutional networks." (Zeiler, 2014, September):

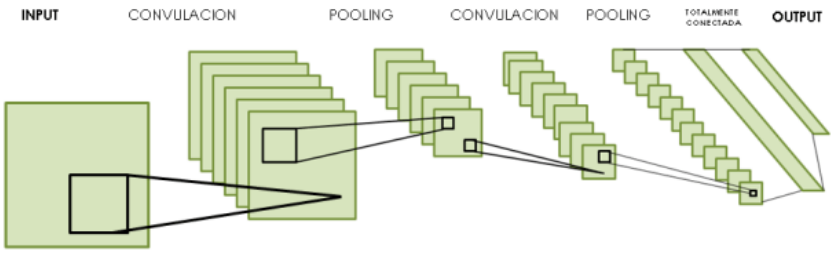

ILUSTRACIÓN 5: ARQUITECTURA CNN

a) **Capas convolucionales:**

Estas capas, las más importantes y las que dan nombre a la arquitectura, emplean filtros de tres dimensiones de pequeño tamaño que van recorriendo la imagen obteniendo las salidas de la capa como se muestra en la siguiente imagen (Karpathy, 2016):

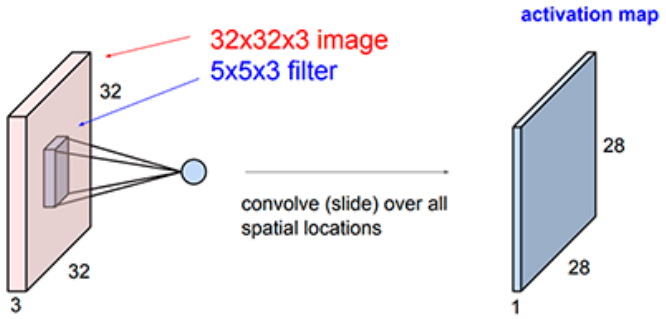

ILUSTRACIÓN 6: CAPAS CONVOLUCIONALES: CONVOLUCIÓN A PARTIR DE UNA IMAGEN

Realizando el mismo proceso con varios filtros, se van creando capas con más dimensiones en profundidad (eje Z), pero reduciendo sus dimensiones nuevas dimensiones en el plano X, Y:

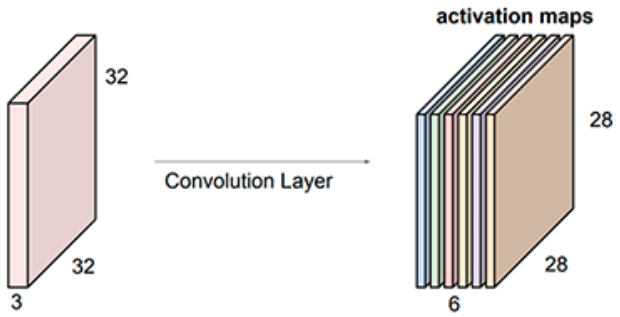

ILUSTRACIÓN 7: CAPAS CONVOLUCIONALES CON VARIOS FILTROS

b) Capas Max Pooling:

Estas capas, suelen ir a continuación de las capas de convolución, y son las encargadas de reducir las representaciones obtenidas de las capas anteriores con el fin de hacerlas más pequeñas y más manejables computacionalmente, reduciendo sus parámetros. A continuación, se muestra una imagen (Karpathy, 2016) del funcionamiento de una capa Max Pooling:

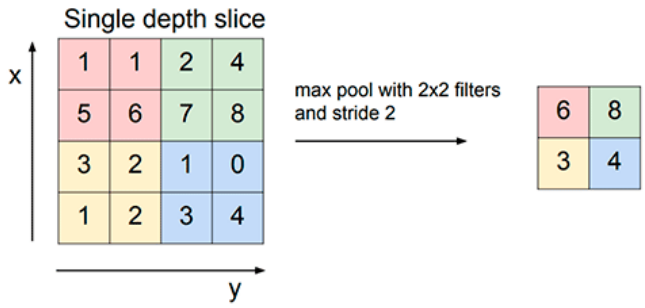

ILUSTRACIÓN 8: CAPAS MAX POOLING

c) Capas Fully Connected:

En la clasificación de objetos a través de redes convolucionales es normal encontrar al final de la red capas Fully Connected tradicionales con el fin de disponer de una neurona por cada una de las clases a identificar y así obtener un resultado para cada clase u objeto a identificar.

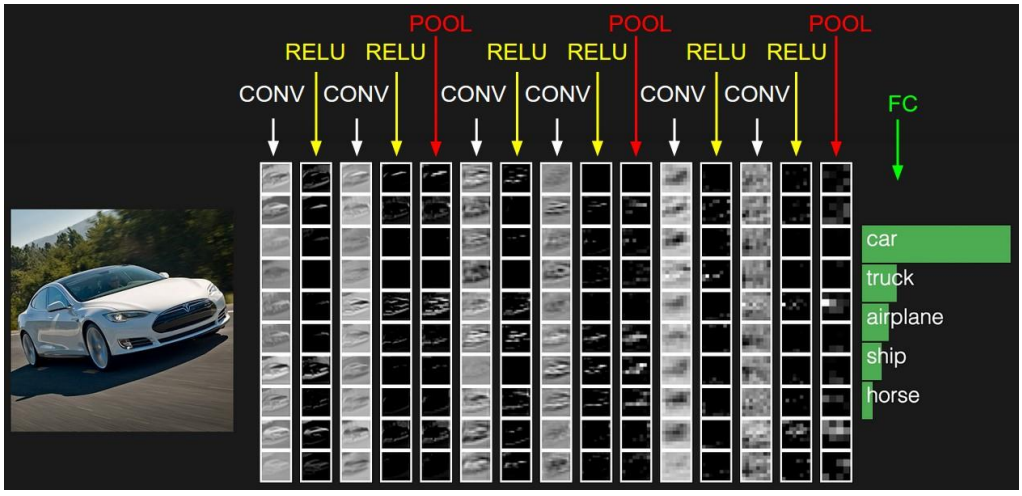

ILUSTRACIÓN 9: EJEMPLO DE ARQUITECTURA CNN APLICADA A LA DETECCIÓN DE UN COCHE

Un proyecto de aplicación de CNN para el reconocimiento de objetos es el de Gidaris, S., & Komodakis, N. (2015), en el cual “proponen un sistema de reconocimiento de objetos basado en una CNN multirregional, que tiene como objetivo capturar un conjunto diverso de factores de apariencia discriminativos y exhibe sensibilidad de localización esencial para la localización de objetos” (Gidaris, 2015).

- **R-CNN y Faster R-CNN:**

Como mejoras a los modelos de redes convolucionales, surgieron modelos como las “Faster R-CNN” cuyo objetivo era reducir el tiempo de ejecución de las redes de neuronas empleadas para detección de objetos.

Dos de los trabajos más importantes sobre las mejoras de estas redes, son el Girshick, R. (2015) en el cual muestra las ventajas de las Fast R-CNN en contraposición de las redes convolucionales previas a esa época (Girshick, 2015), o el trabajo de Ren, S., He, K., Girshick, R., & Sun, J. (2015), en el cual “introducen una RPN *(Regional Proposal Network)* que no es más que una red convolucional que predice simultáneamente los límites de los objetos y las puntuaciones de clasificación de dichos objetos (*objectness scores*) por cada posición” (Ren, 2015).

A continuación, se muestra una imagen explicativa del modelo propuesto, sacado de la referencia bibliográfica mencionada en este apartado:

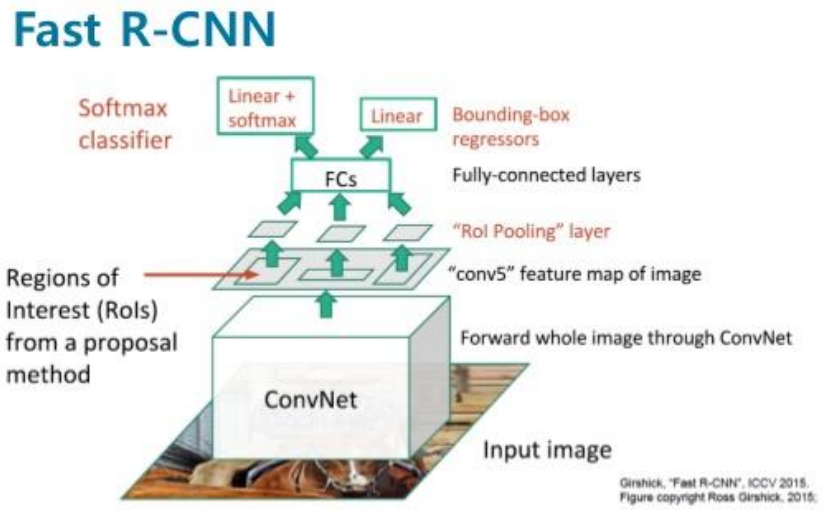

ILUSTRACIÓN 10: MODELO FAST R-CNN

- ***"You Only Look Once"* (YOLO):**

Otro de los últimos avances en el área del reconocimiento de imágenes y que más aceptación está cogiendo por sus buenos resultados es el modelo de detección en tiempo real YOLO propuesto por (Redmon, J., 2016), en el cual, como ellos mencionan en el trabajo, "presentan un nuevo enfoque para la detección de objetos. A diferencia de los modelos anteriores que reutiliza los clasificadores para realizar las detecciones, su modelo encuadra la detección de objetos como un problema de regresión a cuadros delimitadores separados espacialmente y probabilidades de clase asociadas" Dicho modelo es capaz de "procesar imágenes en tiempo real a 45 cuadros por segundo". (Redmon, 2016)

En la siguiente imagen, sacada del trabajo mencionado anteriormente, se puede observar cómo es la arquitectura de la red propuesta por sus diseñadores:

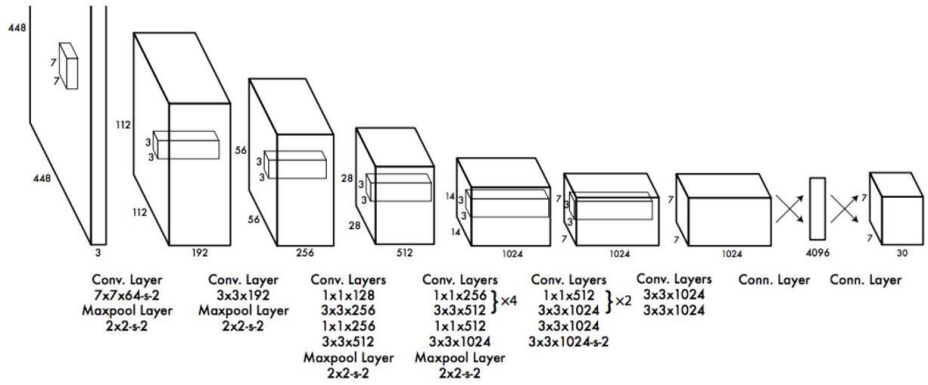

Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

ILUSTRACIÓN 11: ARQUITECTURA YOLO

- **"Single Shot MultiBox Detector" (SSD):**

Como mejora a la velocidad de las Faster R-CNN presentadas anteriormente, en 2016 aparecieron las SSD (Single Sol Multibox Detector), cuyo objetivo es realizar un reconocimiento de imágenes lo más cercano posible al tiempo real.

Como indican en su trabajo (Liu, 2016, October) las SSD "discretizan el espacio de salida de los cuadros delimitadores en un conjunto de cuadros predeterminados sobre diferentes relaciones de aspecto y escalas por ubicación de mapa de características. Durante la predicción, la red genera puntajes para la presencia de cada categoría de

objeto en cada cuadro predeterminado y produce ajustes para adaptarse mejor a la forma del objeto a detectar. Junto con otras mejoras propuestas, hace que el SSD sea fácil de entrenar y sencillo de integrar en sistemas que requieren componentes de detección" (Liu, 2016, October).

- **Comparativa entre los diferentes modelos presentados:**

Para aclarar un poco más lo visto hasta ahora en este apartado, se muestra una comparativa de los diferentes modelos en cuanto a precisión y velocidad, cuya información ha sido sacada del artículo *"Zero to Hero: Guide to Object Detection using Deep Learning"* (SACHAN, 21/02/2019).

En cuanto a la velocidad de detección, se puede observar como el modelo YOLO propuesto por Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. es mucho más rápido que el resto de los modelos, pese a que su precisión de clasificación es un poco peor.

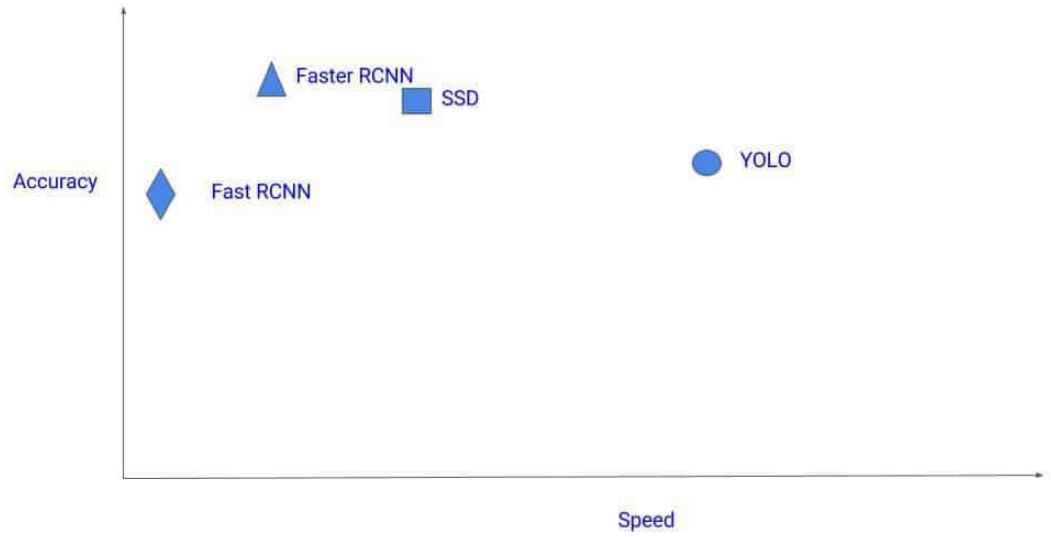

ILUSTRACIÓN 12: YOLO VS SSD VS FASTER R-CNN: PRECISIÓN VS VELOCIDAD

Si nos fijamos en la precisión de clasificación en función del tamaño de los objetos, podremos observar cómo según aumenta el tamaño de estos, las redes SSD se van acercando a las Faster R-CNN, mientras que el modelo YOLO en su versión 2, va mostrando menos precisión.

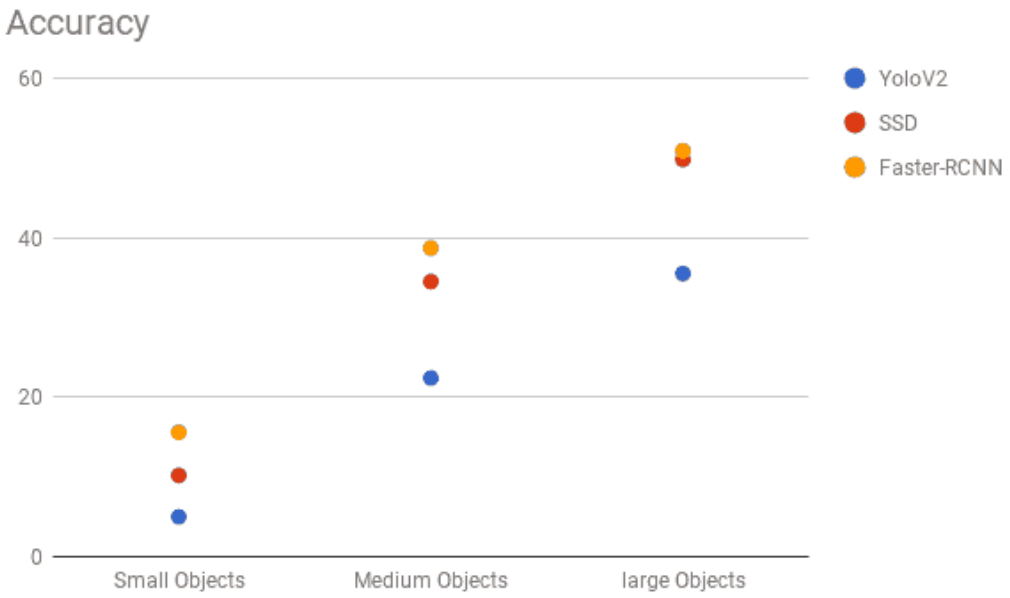

ILUSTRACIÓN 13: YOLO VS SSD VS FASTER R-CNN: PRECISIÓN POR TAMAÑOS

- **Inception network:**

Una de las tecnologías que se empleará en el desarrollo del trabajo, es la diseñada para la red Inception (GoogleNet). Para su explicación se ha recurrido al artículo de Bharath Raj incluido en el apartado de referencias (Raj, A Simple Guide to the Versions of the Inception Network, 2018) así como a diferentes artículos y libros que explican este tipo de arquitecturas (Szegedy C. V., 2016) (Szegedy C. I., 2017).

Esta nueva arquitectura, la cual dispone de diferentes versiones las cuales son una evolución de las otras, tenía como objetivo romper con la visión tradicional de las CNN que se basaban en la aplicación de capas de convolución cada vez más profundas, con la esperanza de obtener un mejor rendimiento.

La primera versión de esta tecnología pretendía resolver problemas como la variación en la ubicación de la información entre imágenes de objetos similares, los problemas de *overfitting* presentados por las redes muy profundas, las cuales son capaces de memorizar objetos, o el alto costo computacional de las redes neuronales muy profundas. Para ello, propuso emplear filtros con múltiples tamaños operando en el mismo nivel, con la finalidad de obtener “más ancho” en lugar de “más profundo”.

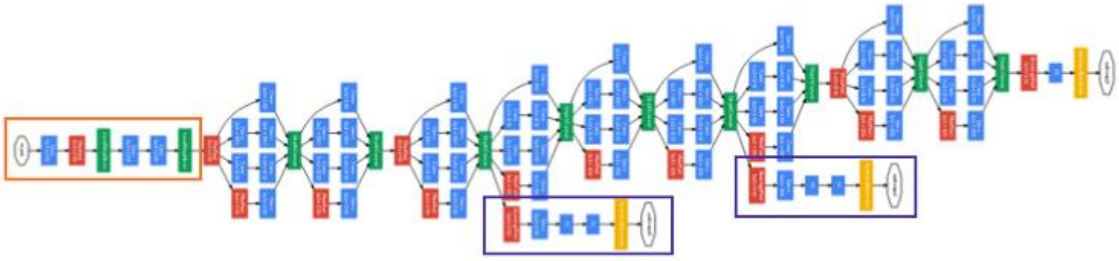

ILUSTRACIÓN 14: ARQUITECTURA INCEPTION V1 (GOOGLENET)

En la imagen de la arquitectura se pueden observar en los cuadros morados los clasificadores auxiliares, así como en las partes anchas de la arquitectura los múltiples filtros en los mismos niveles.

Las diferentes versiones posteriores a esta (v2 y v3) siguieron mejorando problemas como reducir los cuellos de botella producidos al reducir las dimensiones de la entrada, o comprobar si los clasificadores auxiliares contribuían lo suficiente al proceso final como se pensaba, o si actuaban como regularizadores.

La versión 4 de Inception, así como la versión Inception-ResNet, tenía como objetivo hacer los módulos más uniformes, para lo que modificaron el "vástago" (*stem*), introduciendo además los "Bloques de Reducción" (*Reduction Blocks*) empleados para cambiar el ancho y la altura de las cuadriculas.

Para entender un poco mejor las ventajas de los diferentes extractores de características presentados en este apartado, se empleará una imagen del articulo realizado por Jonathan Hui (Hui, 2018) donde hace una comparativa de los diferentes modelos presentados en este apartado con cada uno de los extractores de características de Inception presentados en este punto.

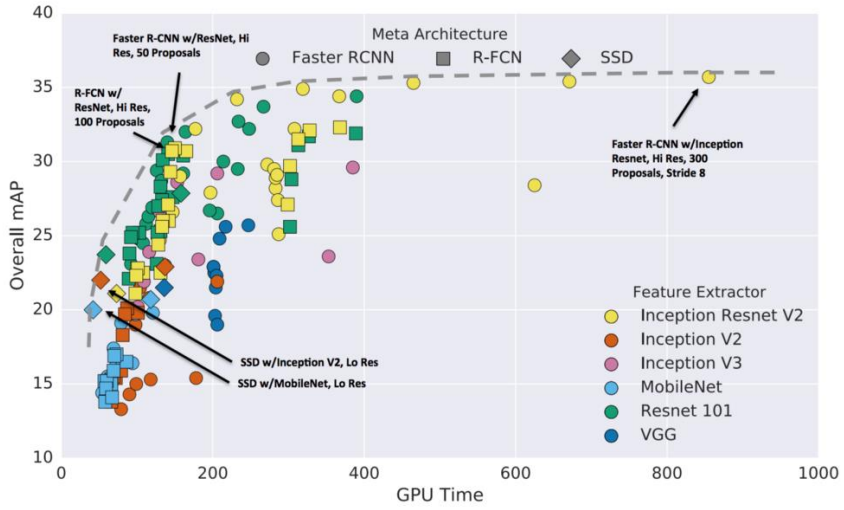

ILUSTRACIÓN 15: INCEPTION: VELOCIDAD VS PRECISIÓN

En esta imagen anterior se observan las diferentes combinaciones de modelos y extractores de características, donde por ejemplo Faster R-CNN es más precisa mientras que R-FCN y SSD son más rápidas.

- **CPU, GPU y TPU:**

Una de las partes más importantes a la hora de entrenar redes neuronales, y más aún para realizar tareas de procesamiento de imágenes, es la definición del procesador que llevará a cabo el proceso de entrenamiento.

Entrenar una red neuronal para el reconocimiento de imágenes es un problema computacionalmente muy costoso, llegando en ocasiones a durar varios días, por lo tanto, es fundamental elegir el sistema de procesamiento que se encargará de realizar el procesamiento.

La principal ventaja que presentan las GPU (*Graphical Processor Unit*) frente a las CPU (*Central Processing Unit*) es que las primeras son tarjetas gráficas especializadas en renderizar gráficos 2D y 3D junto con una CPU, acelerando el procesamiento de las imágenes enormemente en comparación con las unidades CPU tradicionales.

Debido a los enormes avances sufridos por el aprendizaje automático y los requisitos de las redes neuronales actuales, se desarrollaron las TPU (*Tensor Processing Unit*) los cuales son sistemas de procesamiento empleados en inteligencia artificial capaces de aumentar enormemente la velocidad de procesamiento de las GPU's.

A lo largo del trabajo, se emplearán unas u otras unidades de procesamiento, ya sea usando dispositivos físicos, o empleando unidades de procesamiento en la nube para acelerar el proceso de entrenamiento de los modelos.

2.4. Casos de estudio similares:

Definidas las ideas claves que formarán parte de la obtención de las imágenes necesarias para entrenar el clasificador, y decidido que se empleará el API *Object Detection* de TensorFlow para dicho entrenamiento, el siguiente paso necesario es realizar un trabajo de investigación con el fin de encontrar proyectos que hayan usado dicha API para entender los pasos a realizar y detectar los posibles problemas que pudiesen aparecer durante la realización de este.

Son múltiples los proyectos encontrados en la literatura que emplean esta API de TensorFlow para la detección de objetos en casos particulares. Algunos ejemplos de interés encontrados son los siguientes:

- **Detector de cartas:**

  Uno de los ejemplos encontrados durante la investigación, es este proyecto realizado por EdjeElectronics (EdjeElectronics, 21/02/2019) en el cual, muestra los diferentes pasos necesarios para crear un modelo capaz de clasificar las diferentes cartas de una baraja en tiempo real usando la API de TensorFlow.

ILUSTRACIÓN 16: PROYECTOS OBJECT DETECTION: CARTAS

- **Detección de objetos cotidianos:**

  El modelo diseñado por BalA VenkatesH, similar al mostrado anteriormente, emplea la API de TensorFlow para detectar objetos cotidianos del día a día como tijeras, ratones de teclado, etc. En su artículo muestra los pasos necesarios para lograrlo (VenkatesH, 2018).

- **Detector "Not Santa":**

  En el proyecto llevado a cabo por Rosebrock, A. (Rosebrock, 2017), diseño un modelo de detección de "Papa Noel" para clasificar entre personas disfrazadas como este personaje y otras que no lo eran. Este modelo tiene de especial que es instalado en una Raspberry Pi, y servirá de ayuda en el proyecto para saber qué pasos son necesarios seguir y cuáles son los problemas que se pueden encontrar durante su implementación en dicho dispositivo.

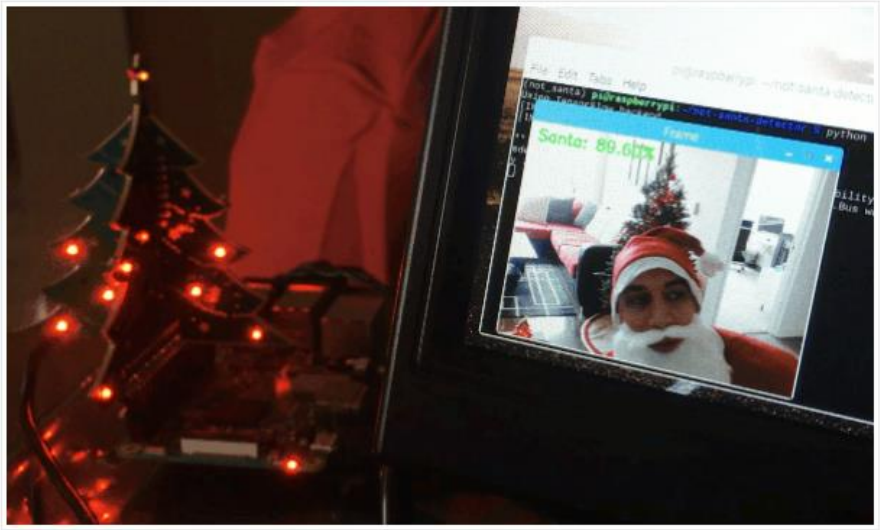

ILUSTRACIÓN 17: PROYECTOS OBJECT DETECTION: NOT SANTA

- **Análisis de objetos en una carretera:**

  Otro proyecto interesante es el presentado por Olafenwa, M. (Olafenwa, 2018) en el cual muestra como instalar y ejecutar un clasificador capaz de reconocer en imágenes las diferentes personas y vehículos presentes en imágenes de carreteras.

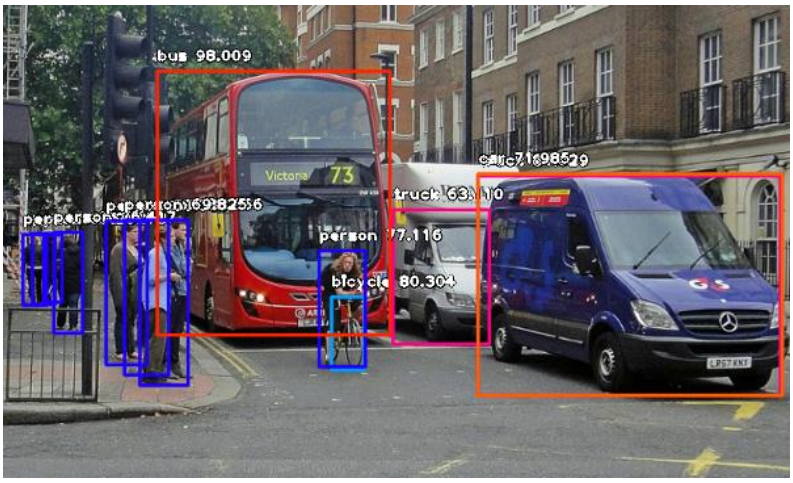

ILUSTRACIÓN 18: PROYECTOS OBJECT DETECTION: ELEMENTOS DE UNA CARRETERA

- **Control de videojuegos:**

  Una aplicación muy interesante encontrada es la diseñada por Chintan Trivedi (Trivedi, 21/02/2019) en la cual, diseña un modelo capaz de jugar al famoso juego Counter Strike a través de movimientos realizados por sí mismo a través de la webcam.

ILUSTRACIÓN 19: PROYECTOS OBJECT DETECTION: COUNTER STRIKE

- **Análisis de partido de futbol con TensorFlow:**

  Una aplicación interesante encontrada es la aplicación de la API para detectar diferentes elementos en videos de un partido de futbol. Esta aplicación, diseñada por Priya Dwivedi (Dwivedi, 21/02/2019) es capaz de diferenciar a los jugadores de ambos equipos (Austrialia y Peru) a través de un video del partido.

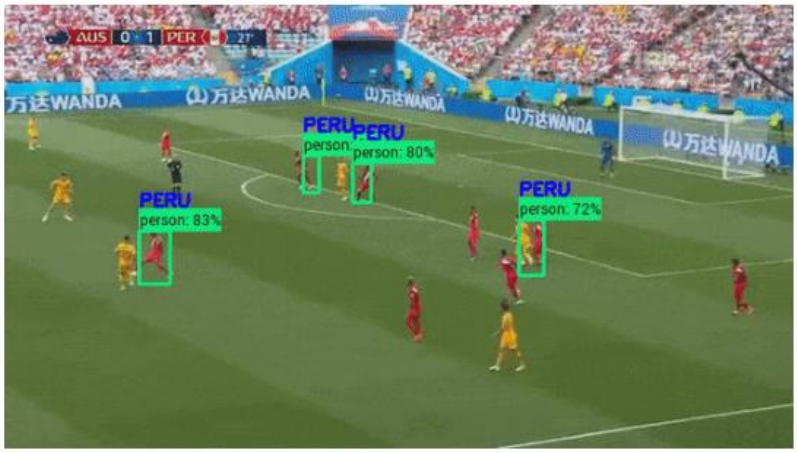

ILUSTRACIÓN 20: PROYECTOS OBJECT DETECTION: PARTIDO DE FÚTBOL

- **Detectando a Pikachu en Andorid**:

  Nombrando más ejemplos, encontramos este modelo diseñado por De Dios Santos, J., en el cual diseña un reconocedor capaz de detectar a Pikachu, el famoso Pokémon de los videojuegos de Niantic, en las diferentes imágenes que se le pasan al modelo empleando TensorFlow. La innovación que propone es la inclusión de este modelo en un dispositivo Android, permitiendo así disponer de un reconocedor en el teléfono móvil (De Dios Santos, 2018).

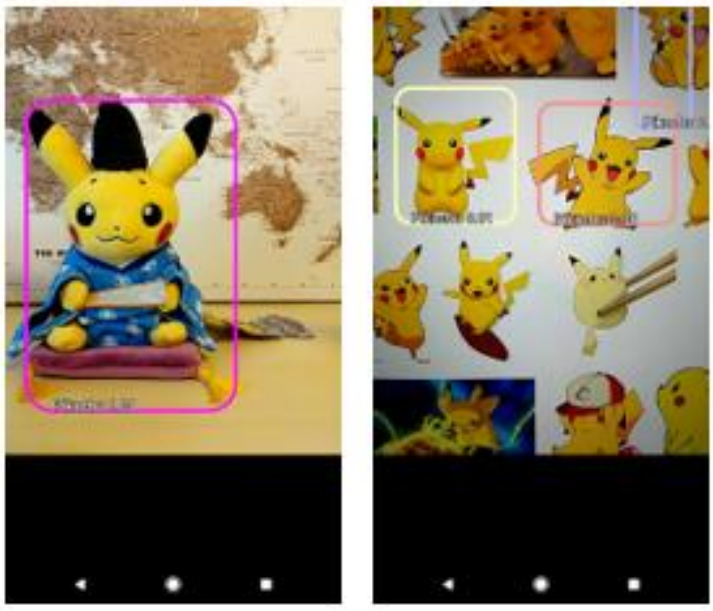

ILUSTRACIÓN 21: PROYECTOS OBJECT DETECTION: PIKACHU EN ANDROID

- **Jugar al Quidditch:**

  Otro de los proyectos interesantes encontrados por la web y con el que se dará por finalizado este apartado, es un modelo diseñado por Bharath Raj, “How to play Quidditch using the TensorFlow Object Detection API” (Raj, B., 21/02/2019) en el cual a través de diferentes frames de las películas de Harry Potter en las cuales aparecen los personajes jugando al Quidditch, diseña un modelo capaz de detectar las diferentes pelotas que forman parte del juego. Una vez terminado, pasando un fragmento de la película al modelo, es capaz de detectar cada una de las pelotas que van apareciendo.

ILUSTRACIÓN 22: PROYECTOS OBJECT DETECTION: QUIDDITCH

3. Objetivos concretos y metodología de trabajo:

El objetivo de este trabajo final de máster consiste en desarrollar una metodología de reconocimiento de placas que será integrada en una prueba piloto de un sistema de detección y clasificación en tiempo real a través de visión artificial, integrando el desarrollo software del algoritmo de detección de objetos en dispositivos IoT con sensores visuales, abaratando el precio de los sistemas actuales disponibles en el mercado.

El trabajo se podría dividir en 5 apartados:

1. Desplegar las herramientas existentes de Object Recognition y adaptarla al reconocimiento de placas de luminarias.
2. Usar técnicas de Deep Learning con aumentación de datos y transferencia de aprendizaje.
3. Incluir dispositivos IoT que acceda a la información para crear un entorno visual inteligente dentro de la factoría.
4. Diseñar una solución empleando sistemas de bajo coste para disminuir gastos.
5. Evaluar el sistema de reconocimiento de placas a través de las imágenes recolectadas en el entorno real de la industria.

El sistema de detección y clasificación propuesto será diseñado para la empresa Signify Manufacturing Valladolid, implementado en una zona específica de la cadena mecánica de su área de pintura. La finalidad es detectar e identificar el tipo de luminaria, o parte de la misma, que se encuentra colgada sobre las diferentes perchas que circulan a lo largo de dicha cadena, siendo capaz el sistema de obtener dicha información antes de que dicha percha alcance la cabina de pintado.

Este proyecto se encuentra enfocado dentro de uno de los puntos principales del *"Hossing Plan"* de la compañía, el cual, entre muchos otros objetivos, incluye un proceso de digitalización global de la empresa, entendiendo por digitalización diferentes conceptos como la automatización de los procesos, digitalización de la información manual actual, y al registro o trazabilidad completa de los diferentes procesos realizados en las diferentes factorías de la empresa.

Los pasos e ideas desarrolladas durante el desarrollo del TFM se basan en las ideas propuestas en el trabajo de investigación *"Straightforward Recognition of Daily Objects in Smart Environments from Wearable Vision Sensor"* (Medina Quero, 2018) desarrollado por Medina Quero, Javier (tutor del TFM), en el cual *"*muestran un método novedoso que facilita

el reconocimiento visual de objetos cotidianos en entornos inteligentes a partir de sensores de visión wearables". Dicha metodología se ha integrado en el reconocimiento de este proyecto, que abarca un ámbito de aplicación y evaluación mayor con dispositivos IoT creando en la factoría un espacio visual inteligente para el reconocimiento de piezas industriales.

En dicho trabajo de investigación, ante la gran cantidad de datos etiquetados requeridos para la obtención de modelos de reconocimiento de imágenes válidos, proponen una solución para la cual, a partir de unas pocas imágenes desde diferentes ángulos de un objeto, y de un video del fondo por el que estará dicho objeto, ser capaz de crear suficientes datos de entrada etiquetados para aportar al modelo, (aplicando transformaciones a las imágenes del objeto sin fondo y superponiéndolas sobre el video del fondo) y que dicho modelo sea capaz de detectar los objetos indicados de una manera eficaz.

A continuación, se muestra una imagen de dicho trabajo en el que se puede observar las ideas claves mostradas:

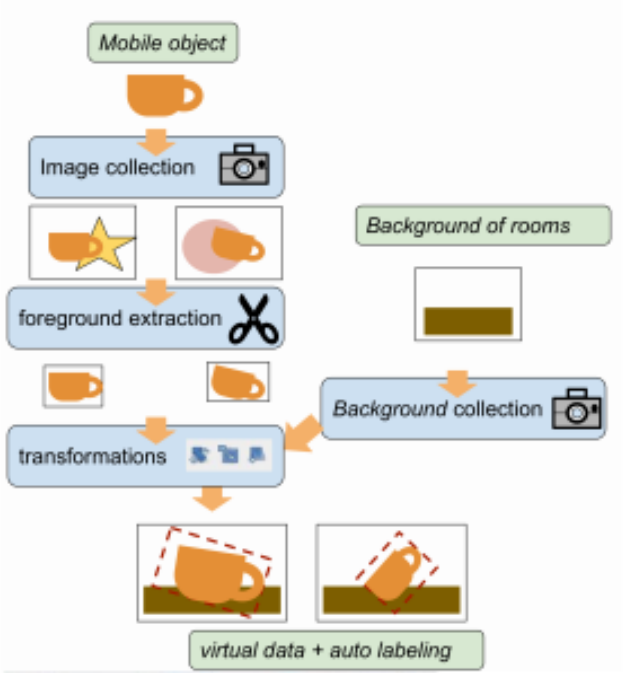

ILUSTRACIÓN 23: GENERACIÓN DE IMÁGENES DE ENTRADA A PARTIR DE IMÁGENES INDIVIDUALES

Como se muestra en la ilustración, el objetivo es el de generar una cantidad de imágenes de los objetos deseados etiquetados, la cual sea suficiente extensa para poder entrenar un clasificador de imágenes. Para ello se siguen los siguientes pasos:

1. Obtención de unas pocas imágenes de los objetos a detectar sobre un fondo de un color liso diferente al del objeto con un dispositivo móvil o una cámara de fotos normal.
2. Extracción del contorno de las imágenes mediante algún software específico con la finalidad de crear imágenes de los objetos sin fondo o sobre un fondo con un valor de los pixeles específico.
3. Realizar una serie de transformaciones sobre las imágenes para generar a partir de una imagen individual, multitud de imágenes del mismo objeto, pero con ligeras modificaciones. Esta será una de las partes más importantes a tener en cuenta, puesto que, para la generación de imágenes aumentadas a partir de las pocas imágenes de entrada, es fundamental realizar sobre estas imágenes todas las modificaciones posibles para dotar al clasificador de la mayor variedad posible de imágenes diferentes, e impedir que sea capaz de memorizar datos concretos. Entre las diversas transformaciones realizadas se encuentran:
    - Traslación.
    - Rotación.
    - “Flip”.
    - Escalado.
    - Recortes.
4. Captación de imágenes del fondo sobre el que estarán los objetos.
5. Superponer cada una de las imágenes generadas con las transformaciones sobre cada una de las imágenes del fondo.
6. Obtención de una cantidad suficiente de imágenes de los objetos etiquetadas sobre el fondo, con diferentes transformaciones, que ayudarán al futuro modelo a tener suficientes imágenes diferentes como para no memorizar cada objeto y ser capaz de generalizar.

El trabajo de investigación para la selección de las imágenes a realizar, así como el desarrollo de las herramientas (código en Python) empleadas para la generación de imágenes, y las herramientas en código Java y Arduino para la implantación en el entorno de producción, son ideas originales propuestas para la correcta implementación en el entorno industrial de este TFM, a partir de las diferentes librerías existentes ya en la web.

4. Desarrollo y resultados:

4.1. Identificación de entorno de implementación:

Para empezar con el diseño de la implementación del reconocedor de piezas, el primer paso consiste en analizar el proceso de pintado de piezas de la fábrica para seleccionar la mejor ubicación física del mismo, analizando las ventajas que aportará el sistema situándolo en una ubicación o en otra y el entorno de la instalación, detectando las posibles fuentes de ruido futuras que pudiesen entorpecer el sistema de reconocimiento.

Por lo tanto, la ubicación deseada se seleccionó en función de los siguientes aspectos:

a) Ubicación con respecto a la instalación de pintura:

La instalación de pintura se encuentra formada por una cadena mecánica situada a 3 metros de altura con respecto del suelo, por la cual las piezas, colgadas de unas perchas, van circulando desde la estación de carga a la estación de descarga con una frecuencia de paso entre perchas de 45 segundos en un trayecto total de 2 horas (suponiendo que no hay incidentes). Durante el recorrido, las piezas situadas en cada una de las perchas van pasando por una serie de instalaciones en las cual reciben los tratamientos necesarios para su correcto pintado, entre los que se incluyen, por orden:

1. Estación de carga
2. Túnel de tratamiento
3. Horno de secado
4. Cabinas de pintura
5. Polimerizado
6. Estación de descarga

Uno de los objetivos principales del reconocedor propuesto en este TFM es el de aportar información a la instalación de pintura de qué tipo de pieza se encuentra cargada en la cadena lo antes posible. Para ello, es fundamental que el sistema sea capaz de reconocer las piezas instantes después de ser cargadas en la cadena, para que, al llegar a las diferentes instalaciones, se disponga de la información suficiente.

Por esta razón, la ubicación del sistema reconocedor de piezas tiene que estar situada entre la estación de carga de piezas y la primera instalación de tratamientos.

En segundo lugar, es muy importante buscar una zona de la cadena situada entre la estación de carga y la primera instalación de tratamientos, en la cual las perchas no se acumulen una detrás de otra cuando haya sobreproducción o congestionamiento de la cadena, lo cual provocaría problemas en la detección de las piezas, mezclando las piezas situadas en una percha con las siguientes.

Tras analizar el funcionamiento de la cadena, se observó que los únicos puntos que cumplen con las especificaciones son las curvas de las cadenas, las cuales disponen de un sensor a la entrada y otro a la salida de la curva, las cuales permiten el paso de las perchas única y exclusivamente cuando no hay otra en el interior. Por esta razón, se eligió la primera curva disponible tras la estación de carga, situada a menos de un minuto de dicha estación.

En la siguiente imagen se muestra el *layout* de la instalación, con el camino seguido por las piezas indicado por las diferentes flechas y la ubicación seleccionada para ubicar el sistema de detección de piezas:

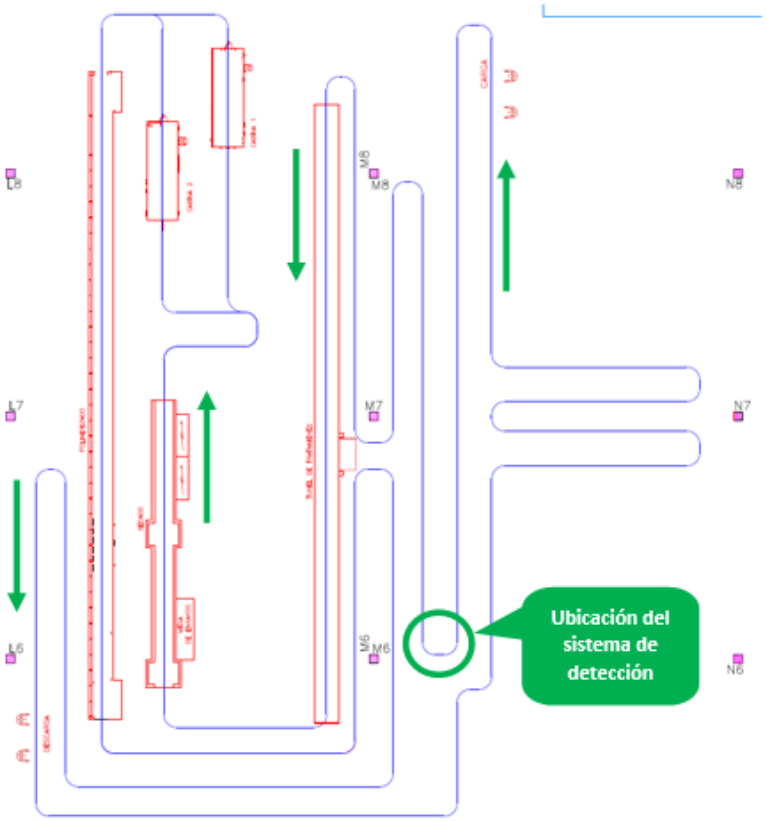

ILUSTRACIÓN 24: *LAYOUT* DEL ÁREA DE PINTURA

b) Eliminación del ruido de fondo:

Decidida la ubicación del reconocedor de piezas, el siguiente paso consiste en analizar el fondo o "*background*" de la zona seleccionada. En este caso, la posibilidad de acumulación de piezas se ha eliminado gracias a la decisión de tomar una curva como zona de toma de imágenes. Sin embargo, esto no elimina del todo la posibilidad de ruido causado por piezas situadas detrás del objetivo a analizar, porque como se observa en el layout de la instalación, en la parte de detrás de la zona seleccionada se encuentra otra zona de la cadena, por la que circulan las piezas pintadas tras salir del horno de secado.

Para eliminar esta posible fuente de ruido sin afectar al proceso de la instalación, se opta por instalar una lona blanca entre la curva seleccionada para la detección y la zona de la cadena que queda justo detrás, para que el fondo pase a ser lo más estable posible durante la detección.

A continuación, se observa la situación de la instalación antes y después de la instalación de la lona blanca:

ILUSTRACIÓN 25: "BACKGROUND" ANTES Y DESPUÉS DE LAS MODIFICACIONES

4.2. Selección y categorización de piezas a detectar:

Una vez decidida la ubicación física del reconocedor de imágenes, el siguiente paso consiste en identificar todas las posibles piezas que pueden pintarse en la instalación. Para ello, se realizan los siguientes pasos:

a) Identificación de todas las piezas posibles:

A partir de todas las órdenes de producción desarrolladas en la fábrica entre los meses de junio y octubre, se han obtenido aquellas luminarias que tienen una fase de pintura previa a su montaje en las líneas de ensamblado final.

Para facilitar la obtención de toda esta casuística, se desarrolló un programa en java, que accediendo a la base de datos de la fábrica en la cual se almacena la información de todas las ordenes de producción, es capaz de identificar solo aquellos 12NC de materiales con marcas de pintura, relacionándolos con la familia a la que corresponden obteniendo un listado de 447 registros diferentes clasificados en función de su tipo (Apartado A1.).

Gráficamente, se puede observar el porcentaje de cada uno de los tipos de piezas con respecto al total:

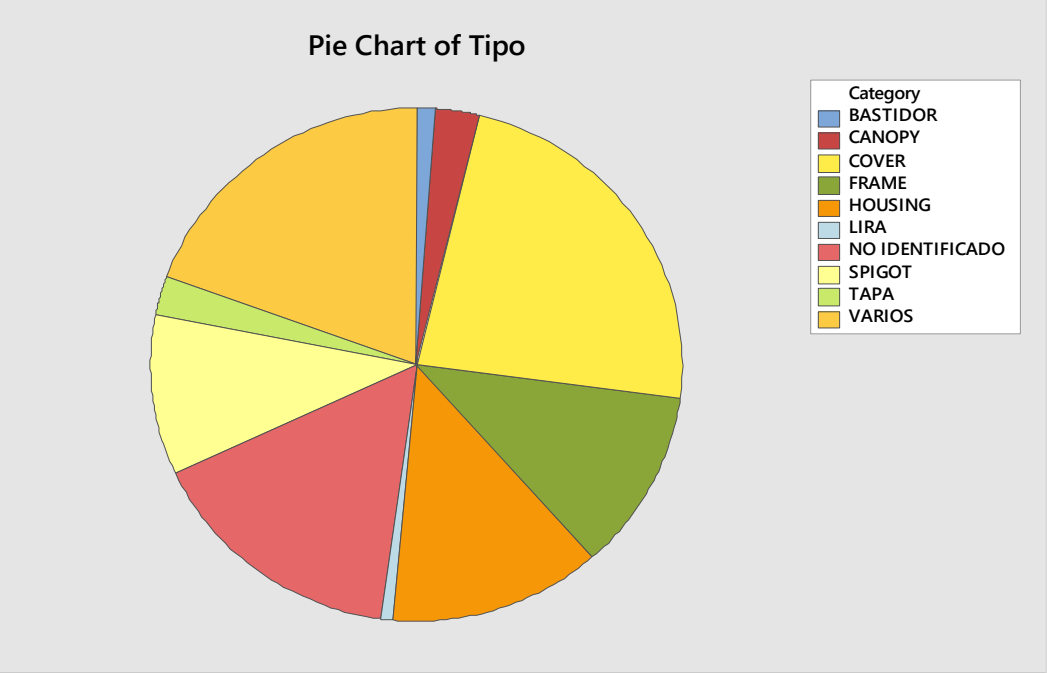

ILUSTRACIÓN 26: PIE CHART DE LOS DIFERENTES TIPOS DE ELEMENTOS INICIALES

b) Unificación por familia y tipo:

Con todos los datos disponibles, se procedió a unificar por familia y tipo de pieza, para realizar un posterior análisis y elegir entre todas las piezas disponibles, aquellas más habituales.

Para analizarlo de una forma más visual, se obtuvo un “Pie chart” de todos los tipos de piezas por luminaria, formando un total de **45 luminarias diferentes**:

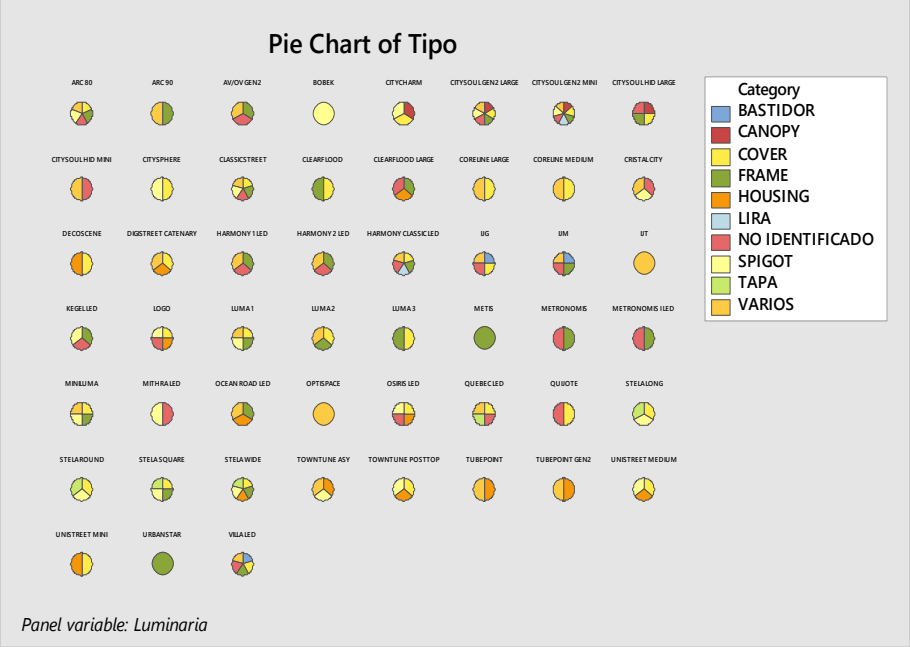

ILUSTRACIÓN 27: PIE CHART DE LOS TIPOS DE ELEMENTOS POR FAMILIA

Con todos los datos agrupados por familias y tipos, se obtuvo un listado con la cantidad de piezas fabricadas durante ese periodo de tiempo, para realizar la selección de piezas finales a analizar en la prueba piloto, ocupando el mayor porcentaje posible de piezas, y obteniendo también el nombre final que recibirá cada tipo de pieza, formado por la siguiente nomenclatura:

“NOMBRE DE FAMILIA” + “ - “ + “TIPO DE PIEZA”

c) Selección en función del tipo de pieza y de la producción:

Una vez se dispone de todos los posibles materiales identificados y renombrados, el siguiente paso es seleccionar que tipos de piezas entran dentro del proyecto inicial formando parte de la prueba piloto que engloba este trabajo, y cuáles son los que quedan fuera del "scope" inicial.

Para ello, en primer lugar, se optó por realizar una división inicial en función del tipo de material, independiente de la familia a la que pertenecen. Los aspectos a tener en cuenta para la selección en primer lugar es el número de piezas fabricadas por tipo. Para ello, a partir de todos los datos, se realiza un Pareto para ver qué tipo de material es el más fabricado:

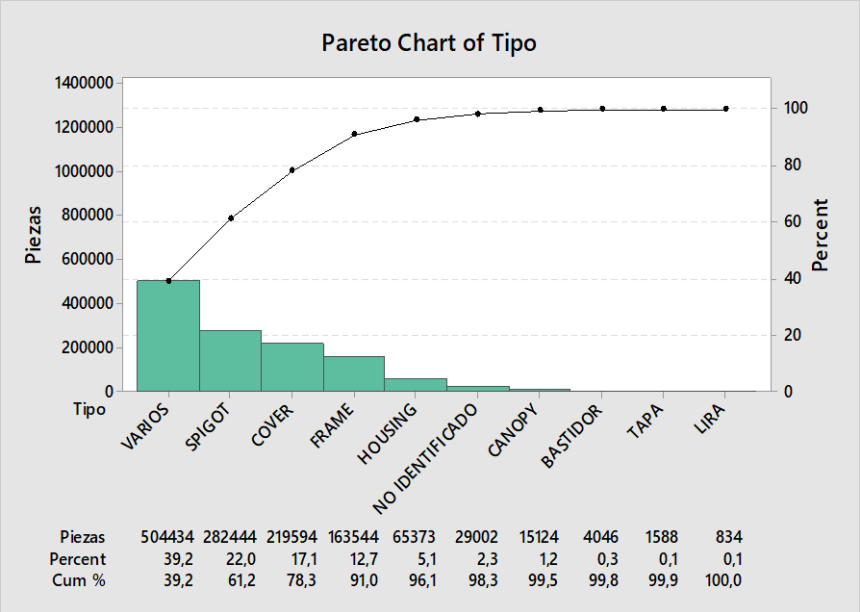

ILUSTRACIÓN 28: PARETO DE LA CANTIDAD DE UNIDADES FABRICADAS POR TIPO

A partir del Pareto anterior, se observa cómo el **90% de las piezas fabricadas** están formadas por los tipos:

1. Varios: elementos **pequeños** que forman parte de las luminarias entre los que se incluyen clips, arandelas, brackets, mascaras, bandejas…
2. Spigot: elementos de tamaño **pequeño** entre los que se incluyen todos los spigots de las luminarias
3. Cover: elementos de tamaño **medio/grande** entre los que se incluyen los cuerpos de las luminarias.

4. Frame: elementos de tamaño **medio/grande** que junto a los cover y los housing forman el cuerpo de las luminarias.
5. Housing: elementos de tamaño **medio/grande** similares a los cover para luminarias específicas.
6. Canopy/tapa: elementos de tamaño **medio/grande** similares a los frame para luminarias específicas.
7. Bastidor: elementos de tamaño **medio/grande** similar a los cover para luminarias específicas.
8. Lira: elemento de tamaño **medio** que sirve de accesorio similar a los spigots.

A partir de lo mostrado anteriormente, se decidió seleccionar todos aquellos tipos de elementos que cumplen con las siguientes características:

1. Elementos definitorios de las luminarias.
2. Elementos de tamaño medio/grande.
3. Elementos con poca variabilidad dentro de las familias.

Por ello, se hace una primera selección de los siguientes elementos, puesto que los materiales como accesorios o liras, al ser similares entre familias, no permiten la identificación sencilla de las luminarias:

- Cover y Housing.
- Frame, Canopy y Tapa.

Una vez se dispone de esa selección en función del tipo de pieza, el siguiente paso es seleccionar aquellas combinaciones familia-tipo que son más fabricadas, para obtener la mayor recompensa posible del sistema al implementarlo.

Con las **38 combinaciones** más fabricadas de los 78 totales, se cubre **el 95% de las combinaciones** más fabricadas.

Realizando esta división, se obtienen 38 tipos diferentes de combinaciones, de un total de 24 tipos de familias diferentes, incluyendo 4 tipos de piezas diferentes, como se muestra en los siguientes gráficos:

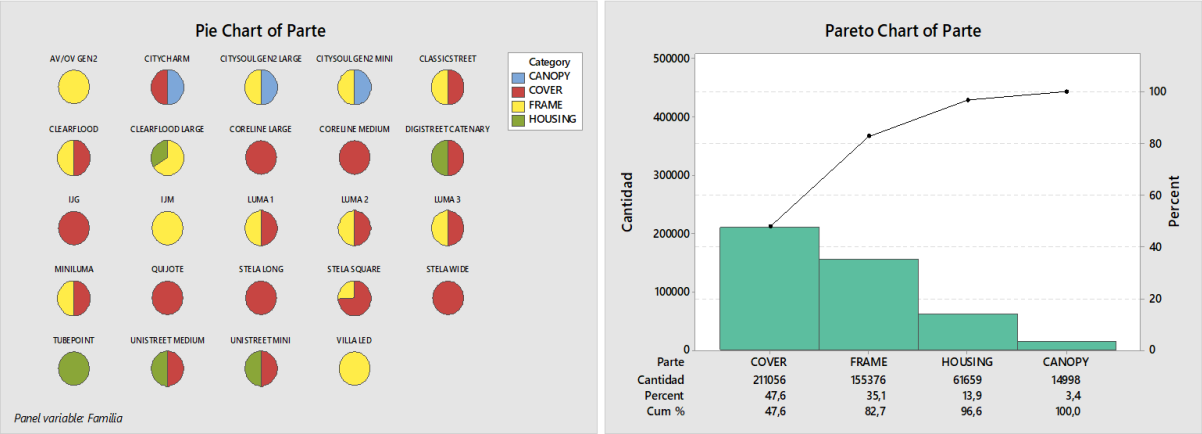

ILUSTRACIÓN 29: PIE CHART Y PARETO DE LOS DATOS TRAS LA SEGUNDA SELECCIÓN

d) Unificación en función de la apariencia física y listado final:

Con todas estas divisiones realizadas, la última parte de este apartado de diseño es el de analizada cada una de las piezas seleccionadas, y detectar si hay grandes similitudes entre unas combinaciones y otras, para reducir al máximo posible el ruido que puedan generar las imágenes para la prueba piloto.

Para la primera versión del sistema que se desarrollará durante este trabajo, se realizará una unificación de las luminarias LUMA 1, 2 y 3, en una sola categoría denominada LUMA, y otra denominada MINILUMA.

Al igual que pasaba en la luma, la familia UNISTREET tiene un caso similar. En este caso, dispone de 2 versiones de la luminaria (MINI y MEDIUM), cuya única diferencia son sus proporciones físicas.

Por lo tanto, como se hizo en la LUMA, se define una única familia denominada UNISTREET, y será después de forma computacional donde se tratará de diferenciar entre las 2 versiones de la luminaria.

El resultado final de la selección quedaría como se muestra a continuación:

TABLA 1: SELECCIÓN FINAL DE LOS DATOS A IMPLEMENTAR

Familia	Parte	Cantidad	12NCs diferentes	Descripción	ID
LUMA	COVER	57026	12	LUMA _COVER	12
LUMA	FRAME	56966	5	LUMA_FRAME	13
CORELINE MEDIUM	COVER	34642	1	CORELINE MEDIUM_COVER	9
CLEARFLOOD	FRAME	32273	6	CLEARFLOOD_FRAME	7
MINILUMA	COVER	25601	4	MINILUMA_COVER	14
MINILUMA	FRAME	25552	2	MINILUMA_FRAME	15
TUBEPOINT	HOUSING	23342	8	TUBEPOINT_HOUSING	18
UNISTREET	HOUSING	23248	16	UNISTREET_HOUSING	19
UNISTREET	COVER	23148	6	UNISTREET_COVER	-
CORELINE LARGE	COVER	21771	1	CORELINE LARGE_COVER	8
CLEARFLOOD	COVER	21549	4	CLEARFLOOD_COVER	6
CITYSOUL GEN2	FRAME	11559	2	CITYSOUL GEN2_FRAME	3
CITYSOUL GEN2	CANOPY	11437	9	CITYSOUL GEN2_CANOPY	2
CLASSICSTREET	COVER	9692	7	CLASSICSTREET_COVER	4
CLASSICSTREET	FRAME	9692	1	CLASSICSTREET_FRAME	5
AV/OV GEN2	FRAME	7818	2	AVOVGEN2_FRAME	1
STELA SQUARE	COVER	6734	6	STELA SQUARE_COVER	17
STELA LONG	COVER	5137	16	STELA LONG_COVER	16
VILLA LED	FRAME	4848	2	VILLA LED_FRAME	-
IJM	FRAME	4379	1	IJM_FRAME	-
DIGISTREET CATENARY	COVER	4293	2	DIGISTREET CATENARY_COVER	-
DIGISTREET CATENARY	HOUSING	4293	3	DIGISTREET CATENARY_HOUSING	10
CITYCHARM	CANOPY	3561	1	CITYCHARM_CANOPY	-
CITYCHARM	COVER	3561	3	CITYCHARM_COVER	-
STELA WIDE	COVER	3434	2	STELA WIDE_COVER	-
QUIJOTE	COVER	3075	2	QUIJOTE_COVER	-
STELA SQUARE	FRAME	2289	4	STELA SQUARE_FRAME	-
IJG	COVER	2169	2	IJG_COVER	11

Finalmente se obtiene los siguientes datos:

1. 19 familias de luminarias.
2. 4 tipos de elementos diferentes
3. 28 tipos de combinaciones LUMINARIA – TIPO diferentes.

De esa selección final, se reduce el número de piezas a detectar a 19 tipos de combinaciones para simplificar la primera versión del sistema, quedando de ahora en adelante habilitadas aquellas combinaciones con un ID numérico, seleccionadas por su diferente apariencia física con respecto a las demás y por su tamaño. Se encuentran numeradas en orden alfabético

4.3. Desarrollo del sistema de detección:

En este apartado se explicará el desarrollo de la solución propuesto para la obtención del clasificador de imágenes, desde la captación de imágenes iniciales, hasta la obtención del resultado obtenido con el modelo creado.

Antes de empezar con el desarrollo del sistema paso a paso, es necesario definir una serie de herramientas que se emplearán a lo largo del documento para entender en todo momento toda la información:

- **Python:**

  Python, es el lenguaje de programación interpretado que se empleará en el desarrollo de la aplicación de visión artificial. Administrado por la *Python Software Foundation*, es un lenguaje de programación multiplataforma, de código abierto y desarrollado con una filosofía bastante similar a la filosofía de Unix entre los que se incluyen principios como los descritos por Tim Peters en "El Zen de Python" (Peters, 2010).

  Este lenguaje de programación es uno de los más empleados en el área del aprendizaje automático, y por esa razón será el empleado durante el trabajo.

- **TensorFlow:**

  Desarrollado y lanzado por Google en 2005, TensorFlow (Tensorflow, 21/02/2019) es una de las bibliotecas de código abierto más importantes en el área del aprendizaje automático cuya finalidad es la de aportar un sistema capaz de entrenar redes neuronales.

  Gracias a la capacidad de este "*framework"* de permitir emplear redes neuronales, y a ser el software más empleado a nivel mundial en el reconocimiento de imágenes, se hace fundamental su uso para el desarrollo del trabajo.

- **TensorBoard:**

  Con el fin de facilitar el entrenamiento de las redes neuronales, haciendo más visual e intuitivo el proceso, TensorFlow, entre sus múltiples utilidades, ha diseñado TensorBoard. Como menciona en la página oficial (Tensorflow, Tensorboard: Visualizing Learning, 21/02/2019), TensorBoard es un "conjunto de herramientas de visualización, con las cuales se pueden visualizar los gráficos de TensorFlow, trazar métricas

cuantitativas sobre la ejecución de su gráfico y mostrar datos adicionales como imágenes que pasan a través de él".

Esta herramienta será de gran utilidad durante el desarrollo del trabajo, sobre todo en el apartado de entrenamiento del clasificador de imágenes para visualizar el estado del entrenamiento y evaluar las diferentes métricas de error de una manera sencilla.

a) Captación de imágenes a categorizar:

Una vez seleccionadas y categorizadas todas las piezas fabricadas que se desean detectar, se procede a realizar unas 20-25 capturas de cada luminaria. Los requerimientos de las imágenes son los siguientes:

1. Imágenes frontales de las piezas desde diferentes ángulos.
2. Fondo de las imágenes uniforme de un color diferente al de la pieza a detectar.
3. Formato .jpg o .png.

Para ello, durante un periodo de 2 días, se realizó dicha captación de imágenes a nivel de planta en la propia fábrica, obteniendo para cada luminaria definida en el apartado anterior el número de imágenes deseadas. A continuación, se muestran varios ejemplos:

ILUSTRACIÓN 30: IMÁGENES FRONTALES DE 4 REFERENCIAS

Una vez almacenadas todas, se creó una estructuración de carpetas para tenerlas organizadas nombrando a cada carpeta con la siguiente codificación:

"FAMILIA" + "_" + "PARTE"

Ejemplos:

- AVOVGEN2_FRAME
- CITYSOULGEN2_CANOPY
- CLASSICSTREET_COVER

Con la estructuración de carpetas fijada, se almacenó dentro de cada carpeta las imágenes correspondientes. Para tenerlas mejor identificadas aún, y no usar el nombre por defecto, se desarrolló un código de Python (Apartado A2.) cuya finalidad es leer todas las imágenes que se encuentran en la misma dirección que dicho código, y renombrarlas con la siguiente codificación:

"NOMBRE DE LA CARPETA" + "_" + "NÚMEROS CORRELATIVOS DE 4 DÍGITOS" + ".jpg"

b) Eliminación del fondo de las imágenes seleccionadas:

Con todas las imágenes captadas, almacenadas y nombradas correctamente, el siguiente objetivo es el de eliminar el fondo de las imágenes, dejando imágenes de cada una de las piezas con su forma física, sobre un fondo trasparente o de un color de pixeles fijado correctamente.

Para realizarlo, el primer paso consistió en diseñar un algoritmo en Python, basado en la API GrabCut (Amro, 21/02/2019) explicada en el apartado estado del arte. Para adaptarlo al proyecto, se realizaron una serie de modificaciones:

1. **Selección de imágenes a eliminar el fondo:**

Para agilizar el proceso de eliminación de fondo de todas las imágenes, el primer paso consistió en realizar un algoritmo capaz de detectar todas las imágenes presentes en el directorio del archivo actual con formato ".jpg".

2. **Disminución de las imágenes:**

El siguiente objetivo es el de reducir todas las imágenes a un tamaño del 20% con respecto del tamaño de entrada. Para ello se emplea la librería *resize()* de OpenCV.

3. **Llamada recursiva al programa de eliminación del fondo:**

Una vez disminuidas todas las imágenes, se recorre cada una de ellas de 1 en 1, llamando al api GrabCut creada a partir de las modificaciones introducidas. En el apartado de estado del arte se explica más al detalle las otras alternativas disponibles.

4. Eliminación del fondo:

El objetivo de este apartado es realizar dicha eliminación con el mínimo esfuerzo y en el menor tiempo posible. Para ello se emplea el siguiente algoritmo:

- Lectura de la imagen de entrada al programa, añadida al llamar al script externamente.
- Creación de 2 ventanas de interfaz con el usuario, una con la imagen de entrada, y otra con la salida a generar
- Reconocimiento de las teclas introducidas por el usuario para generar las funciones correspondientes, entre las que se incluyen:
    - 'f': Seleccionar área del fondo seguro
    - 'p': Seleccionar área de la pieza segura
    - '2': Seleccionar áreas de posible fondo
    - '3': Seleccionar áreas de posibles piezas
    - 'n': Actualizar en la salida la segmentación de las áreas marcadas en la entrada
    - 'r': Resetear la salida para empezar de cero con la segmentación
    - 's': Guardar la imagen de salida con la segmentación última seleccionada eliminando todas aquellas filas o columnas de pixeles que no tengan partes de las piezas.
    - 'q': Salir de la segmentación de esa imagen.

Con todas estas funcionalidades creadas, el proceso de segmentación es el siguiente (imágenes adjuntadas del proceso realizado sobre una imagen LUMA_FRAME_0003.png):

1. Esperar a que la siguiente imagen se abra:
2. Marcar el área de la pieza manteniendo pulsado el botón derecho del ratón:
3. Pulsar 'p' y seleccionar varios píxeles pertenecientes a la pieza:
4. Pulsar 'f' y seleccionar varios píxeles del fondo de la imagen:
5. Pulsar 'n' para actualizar la imagen de salida y comprobar que la imagen queda segmentada correctamente:
6. Comprobar si la imagen queda como se esperaba. Si no están correctamente separadas la pieza del fondo, repetir los pasos 2, 3 y 4.

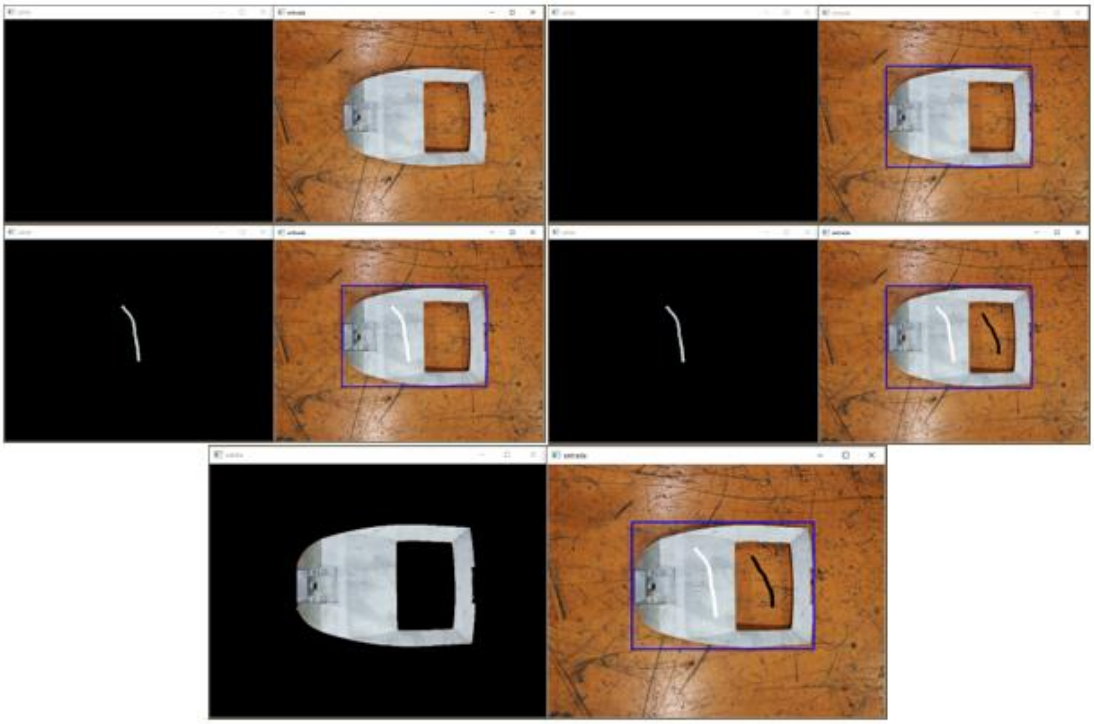

ILUSTRACIÓN 31: ELIMINACIÓN DEL FONDO

7. Pulsar ‘s’ para guardar la imagen en la carpeta “SalidaBordes” del directorio actual, creada automáticamente por el programa:

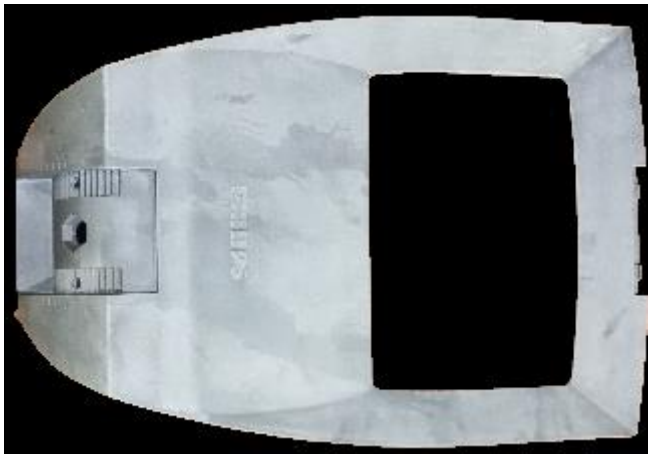

ILUSTRACIÓN 32: ELIMINACIÓN DE FONDO - PASO 7: GUARDAR IMAGEN FINAL

8. Pulsar ‘q’ y volver al paso 1 en caso de quedar más imágenes por analizar.

Finalizado este proceso, ya se disponen de imágenes todas las piezas a identificar sobre un fondo negro con valores de píxel RGB(0,0,0), reducida a un tamaño cercano a 300x220 pixeles y nombradas según la nomenclatura definida para una fácil identificación.

c) Captación del entorno:

Una vez capturadas las piezas, y eliminado su fondo, el siguiente paso consiste en capturar el entorno por donde estarán las piezas que se quieren detectar. Para ello, una vez instalada la tela blanca que servirá de fondo, se realizó un video de aproximadamente 20 segundos de duración con diferentes orientaciones, del cual posteriormente se sacarán 8-10 frames mediante programación, que servirán para formar las imágenes para entrenar el modelo.

A continuación, se muestran algunos de los frames obtenidos a partir del vídeo grabado:

ILUSTRACIÓN 33: FRAMES DEL FONDO OBTENIDOS A PARTIR DEL VÍDEO GRABADO

d) Generación de imágenes de entrenamiento y de test:

Con todas las imágenes de las piezas a detectar con fondo unificado e identificadas, y un video del fondo, el siguiente paso consiste en generar las diferentes imágenes de entrenamiento y de test. Para ello, se creó un programa que se encarga de forma automática de superponer cada una de las imágenes de las piezas sobre diferentes frames del video de entrada, así como de encuadrar cada pieza en un “Bounding Box” sacando sus coordenadas .

Los pasos seguidos por el programa son los siguientes (información complementaria en Apartado A3, A4 y A5):

1. **Creación de los DataFrames de Pandas donde se almacena la información de los archivos .csv de salida:**

En primer lugar, se inicializan los DataFrames “df_salida_train” y “df_salida_test” empleados para almacenar los valores de los archivos .csv de salida, cuyas columnas son:

- **FILENAME**: nombre de la imagen de salida
- **WIDTH**: tamaño de ancho de la imagen de salida
- **HEIGHT**: tamaño de alto de la imagen de salida
- **CLASS**: clase a la que pertenece.
- **XMIN**: valor en x de la esquina superior izquierda del “Bounding Box” de la pieza con respecto a la imagen total.
- **YMIN**: valor de la y de la esquina superior izquierda del “Bounding Box” de la pieza con respecto a la imagen total.
- **XMAX**: valor en x de la esquina inferior derecha del “Bounding Box” de la pieza con respecto a la imagen total.
- **YMAX**: valor de la y de la esquina inferior derecha del “Bounding Box” de la pieza con respecto a la imagen total.

2. **Creación de la carpeta de salida:**

En segundo lugar, el programa genera las carpetas de salida necesarias para almacenar las imágenes generadas, entre las que se incluyen:

- Carpeta “test”: con las imágenes empleadas para testear el algoritmo generado más adelante tras el entrenamiento.
- Carpeta “train”: con las imágenes empleadas para entrenar el algoritmo.
- Carpeta “Video”: con todos los frames del fondo generados a partir del archivo .mp4.
- Carpeta “Bounding”: con todas las imágenes generadas, identificadas por el nombre de entrada de la imagen de la pieza, y con un rectángulo o “Bounding box” alrededor de la pieza.

3. **Obtención de todos los archivos existentes en el directorio actual:**

El siguiente paso consiste en identificar el directorio actual y obtener todos los archivos presentes en dicho directorio, entre los que se tienen que incluir:

- Todas las imágenes con fondo negro generadas anteriormente en formato .png.
- El video del fondo en formato .mp4.

4. **Comprobación de la existencia del video del fondo en formato .mp4 y generación de frames:**

Una vez se disponen de los archivos del directorio actual, el programa identifica si existe un archivo con extensión .mp4 y en el caso de existir, llama a la función:

SacarImagenesVideo(nombre, parámetro)

El objetivo de esta función es el de capturar el video de entrada a través de la función "VideoCapture" de OpenCV, e ir generando diferentes frames del video según la va recorriendo según la frecuencia definida a través de la variable parámetro.

ILUSTRACIÓN 34: FRAME GENERADO A PARTIR DEL VIDEO DE ENTRADA

Una vez generados los diferentes frames a partir del video del fondo, estas imágenes se almacenan en la carpeta "Video" con la siguiente codificación:

"F" + "NUMERO DEL FRAME CON 3 DÍGITOS" + ".png"

Algún ejemplo de imágenes guardadas para un parámetro de entrada 30:

- F000.png
- F030.png
- F060.png

5. Generación de imágenes sintéticas con aumentación de datos:

El siguiente paso realizado por el programa, consiste en ir recorriendo una a una todas las imágenes encontradas en el directorio actual con formato .png, y tras sacar el nombre de la clase de la pieza a partir de su nombre de archivo definido en el apartado a) "Captación de imágenes a categorizar", se realizan 10 transformaciones entre las que se incluyen:

- Primera transformación: imagen original de entrada, almacenada con el sufijo "_0".
- Segunda transformación: imagen rotada 90º, almacenada con el sufijo "_90".
- Tercera transformación: imagen rotada 180º, almacenada con el sufijo "_180".
- Cuarta transformación: imagen rotada 270º, almacenada con el sufijo "_270".
- Resto de transformaciones: se activa la funcionalidad "*random*", para que a la hora de superponer se realicen las transformaciones aleatorias correspondientes en la función de superposición de la imagen sobre el fondo. Se almacena con el sufijo "_RANDOM" + "ITERACIÓN CORRESPONDIENTE DESDE 4 A 9".

Después de realizar las diferentes transformaciones indicadas, el programa llama a la función encargada de superponer la imagen de la pieza sobre el fondo, eliminando todos aquellos pixeles de valor RGB(0,0,0).

La función es la siguiente:

```
SuperponerPiezaEnFondo(nombre_imagen, imagen_pieza_ columns, clase,
                       ImagenGuardo, randomIt)
```

donde:

- nombre_imagen: es el nombre de la imagen original con el sufijo de la transformación anterior.
- imagen_pieza: es la imagen original con la transformación indicada anteriormente
- columns: nombres de las columnas del DataFrame de salida.
- clase: corresponde a la clase de la imagen definida a partir del nombre de la imagen de entrada.
- ImagenGuardo: es el número que se le dará a la imagen a la hora de guardarla, empezando desde 0.
- randomIt: corresponde al parámetro que indica con un 1 o un 0 si hay que realizar alguna transformación random o no a la imagen respectivamente.

Con todos estos parámetros, al llamar a la función, esta se encarga de realizar los siguientes pasos:

- Coger todos los frames de las imágenes del fondo.
  - Recorrer una a una para ir superponiendo cada imagen sobre todas ellas.
  - Aplicar las rotaciones random correspondientes o no, entre las que se incluyen:
    i. Rotar aleatoriamente la imagen entre -45º y 45º.
    ii. Eliminar las filas y columnas con píxeles completamente negros.
    iii. Flip de la imagen de acuerdo con la salida de la siguiente función:

```
flip = choice(range(1,10)) % 2
```

    iv. Reducir la imagen en un valor aleatorio seleccionando un valor entre 0.6 y 1.0 (variable determinada en experimentación) con la siguiente función (esto lo realiza tanto si es modo “RANDOM” como si no lo es):

```
round((choice(np.arange(0.6, 1.0, 0.05))),1)
```

  - Superponer la imagen de la pieza en el centro de la imagen del fondo correspondiente con las transformaciones indicadas a través de la función:

```
PonerImagenEnCentro(fondo, pieza)
```

donde,

- Fondo: es la imagen del fondo.
- Pieza: es la imagen de la pieza.

Esta función realiza los siguientes pasos:

- Genera una imagen de pixeles negros exactamente igual que la imagen del fondo de tamaño y saca su centro.
- Saca el centro de la imagen de la pieza.
- En caso de que el centro de la pieza sea menor que el centro de la imagen del fondo calcula las esquinas superior e inferior tanto de la izquierda como de la derecha:

```
sup_izq = center_black[0] - center_pieza[0], center_black[1] - center_pieza[1]
inf_izq = center_black[0] + center_pieza[0], center_black[1] - center_pieza[1]
sup_der = center_black[0] - center_pieza[0], center_black[1] + center_pieza[1]
inf_der = center_black[0] + center_pieza[0], center_black[1] + center_pieza[1]
```

- En caso de ser compatibles la imagen del fondo y la de la pieza (por tema de dimensiones) se recorre pixel a pixel la imagen del fondo almacenando el pixel del fondo siempre y cuando el pixel recorrido en ese momento no se encuentre dentro

de los pixeles de la imagen a superponer y en el caso de que si sea un pixel donde va la pieza, siempre y cuando no sea un pixel de valor RGB(0, 0, 0).

- Terminada la superposición, se genera otra imagen idéntica, pero con el “Bounding Box” de la pieza dibujada en forma rectangular sobre la imagen en color verde, empleando la función “rectangle()” de OpenCV.

ILUSTRACIÓN 35: IMÁGENES CON BOUNDING BOX

- Una vez realizada la superposición se devuelven 4 parámetros:
  - Imagen con la pieza superpuesta.
  - Imagen con la pieza superpuesta y el “Bounding Box” dibujado.
  - Coordenadas de la esquina superior izquierda.
  - Coordenadas de la esquina inferior derecha.

Finalizado el trabajo de esta función, la función “SuperponerPiezaEnFondo” termina su trabajo realizando los siguientes pasos:

- Decidir si es una imagen de entrenamiento o de test a través de la siguiente función:

```
choice(range(1,300)) % 50
```

- Almacenar los datos necesarios en el DataFrame de entrenamiento o de test.
- Generar la imagen correspondiente tanto en la carpeta “train” o “test” dependiendo de la decisión anterior, como en la carpeta “Bounding” siguiendo la siguiente nomenclatura:

"NOMBRE DE LA IMAGEN ORIGINAL" + "_" + "_TRANSFORMACIÓN INICIAL" + "_" + "IMAGEN DE FONDO" + ".png"

ILUSTRACIÓN 36: IMÁGENES DE SALIDA DEL PROGRAMA DE SUPERPOSICIÓN DE IMÁGENES

Algún ejemplo para la imagen "LUMA_FRAME_0003.png" usada anteriormente sería:

- LUMA_FRAME_0003_0_F000.png
- LUMA_FRAME_0003_0_F030.png
- LUMA_FRAME_003_90_F000.png
- LUMA_FRAME_003_RANDOM4_F030.png

6. Determinación del Bounding-box de los archivos .csv de salida:

Una vez realizado el proceso de forma automática con todas las imágenes de entrada, se generan los archivos .csv de salida con el formato mencionado anteriormente. A continuación, se muestran las salidas para la ejecución del programa con la imagen que contempla el informe:

- **Train.csv:**

TABLA 2: REGISTROS DEL ARCHIVO TRAIN.CSV DE SALIDA

filename	width	height	class	xmin	ymin	xmax	ymax
train/0.png	270	480	LUMA_FRAME	86.75	206.5	183.25	273.5
train/1.png	270	480	LUMA_FRAME	86.75	206.5	183.25	273.5
train/2.png	270	480	LUMA_FRAME	54.75	184.0	215.25	296.0
train/3.png	270	480	LUMA_FRAME	62.75	189.5	207.25	290.5
train/4.png	270	480	LUMA_FRAME	62.75	189.5	207.25	290.5
train/5.png	270	480	LUMA_FRAME	86.75	206.5	183.25	273.5
train/6.png	270	480	LUMA_FRAME	62.75	189.5	207.25	290.5
train/7.png	270	480	LUMA_FRAME	86.75	206.5	183.25	273.5
train/8.png	270	480	LUMA_FRAME	86.75	206.5	183.25	273.5
train/9.png	270	480	LUMA_FRAME	62.75	189.5	207.25	290.5
train/10.png	270	480	LUMA_FRAME	62.75	189.5	207.25	290.5
train/11.png	270	480	LUMA_FRAME	62.75	189.5	207.25	290.5
train/12.png	270	480	LUMA_FRAME	70.75	195.25	199.25	284.75
train/13.png	270	480	LUMA_FRAME	70.75	195.25	199.25	284.75
train/14.png	270	480	LUMA_FRAME	70.75	195.25	199.25	284.75
train/15.png	270	480	LUMA_FRAME	54.75	184.0	215.25	296.0
train/16.png	270	480	LUMA_FRAME	70.75	195.25	199.25	284.75
train/17.png	270	480	LUMA_FRAME	78.75	200.75	191.25	279.25
train/18.png	270	480	LUMA_FRAME	54.75	184.0	215.25	296.0
…	…	…	…	…	…	…	…
train/187.png	270	480	LUMA_FRAME	74.75	190.0	195.25	290.0

- **Test.csv:**

TABLA 3: REGISTROS PARA EL ARCHIVO TEST.CSV DE SALIDA

filename	width	height	class	xmin	ymin	xmax	ymax
test/44.png	270	480	LUMA_FRAME	54.75	184.0	215.25	296.0
test/69.png	270	480	LUMA_FRAME	84.5	167.75	185.5	312.25
test/175.png	270	480	LUMA_FRAME	53.0	181.75	217.0	298.25

e) Compresión de archivos TFRecord para el entrenamiento:

Una vez se dispone de todas las imágenes superpuestas y clasificadas en entrenamiento y test, el siguiente paso consiste en generar a partir de los archivos .csv los archivos correspondientes en formato ".record" que se emplearán en la API "*Object Detection*" de TensorFlow como entradas y salidas.

Para ello, se emplea el algoritmo de Python “generate_tfrecord.py” provisto por el tutor y se le realizan unas pequeñas modificaciones para que de forma automática sepa leer tanto el archivo train.csv como el archivo test.csv (Apartado A5). El programa realiza las siguientes operaciones:

1. Se genera el archivo record de salida.
2. Se define la dirección de salida.
3. Se toman los archivos de entrada del csv.
4. Se coge la columna llamada “filename”.
5. Se recorren los datos cogidos y se va escribiendo en la salida. Para ello, previamente es necesario definir en la clase interna “class_text_to_int” el valor correspondiente para cada clase. Para este proyecto sería:

```
def class_text_to_int(row_label):
                        if row_label == 'AVOVGEN2_FRAME':
                            return 1
                        elif row_label == 'CITYSOULGEN2_CANOPY':
                            return 2
                        elif row_label == 'CITYSOULGEN2_FRAME':
                            return 3
                        elif row_label == 'CLASSICSTREET_COVER':
                            return 4
                        elif row_label == 'CLASSICSTREET_FRAME':
                            return 5
                        elif row_label == 'CLEARFLOOD_COVER':
                            return 6
                        elif row_label == 'CLEARFLOOD_FRAME':
                            return 7
                        elif row_label == 'CORELINELARGE_COVER':
                            return 8
                        elif row_label == 'CORELINEMEDIUM_COVER':
                            return 9
                        elif row_label == 'DIGISTREETCATENARY_HOUSING':
                            return 10
                        elif row_label == 'IJG_COVER':
                            return 11
                        elif row_label == 'LUMA_COVER':
                            return 12
                        elif row_label == 'LUMA_FRAME':
                            return 13
                        elif row_label == 'MINILUMA_COVER':
                            return 14
                        elif row_label == 'MINILUMA_FRAME':
                            return 15
                        elif row_label == 'STELALONG_COVER':
                            return 16
                        elif row_label == 'STELASQUARE_COVER':
                            return 17
                        elif row_label == 'TUBEPOINT_HOUSING':
                            return 18
                        elif row_label == 'UNISTREET_HOUSING':
                            return 19
                        else:
                            return 0
```

6. Se crea el archivo TFRecord en la ubicación de salida.

A continuación, se muestra un ejemplo de la salida de la ejecución del programa para generar los archivos indicados (Apartado A5):

```
(base) C:\Users\user\Desktop\SalidaBordes>python generate_tfrecord.py
train/0.png
train/1.png
train/10.png
train/100.png
train/101.png
train/102.png
...
train/98.png
train/99.png
Successfully created the TFRecords: train.record
test/175.png
test/44.png
test/69.png
Successfully created the TFRecords: test.record
```

Es importante destacar que para que el programa funcione correctamente, es imprescindible disponer de la carpeta "utils" de la API de TensorFlow "Object Detection" en el mismo directorio para que funcionen determinadas funciones.

f) Selección, ajuste y despliegue del modelo de entrenamiento:

Con todas las imágenes de entrada disponibles, la API Object Detection de TensorFlow instalada, y los archivos .record creados, el siguiente paso es definir los parámetros necesarios del modelo para realizar el entrenamiento.

Para ello, es necesario tener los siguientes archivos:

1. **Object-detection.pbtxt:**

Este archivo, es el encargado de definir la relación entre las diferentes clases y el ID correspondiente de cada una de ellas. Para definirlo, la numeración se basó en la definida en la tabla 1. Se puede ver el archivo definido en los anexos (Apartado A6).

2. **Obj_inception.config:**

Este archivo, tiene como finalidad definir todos los parámetros del modelo con el cual se va a realizar el entrenamiento de la red. Para ello, se toma como ejemplo un archivo aportado por el tutor, al que se le realizan unas pequeñas modificaciones (Apartado A7):

- **Numero de clases**: el primer valor a definir es el número de clases a predecir que tendrá el modelo. Para ello, basándonos en el archivo object-detection.pbtxt.

Para nuestro modelo, el número total de clases es **19**.

```
faster_rcnn {
  num_classes: 19
  image_resizer {
    keep_aspect_ratio_resizer {
      min_dimension: 600
      max_dimension: 1024
    }
  }
```

- **Modelo predefinido a modificar**: el siguiente paso, es seleccionar un modelo predefinido de la librería TensorFlow, el cual será el usado para mantener todos sus parámetros por defecto, y solo modificar los clasificadores que nos interesen, que en este caso son nuestras clases de imágenes. Para ello, se emplea el modelo Inception v2. Se decide usar este modelo debido a sus grandes ventajas en cuanto a Velocidad vs precisión (explicado en el estado del arte):

faster_rcnn_inception_v2

Este modelo debe estar ubicado en una dirección del computador, y hay que indicarlo en el archivo obj_inception.config en el apartado "fine_tune_checkpoint:" de la siguiente forma:

fine_tune_checkpoint: "{DIRECCIÓN A TENSORFLOW}/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"

Este modelo está pre-entrenado, el cual dispone de muchos patrones visuales existentes, y lo que se hará es modificar solo las últimas capas para adaptarlo a el ejemplo concreto que engloba este proyecto, siendo así el aprendizaje mucho más rápido y eficaz.

- **Ubicación del archivo .record de entrenamiento y de test**: es necesario definir tanto para los parámetros de entrenamiento como de evaluación la dirección del ordenador en la que se encuentran los archivos .record generados en el apartado anterior, así como de la dirección del archivo object-detection.pbtxt definido anteriormente.

```
train_input_reader: {
  tf_record_input_reader {
    input_path: "{DIRECCION ORDENADOR}/train.record"
  }
  label_map_path: "{DIRECCION ORDENADOR}/object-detection.pbtxt"
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "{DIRECCION ORDENADOR}/test.record"
  }
  label_map_path: "{DIRECCION ORDENADOR}/object-detection.pbtxt"
  shuffle: false
  num_readers: 1
}
```

A mayores, este archivo dispone de una serie de parámetros predefinidos para realizar "data augmentation" de los datos de entrada y así disponer de muchos más registros para el entrenamiento.

g) Primer caso de estudio de la red neuronal (6 clases):

1. Selección de datos iniciales:

Para realizar un primer entrenamiento, se seleccionan solo 6 clases sin gran parecido físico entre ellas con la finalidad de comprobar que el modelo funciona correctamente y detectar posibles fallos o mejoras a incluir en el modelo final. Para ello, se escogen las siguientes 6 categorías:

TABLA 4: DATOS INICIALES: 6 CLASES

Familia	Parte	Etiqueta	ID
AVOV Gen 2	Frame	AVOVGEN2_FRAME	1
Citysoul Gen 2	Canopy	CITYSOULGEN2_CANOPY	2
Coreline Large	Cover	CORELINELARGE_COVER	3
Digistreet Catenary	Housing	DIGISTREETCATENARY_HOUSING	4
Luma	Frame	LUMA_FRAME	5
Unistreet	Housing	UNISTREET_HOUSING	6

ILUSTRACIÓN 37: IMÁGENES PARA EL PRIMER CASO DE ESTUDIO

Con esta selección hecha, se modifica y ejecuta el código generador de los archivos TFRecord, así como los archivos Object-detection.pbtxt y Obj_inception.config para realizar el entrenamiento.

Los datos a tener en cuenta antes de la ejecución del entrenamiento, relativos a la creación de las imágenes de entrenamiento y de validación son los siguientes:

- Entradas del entrenamiento: Se han empleado como entradas 21922 imágenes en formato .png generadas a partir de:
  i. 118 imágenes originales de las luminarias similares a las de la ilustración anterior.
  ii. 19 *frames* tomados de un video del fondo de unos 25 segundos de duración.
  iii. Tiempo de generación: 57 segundos por cada una de las 118 imágenes originales: total 1 hora y 56 minutos.

- Imágenes de validación: 385 imágenes en formato .png generadas durante el proceso anterior.

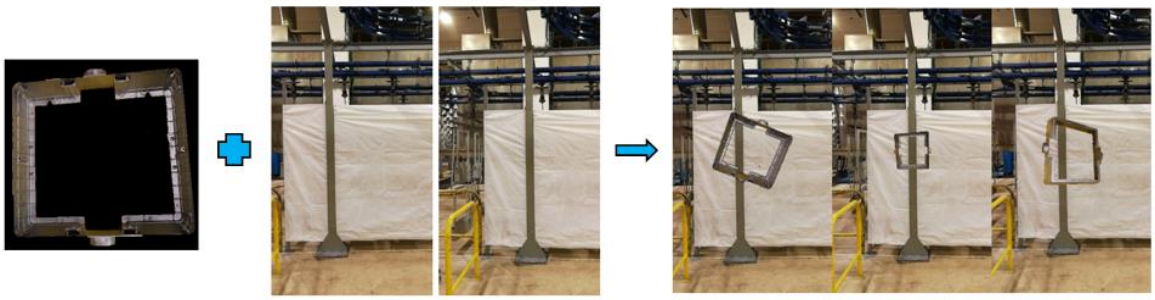

ILUSTRACIÓN 38: GENERACIÓN DE IMÁGENES DE ENTRENAMIENTO Y TEST

2. Ejecución del entrenamiento:

Con todos los archivos preparados correctamente, se ejecuta el entrenamiento de la red neuronal. Para ello, se emplea la función “train.py” del API Object Detection de TensorFlow, siguiendo los siguientes pasos:

- Abrir un terminal “cmd”.
- Navegar hasta la carpeta models/research:

```
cd {DIRECCION DE TENSORFLOW}/models/research
```

- Ejecutar el siguiente comando para modificar el PATH y que apunte hasta la carpeta research:

```
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

- Crear la carpeta “training” y copiar dentro del archivo obj_inception.config:

```
mkdir training
cd training
cp {DIRECCION AL ARCHIVO}/obj_inception.config training/
```

- Ejecutar el entrenamiento:

```
python3 object_detection/legacy/train.py
--logtostderr
--train_dir=training/
--pipeline_config_path=training/obj_inception.config
```

Realizados estos pasos, tras cargar todos los mensajes y avisos correspondientes, el sistema empieza a entrenar como se muestra a continuación (Apartado A8):

```
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
...
I1224 12:21:36.284593 140159613925120 tf_logging.py:159] global_step/sec: 0
INFO:tensorflow:Recording summary at step 0.
I1224 12:21:41.200845 140159605532416 tf_logging.py:115] Recording summary at step
0.
INFO:tensorflow:global step 1: loss = 1.9927 (12.242 sec/step)
I1224 12:21:46.488050 140161595688768 tf_logging.py:115] global step 1: loss =
1.9927 (12.242 sec/step)
INFO:tensorflow:global step 2: loss = 2.0384 (3.275 sec/step)
I1224 12:21:49.875236 140161595688768 tf_logging.py:115] global step 2: loss =
2.0384 (3.275 sec/step)
INFO:tensorflow:global step 3: loss = 1.9582 (3.761 sec/step)
I1224 12:21:53.636834 140161595688768 tf_logging.py:115] global step 3: loss =
1.9582 (3.761 sec/step)
INFO:tensorflow:global step 4: loss = 1.7209 (3.594 sec/step)
I1224 12:21:57.231038 140161595688768 tf_logging.py:115] global step 4: loss =
1.7209 (3.594 sec/step)
...
```

A través de la ventana de comandos se puede ir observando los resultados obtenidos para la función de perdida en cada una de las etapas correspondientes según se va realizando el entrenamiento de la red.

El entrenamiento se ha realizado en un ordenador con las siguientes características:

- Procesador Intel Core i5 de 7ª generación, con CPU 2.50 GHz 2.70 GHz.
- S.O. Ubuntu (Linux).
- Memoria RAM interna de 8 GB.

3. Visualización del entrenamiento con TensorBoard:

Para visualizar de una forma más gráfica el resultado del entrenamiento, se emplea la herramienta TensorBoard. Para emplearla, se abre una nueva ventana de comandos y se navega hasta la dirección de la carpeta models/research:

```
cd {DIRECCION DE TENSORFLOW}/models/research
```

Una vez en esa dirección, se ejecuta la llamada a TensorBoard, apuntando a la carpeta en la cual se están almacenando los datos del entrenamiento:

```
tensorboard --logdir=training
```

Si todo funciona correctamente, la ventana de comandos devolverá una dirección web apuntando al ordenador local en la cual se puede observar el proceso del entrenamiento. Simplemente accediendo a un buscador web del ordenador y pegando la dirección indicada se observará de manera gráfica el entrenamiento gracias a TensorBoard.

Antes de entrar al detalle del entrenamiento, es importante explicar las métricas que se mostrarán en las siguientes gráficas:

- *RPN Loss:*
    a) *Localization_loss:* esta métrica hace referencia a la pérdida del “Bounding Box” o cuadro delimitador de objetos para la RPN (“*Region Proposal Network”).*
    b) *Objectness_loss:* esta métrica indica la pérdida del clasificador que clasifica si un cuadro delimitador es un objeto de interés o fondo.

- *Box Classifier Loss:*
    a) *Classification_loss:* esta métrica indica la pérdida para la clasificación de objetos detectados en las diferentes clases.
    b) *Localization_loss:* perdida de localización o la pérdida del regresor de la "Bounding Box" o cuadro delimitador.

- *Total Loss*:
    a) *Total_loss:* esta métrica es una suma del resto de métricas.
    b) *Clone_loss:* está métrica tiene sentido al trabajar en entornos distribuidos. Como en nuestro caso todo el entrenamiento se realiza en una CPU, no tiene sentido puesto que será la misma que la anterior.

En este caso, se muestran los resultados obtenidos tras 8 horas de entrenamiento, observando las diferentes funciones de pérdida para el primer entrenamiento y observando que la red está entrenando al obtener cada vez valores más bajos en las diferentes métricas:

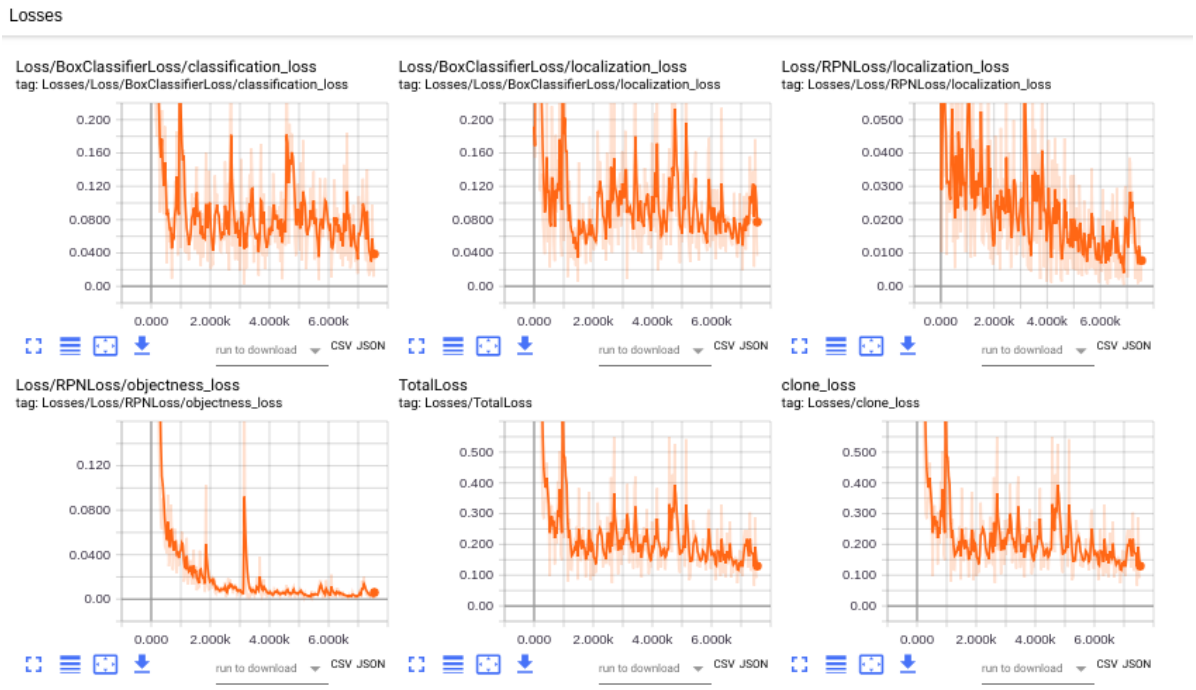

ILUSTRACIÓN 39: FUNCIONES DE PÉRDIDA DEL PRIMER CASO DE ESTUDIO: 6 CLASES

Tras 23675 iteraciones, y un total de 24 horas de entrenamiento se obtienen las siguientes gráficas:

- ***RPN Loss*:** se observan que los valores obtenidos para estás métricas son muy bajos, rondando tanto en localización como en clasificación valores entre 0,01 y 0,02.

ILUSTRACIÓN 40: RPN LOSS PRIMER CASO DE ESTUDIO: 6 CLASES

- ***Box Classifier Loss*:** en este caso se observa como los valores tanto de la localización como de la clasificación de los objetos se mantienen entre valores de 0.01 y 0.1, indicando tanto que hay clases que será capaz de clasificar mejor que otras, como que habrá determinados objetos del fondo que confundirá en algunas imágenes con piezas reales.

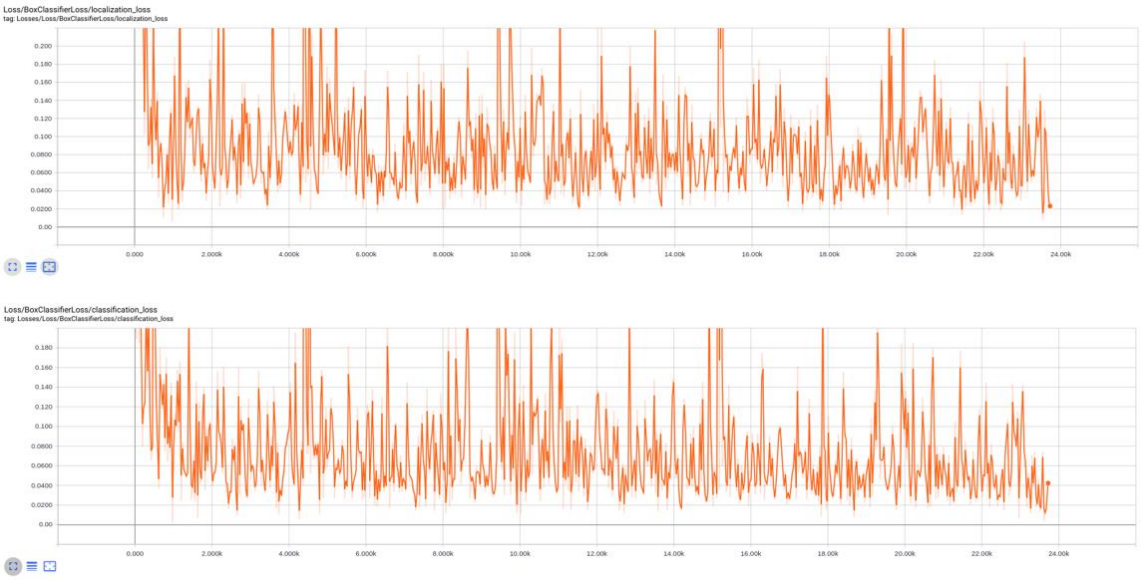

ILUSTRACIÓN 41: BOX CLASSIFICATION LOSS PRIMER CASO DE ESTUDIO: 6 CLASES

- ***Total Loss*:** haciendo el sumatorio de todas las gráficas anteriores se obtiene el resultado de pérdida total, que como se observa se mantiene en valores entre 0.1 y 0.3.

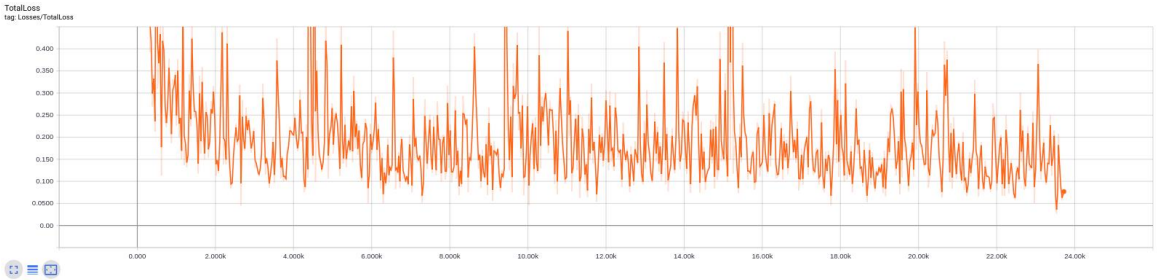

ILUSTRACIÓN 42: TOTAL LOSS PRIMER CASO DE ESTUDIO: 6 CLASES

4. Generación del grafico de inferencia a partir del último modelo entrenado:

Durante el entrenamiento, cada cierta iteración se va almacenando los modelos correspondientes a cada una de las iteraciones. El siguiente paso es crear un modelo fijo a partir de una de esas iteraciones almacenadas. Para ello, se emplea la función "export_inference_graph.py" de la librería Object Detection, a través de los siguientes comandos:

```
cd {DIRECCION DE TENSORFLOW}/models/research

export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

python3 object_detection/export_inference_graph.py
      --input_type image_tensor
      --pipeline_config_path={DIRECCION DEL
      ARCHIVO}/obj_inception.config
      --trained_checkpoint_prefix {DIRECCION DE LA CARPETA
      training}/model.ckpt-{NUMERO DE LA ITERACIÓN A USAR}
      --output_directory {DIRECCION DE LA CARPETA DONDE ALMACENAR}/
```

Ejecutadas las líneas de comandos anteriores, de todo salir correctamente, se obtendrá el modelo a emplear en la carpeta indicada (en el apartado A9 de los anexos se puede ver el resultado de la ejecución realizada para el modelo entrenado seleccionado).

5. Validación del modelo:

- **Ejecución de la validación:**

Para la validación del modelo se analizan las imágenes de validación generadas, así como el código “Object_detection_video.py” adjunto en el apartado A10 de los anexos el cual sigue los siguientes pasos:

- Carga el gráfico de inferencia generado anteriormente, el cual incluye el modelo en TensorFlow entrenado con los diferentes parámetros de la red neuronal.
- Carga el archivo .pbtxt que incluye las diferentes clases a clasificar y sus id’s.
- Busca en la carpeta “sceneA” ubicada en el mismo directorio del programa todas las imágenes a intentar clasificar.
- Genera los porcentajes de clasificación de objetos para cada una de las clases y lo almacena en un archivo pA.tsv.
- Genera en la carpeta “Salida” la imagen con el “*Bounding Box*” correspondiente, mostrando todas aquellas clasificaciones que superen un porcentaje superior al 70%.

Para la ejecución del código es necesario:

- Pegar la carpeta “TFM_model” con el modelo entrenado obtenido del paso de generación del gráfico de inferencia en la dirección TensorFlow/models/research/object_detection.
- Pegar la carpeta “sceneA” con las imágenes de validación en la dirección TensorFlow/models/research/object_detection.
- Pegar el archivo “object-detection.pbtxt” en la dirección TensorFlow/models/research/object_detection.
- Modificar el código “Object_detection_video.py” poniendo en la variable NUM_CLASSES el número de clases a detectar.
- Ejecutar los siguientes comandos en un terminal:

```
cd {Dirección a carpeta Tensorflow}/models/research
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/Slim
cd object_detection/
python3 Object_detection_video.py
```

- **Análisis preliminar de datos:**

Una vez ejecutado el programa con las 385 imágenes de validación generadas en el apartado 1, se obtienen los siguientes resultados (incluido en el archivo Datos6Clases.xlsx adjunto a este informe):

TABLA 5: CLASIFICACIÓN DE LOS DATOS DE VALIDACIÓN: 6 CLASES

ID	CLASE REAL	IMAGENES A CLASIFICAR	VERDADEROS POSITIVOS		FALSOS NEGATIVOS	
1	AVOVGEN2_FRAME	36	36	100%	0	0%
2	CITYSOULGEN2_CANOPY	69	69	100%	0	0%
3	CORELINELARGE_COVER	78	72	92.3%	6	7.7%
4	DIGISTREETCATENARY_HOUSING	68	59	86.76%	9	13.24%
5	LUMA_FRAME	73	45	61.64%	28	38.36%
6	UNISTREET_HOUSING	61	61	100%	0	0%

En la tabla anterior se muestran los resultados obtenidos al intentar clasificar las 385 imágenes de validación generadas. En la tabla se muestra la clase real de la imagen, el número de imágenes de esa clase a clasificar y en las 2 columnas la cantidad de imágenes que han sido clasificadas correctamente o no. Para obtener este valor, se ha tomado de las 6 posibles clases, la clase que obtiene una probabilidad mayor.

Se puede observar como las clases 1, 2 y 6 han sido clasificadas correctamente un 100% de las veces, mientras que la clase 5 ha sido la que peor clasificación a obtenido.

Para analizar esa clase, se muestra la matriz de confusión obtenida:

TABLA 6: MATRIZ DE CONFUSIÓN: 6 CLASES

ID CLASES		1	2	3	4	5	6
1	AVOVGEN2_FRAME	36					
2	CITYSOULGEN2_CANOPY		69				
3	CORELINELARGE_COVER	3		72			3
4	DIGISTREETCATENARY_HOUSING				59		9
5	LUMA_FRAME	6	1			45	21
6	UNISTREET_HOUSING						61

Se puede observar como la clase 6 (UNISTREET_HOUSING) es la que más problemas causa en el resto de las clases, probablemente por ser una de las piezas con menos formas físicas representativas de la misma, al igual que pasa con la clase 1 (AVOVGEN2_FRAME).

Para solucionar este problema, es conveniente observar en los registros en los que el modelo ha clasificado correctamente cada una de las clases, cual es el promedio que ha otorgado a cada una de dichas clases. Para ello, se filtran todos aquellos registros en los que la clasificación ha sido la correcta (342) y se calcula el porcentaje medio otorgado a cada clase, obteniendo la siguiente tabla:

TABLA 7: PORCENTAJES VERDADEROS POSITIVOS: 6 CLASES

	ID CLASES (Real vs Promedio)	1	2	3	4	5	6	Total muestras
1	AVOVGEN2_FRAME	0,9837						36
2	CITYSOULGEN2_CANOPY		0,9831					69
3	CORELINELARGE_COVER			0,9364				72
4	DIGISTREETCATENARY_HOUSING				0,7907			59
5	LUMA_FRAME					0,7461		45
6	UNISTREET_HOUSING						0,9565	61

Como se observa, las clases conflictivas 1 y 6 cuando son clasificadas correctamente tienen porcentajes muy elevados (>95%). Las clases que había sido clasificada erróneamente en mayor medida tiene un porcentaje de clasificación correcto cercano al 75%.

Si observamos el porcentaje medio obtenido en cada una de las imágenes que han sido mal clasificadas (figura siguiente) se observa como cuando por ejemplo una clase 5 (LUMA_FRAME) es clasificada como clase 6 (UNISTREET_HOUSING), ha sido causado porque esa ha sido la clase con mayor porcentaje, sin embargo, el promedio no supera el 61%, muy lejano del 95% que indicaría que es realmente una clase 6.

TABLA 8: PORCENTAJES CLASIFICACIÓN INCORRECTA: 6 CLASES

	ID CLASES (Real vs Promedio)	1	2	3	4	5	6	Total muestras
3	CORELINELARGE_COVER	0,6077		0,3979			0,5043	6
4	DIGISTREETCATENARY_HOUSING				0,5961		0,6905	9
5	LUMA_FRAME	0,3627				0,4139	0,6034	28

Para mejorar la performance sobre la clasificación de las imágenes, se puede determinar que cuando el clasificador tome una imagen de entrada, en función de un umbral aplicado al valor *confidence* de cada clase que haya asignado a cada posible clase, es decir, que el reconocimiento no se determine por el porcentaje mayor y que este indique el tipo de clase que es, sino que posteriormente al análisis de la imagen, se aplique un umbral numérico que optimice qué clase es la correcta y cual no.

Por ejemplo, en el caso de la clase 5 (LUMA_FRAME), las imágenes que han dado problemas son las siguientes (en verde marcada la clase real, y en rojo el porcentaje máximo otorgado por el modelo, lo cual define la clase final):

TABLA 9: CLASIFICACIÓN ERRÓNEA CLASE 5: 6 CLASES

Imagen	AVOVGEN2 FRAME	CITYSOULGEN2 CANOPY	LUMA FRAME	UNISTREET HOUSING	Clasificación errónea
Prueba252	0,0894	0,2132	0,4658	0,5526	UNISTREET_HOUSING
Prueba258	0,1850	0,0938	0,5303	0,6790	UNISTREET_HOUSING
Prueba260	0,4722	0,3152	0,4695	0,5145	UNISTREET_HOUSING
Prueba262	0,3030	0,1966	0,4928	0,5202	UNISTREET_HOUSING
Prueba265	0,2152	0,1516	0,5564	0,6276	UNISTREET_HOUSING
Prueba266	0,5203	0,0371	0,2179	0,6366	UNISTREET_HOUSING
Prueba267	0,3698	0,0788	0,2978	0,7636	UNISTREET_HOUSING
Prueba268	0,2567	0,0506	0,3506	0,6902	UNISTREET_HOUSING
Prueba269	0,5085	0,0290	0,3462	0,4889	AVOVGEN2_FRAME
Prueba271	0,2257	0,1196	0,4254	0,6231	UNISTREET_HOUSING
Prueba272	0,6184	0,0331	0,2967	0,4161	AVOVGEN2_FRAME
Prueba273	0,7065	0,0314	0,4323	0,3946	AVOVGEN2_FRAME
Prueba274	0,2931	0,0702	0,4087	0,7330	UNISTREET_HOUSING
Prueba277	0,6844	0,0599	0,4114	0,4009	AVOVGEN2_FRAME
Prueba280	0,2093	0,3711	0,3733	0,5974	UNISTREET_HOUSING
Prueba282	0,4891	0,0939	0,3683	0,5594	UNISTREET_HOUSING
Prueba284	0,4756	0,0732	0,3969	0,6586	UNISTREET_HOUSING
Prueba286	0,5828	0,1579	0,5131	0,3357	AVOVGEN2_FRAME
Prueba289	0,6041	0,0709	0,4765	0,4181	AVOVGEN2_FRAME
Prueba290	0,2454	0,0379	0,6064	0,6673	UNISTREET_HOUSING
Prueba291	0,2617	0,0662	0,6536	0,6877	UNISTREET_HOUSING
Prueba297	0,1523	0,4981	0,4826	0,3197	CITYSOULGEN2_CANOPY
Prueba300	0,1013	0,0632	0,1987	0,8638	UNISTREET_HOUSING
Prueba312	0,5850	0,1200	0,2403	0,7656	UNISTREET_HOUSING
Prueba313	0,2055	0,0589	0,1830	0,8379	UNISTREET_HOUSING
Prueba315	0,1892	0,1301	0,4496	0,6440	UNISTREET_HOUSING
Prueba318	0,3690	0,0887	0,3237	0,7403	UNISTREET_HOUSING
Prueba322	0,2369	0,0358	0,6217	0,7579	UNISTREET_HOUSING

Se puede observar como en todos los casos en los cuales el sistema ha clasificado mal, ninguno de los porcentajes otorgados está por encima del promedio que indica que una la clase es la correcta. En el caso de la clase 6 (UNISTREET_HOUSING), el mayor porcentaje observado es 86%, lo cual está bastante por debajo del valor que obtendría en promedio si realmente fuese dicha clase (95%).

Al igual, se observa como los porcentajes otorgados para la clase real (LUMA_FRAME) en la mayoría de los casos superan el 40% de porcentaje. A partir de esto se podría fijar que siempre que el modelo otorgue a la clase 5 un porcentaje superior al 40%, independientemente del valor otorgado a el resto de las clases, sea clasificado como clase 5.

Todos estos filtros serán fijados más adelante en el apartado de implantación, pero es bueno analizar las posibles causas en este apartado para continuar o reentrenar el modelo.

- **Análisis de las imágenes de salida:**

Para analizar en profundidad la predicción, se analizan las imágenes de salida con los "Bounding Box".

I. Clase 1: AVOVGEN2_FRAME:

Para esta primera clase, como se puede observar no hay grandes problemas en la clasificación. Los confidence score son cercanos al 100% y solo habría que controlarlo en el caso de que, por tener tan buenos valores, la red pudiese haber aprendido esta clase y afectase en gran medida a otras.

TABLA 10: CLASIFICACIÓN IMÁGENES 6 CLASES: AVOVGEN2_FRAME

Imagen	Clase real	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Máximo
Prueba001	AVOVGEN2 FRAME	0,9947	0,0016	0,0068	0,0022	0,0057	0,0197	AVOVGEN2 FRAME
Prueba014	AVOVGEN2 FRAME	0,9962	0,0019	0,0034	0,0013	0,0079	0,0113	AVOVGEN2 FRAME
Prueba018	AVOVGEN2 FRAME	0,9989	0,0008	0,0019	0,0009	0,0044	0,0093	AVOVGEN2 FRAME
Prueba020	AVOVGEN2 FRAME	0,9965	0,0013	0,0048	0,0009	0,0065	0,0264	AVOVGEN2 FRAME

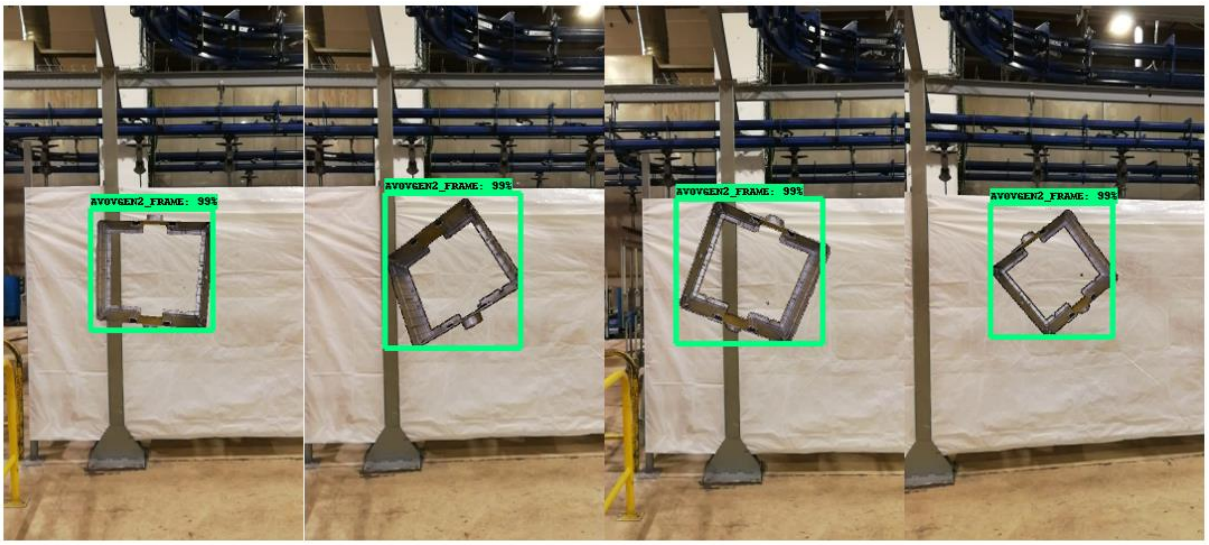

ILUSTRACIÓN 43: IMÁGENES DE VALIDACIÓN 6 CLASES: PRUEBA001, 014, 018 Y 020

II. Clase 2: CITYSOULGEN2_CANOPY:

Esta clase, al igual que la anterior, es otra que tiene valores muy elevados de score. Si hay que tener en cuenta como las imágenes que muestran la parte frontal de la luminaria obtienen valores cercanos al 100%, mientras que aquellas imágenes de la parte trasera de la luminaria se quedan en torno al 90-91%.

Para mejorar el clasificador, en el siguiente entrenamiento habrá que observar si se puede dar el caso de que la luminaria en producción muestre ambas caras o solamente una.

TABLA 11: CLASIFICACIÓN IMÁGENES 6 CLASES: CITYSOULGEN2_CANOPY

Imagen	Clase real	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Máximo
Prueba039	CITYSOULGEN2 CANOPY	0,0162	0,9907	0,0398	0,0994	0,0077	0,7964	CITYSOULGEN2 CANOPY
Prueba052	CITYSOULGEN2 CANOPY	0,1552	0,9972	0,0861	0,0021	0,1689	0,1716	CITYSOULGEN2 CANOPY
Prueba081	CITYSOULGEN2 CANOPY	0,0070	0,9974	0,0083	0,0018	0,0096	0,1781	CITYSOULGEN2 CANOPY
Prueba095	CITYSOULGEN2 CANOPY	0,0210	0,9148	0,0234	0,0270	0,0251	0,2717	CITYSOULGEN2 CANOPY

ILUSTRACIÓN 44: IMÁGENES DE VALIDACIÓN 6 CLASES: PRUEBA039, 052, 081 Y 095

También es conveniente observar como en la imagen 039, se ha clasificado el fondo como UNISTREET_HOUSING (clase 6) con un porcentaje cercano al 80%. Esto habrá que tenerlo en cuenta a la hora de poner los filtros de esta clase, puesto que si cuando de verdad es esa la clase obtiene valores muy elevados (cercanos al 100% como se ha visto y se verá en este apartado) puede no ser un inconveniente.

III. Clase 3: CORELINELARGE_COVER:

Para esta clase se puede observar como la confidence score varía entre las piezas de cara y las piezas desde la parte trasera, en cierta parte debido a la forma característica que muestra la parte delantera en contraposición con la trasera. Cuando la imagen es de parte delantera, se observa una precisión superior al 90%. Sin embargo, al mostrar la parte trasera, esta misma disminuye clasificándolo como otra categoría.

TABLA 12: CLASIFICACIÓN IMÁGENES 6 CLASES: CORELINELARGE_COVER

Imagen	Clase real	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Máximo
Prueba108	CORELINELARGE COVER	0,0191	0,0031	0,9951	0,0007	0,0112	0,0375	CORELINELARGE COVER
Prueba125	CORELINELARGE COVER	0,9004	0,0063	0,1922	0,0047	0,0695	0,1485	AVOVGEN2 FRAME
Prueba139	CORELINELARGE COVER	0,0464	0,0074	0,9244	0,0058	0,0203	0,3062	CORELINELARGE COVER
Prueba179	CORELINELARGE COVER	0,4757	0,0333	0,4725	0,0163	0,1115	0,7056	UNISTREET HOUSING

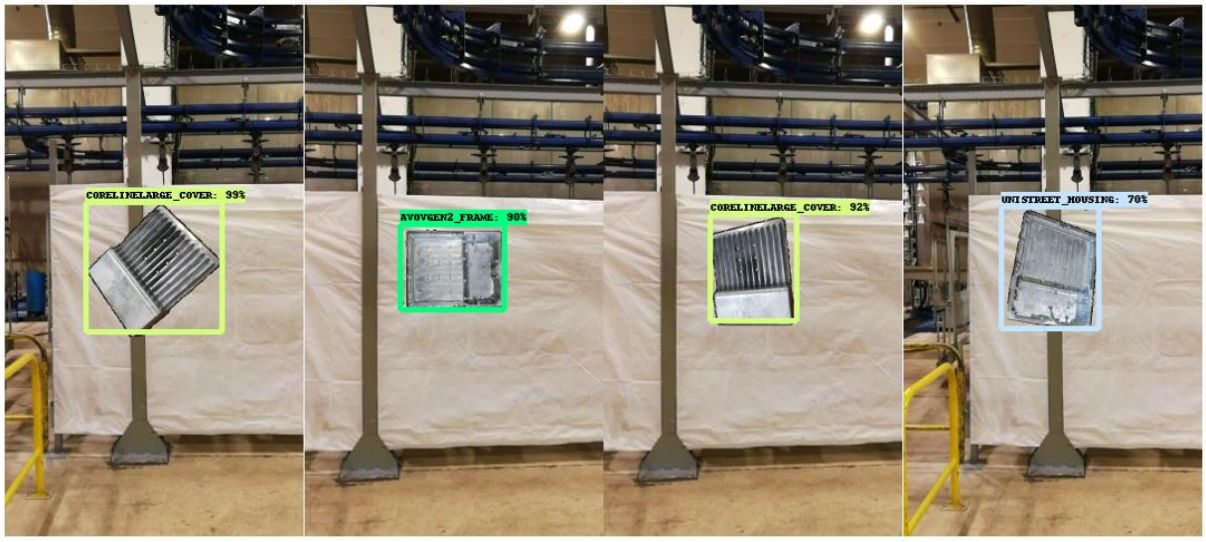

ILUSTRACIÓN 45: IMÁGENES DE VALIDACIÓN 6 CLASES: PRUEBA108, 125, 139 Y 179

En el futuro entrenamiento, al igual que pasaba con la clase 2, será importante tener en cuenta cual es la cara visible con mayor frecuencia y validar si es o no necesaria la eliminación de una de las 2 caras.

IV. Clase 4: DIGISTREETCATENARY_HOUSING:

Para esta clase, se observa que la confidence score de la clasificación se encuentra en torno al 70-80%, por lo que será otra de las luminarias a tener en cuenta en los filtrados posteriores.

Es importante observar que hay alguna imagen que se encuentra con un score por debajo del 70% (rango puesto por defecto como límite en el clasificador).

Para el clasificador final habrá que decidir si conviene o no reducir el límite a partir del cual se muestra en pantalla el "Bounding Box".

TABLA 13: CLASIFICACIÓN IMÁGENES 6 CLASES: DIGISTREETCATENARY_HOUSING

Imagen	Clase real	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Máximo
Prueba190	DIGISTREETCATENARY HOUSING	0,0068	0,0292	0,0135	0,7563	0,0039	0,6319	DIGISTREETCATENARY HOUSING
Prueba204	DIGISTREETCATENARY HOUSING	0,6408	0,1225	0,0569	0,7156	0,3174	0,7065	DIGISTREETCATENARY HOUSING
Prueba224	DIGISTREETCATENARY HOUSING	0,0121	0,0596	0,0048	0,6476	0,0060	0,6597	UNISTREET HOUSING
Prueba240	DIGISTREETCATENARY HOUSING	0,0115	0,0964	0,0154	0,7980	0,0047	0,5858	DIGISTREETCATENARY HOUSING

ILUSTRACIÓN 46: IMÁGENES DE VALIDACIÓN 6 CLASES: PRUEBA190, 204, 224 Y 240

V. Clase 5: LUMA_FRAME:

Esta clase es la que más problemas presentaba en el anterior análisis. A través de las imágenes se puede observar que hay grandes diferencias entre las que si logra clasificar (75-95%) y las que no (19-45%).

TABLA 14: CLASIFICACIÓN IMÁGENES 6 CLASES: LUMA_FRAME

Imagen	Clase real	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Máximo
Prueba259	LUMA FRAME	0,4445	0,0653	0,0378	0,0115	0,7902	0,1692	LUMA FRAME
Prueba279	LUMA FRAME	0,5714	0,0605	0,0243	0,0049	0,7679	0,0532	LUMA FRAME
Prueba292	LUMA FRAME	0,1614	0,0328	0,0839	0,0146	0,9296	0,3678	LUMA FRAME
Prueba303	LUMA FRAME	0,4750	0,1238	0,0589	0,0064	0,9226	0,2905	LUMA FRAME
Prueba269	LUMA FRAME	0,5085	0,0290	0,0153	0,0147	0,3462	0,4889	AVOVGEN2 FRAME
Prueba271	LUMA FRAME	0,2257	0,1196	0,0174	0,0610	0,4254	0,6231	UNISTREET HOUSING
Prueba273	LUMA FRAME	0,7065	0,0314	0,0383	0,0210	0,4323	0,3946	AVOVGEN2 FRAME
Prueba300	LUMA FRAME	0,1013	0,0632	0,0609	0,0314	0,1987	0,8638	UNISTREET HOUSING

ILUSTRACIÓN 47: IMÁGENES DE VALIDACIÓN 6 CLASES: PRUEBA259, 279, 292 Y 303

Sin embargo, se observa que siempre que da valores superiores elevados a esta clase (>40%) aunque otras clases tengan valores superiores, esas clases no terminan de llegar a su límite. Por ejemplo, la clase 1 habíamos visto que sus valores eran cercanos al 100%, los cuales quedan muy alejados de los obtenidos aquí pese a que en algunos caos sean los mayores.

ILUSTRACIÓN 48: IMÁGENES DE VALIDACIÓN 6 CLASES: PRUEBA269, 271, 273 Y 300

VI. Clase 6: UNISTREET_HOUSING:

Para esta clase, al igual que pasaba con la clase 1, los valores obtenidos cuando las imágenes corresponden a la clase correcta son muy elevados (superiores al 90%).

TABLA 15: CLASIFICACIÓN IMÁGENES 6 CLASES: UNISTREET_HOUSING

Imagen	Clase real	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Máximo
Prueba344	UNISTREET HOUSING	0,0085	0,0232	0,0895	0,0339	0,0089	0,9717	UNISTREET HOUSING
Prueba350	UNISTREET HOUSING	0,5835	0,0380	0,1965	0,0240	0,2378	0,8895	UNISTREET HOUSING
Prueba370	UNISTREET HOUSING	0,1423	0,0232	0,0878	0,0320	0,1702	0,9752	UNISTREET HOUSING
Prueba385	UNISTREET HOUSING	0,6402	0,1225	0,0566	0,0686	0,3126	0,9427	UNISTREET HOUSING

ILUSTRACIÓN 49: IMÁGENES DE VALIDACIÓN 6 CLASES: PRUEBA344, 350, 370 Y 385

h) Segundo y último caso de estudio (19 clases):

El modelo de 6 clases desarrollado en el anterior apartado se realizó con el fin de ajustar la implantación y detectar los posibles fallos a tener en cuenta. Una vez ajustada la metodología de reconocimiento, en este apartado gracias al primer entrenamiento realizado, y con los conceptos claros a tener en cuenta, se amplió el modelo de detección a 19 clases. Entre los conceptos a tener en cuenta se incluyen:

- Revisar la cara vista de cada luminaria en la zona deseada de acuerdo con la hoja de proceso de la zona de pintura.
- Analizar una a una las diferentes clasificaciones, y fijar sus valores del filtro de confidence score para añadirlos en el programa empleado para obtener el resultado final.

1. Selección de datos iniciales:

Para completar el primer apartado, se realiza un estudio de cuál es la cara vista de la luminaria en el punto de reconocimiento, definiendo si es la parte frontal o la parte trasera de la misma (o ambas en el caso de que la vista sea indiferente o igual):

TABLA 16: DATOS EMPLEADOS DURANTE EL SEGUNDO CASO DE ESTUDIO: 19 CLASES

Familia	Parte	Cantidad	Descripción	Cara vista	ID
AV/OV GEN2	FRAME	7818	AVOVGEN2_FRAME	Frontal y trasera	1
CITYSOUL GEN2	CANOPY	11437	CITYSOUL GEN2_CANOPY	Trasera	2
CITYSOUL GEN2	FRAME	11559	CITYSOUL GEN2_FRAME	Trasera	3
CLASSICSTREET	COVER	9692	CLASSICSTREET_COVER	Trasera	4
CLASSICSTREET	FRAME	9692	CLASSICSTREET_FRAME	Frontal y trasera	5
CLEARFLOOD	COVER	21549	CLEARFLOOD_COVER	Trasera	6
CLEARFLOOD	FRAME	32273	CLEARFLOOD_FRAME	Trasera	7
CORELINE LARGE	COVER	21771	CORELINE LARGE_COVER	Frontal	8
CORELINE MEDIUM	COVER	34642	CORELINE MEDIUM_COVER	Frontal	9
DIGISTREET CATENARY	HOUSING	4293	DIGISTREET CATENARY_HOUSING	Trasera	10
IJG	COVER	2169	IJG_COVER	Frontal y trasera	11
LUMA	COVER	57026	LUMA _COVER	Trasera	12
LUMA	FRAME	56966	LUMA_FRAME	Trasera	13
MINILUMA	COVER	25601	MINILUMA_COVER	Trasera	14
MINILUMA	FRAME	25552	MINILUMA_FRAME	Trasera	15
STELA LONG	COVER	5137	STELA LONG_COVER	Frontal y trasera	16
STELA SQUARE	COVER	6734	STELA SQUARE_COVER	Frontal y trasera	17
TUBEPOINT	HOUSING	23342	TUBEPOINT_HOUSING	Trasera	18
UNISTREET	HOUSING	23248	UNISTREET_HOUSING	Frontal	19

Con esta selección realizada, se procede a generar las imágenes de entrada y de validación como se había hecho anteriormente, obteniendo los siguientes datos:

- 199 imágenes totales de entrada de las 19 clases a predecir.
- 15 *frames* del video de entrada de 20 segundos de duración.
- 29677 imágenes generadas durante 3.15 horas (57 segundos por imagen de entrada original), dividiendo entre:
    - 19181 imágenes de entrenamiento.
    - 496 imágenes de validación.

2. Ejecución del entrenamiento:

La ejecución del entrenamiento se ha realizado de la misma forma a la realizada en el primer entrenamiento, con el mismo equipo y solo modificando los datos de entrada.

El tiempo total de ejecución y las etapas entrenadas han sido las siguientes:

- 123233 "*steps"* realizadas durante el entrenamiento (modelo empleado para hacer el gráfico de inferencia).
- Aproximadamente 4 días y 20 horas de entrenamiento.

En el siguiente gráfico obtenido con TensorBoard se puede observar los datos mencionados que muestran el tiempo de ejecución requerido por "step" del entrenamiento:

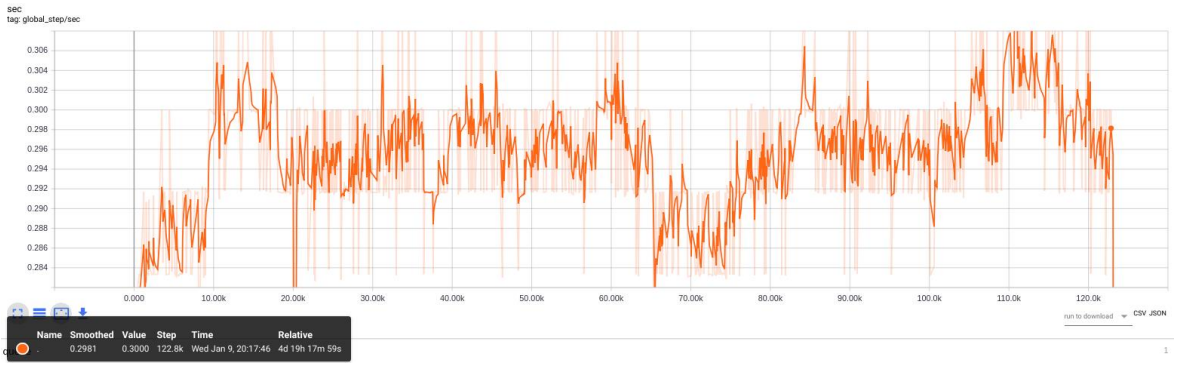

ILUSTRACIÓN 50: TIEMPO DE EJECUCIÓN POR ETAPAS: 19 CLASES

3. Visualización del entrenamiento con TensorBoard:

Al igual que se hizo con el modelo de 6 clases, se muestran las diferentes gráficas obtenidas con TensorFlow para explicar el proceso de entrenamiento en función de las diferentes funciones de pérdida:

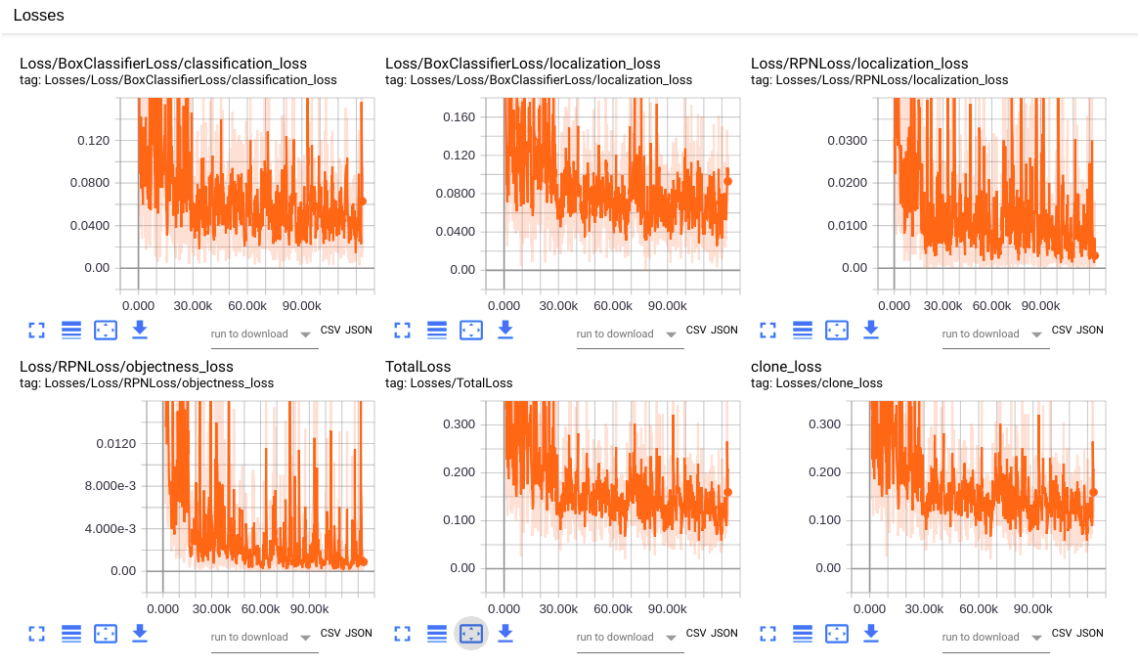

ILUSTRACIÓN 51: FUNCIONES DE PÉRDIDA DEL SEGUNDO CASO DE ESTUDIO: 19 CLASES

- ***RPN Loss*:** se observan que los valores obtenidos para estás métricas son mucho más bajos que los obtenidos en el primer entrenamiento, inferiores a 0.015.

ILUSTRACIÓN 52: RPN LOSS SEGUNDO CASO DE ESTUDIO: 19 CLASES

- ***Box Classifier Loss*:** al igual que con las funciones de pérdida anterior, al reducir la variabilidad de las piezas se observa una ligera mejoría en los resultados de las funciones de pérdida, rondando valores entre 0.04 y 0.1. También se puede observar como al aumentar el tiempo de entrenamiento, tras 30.0 k *steps,* el modelo presenta mejores resultados.

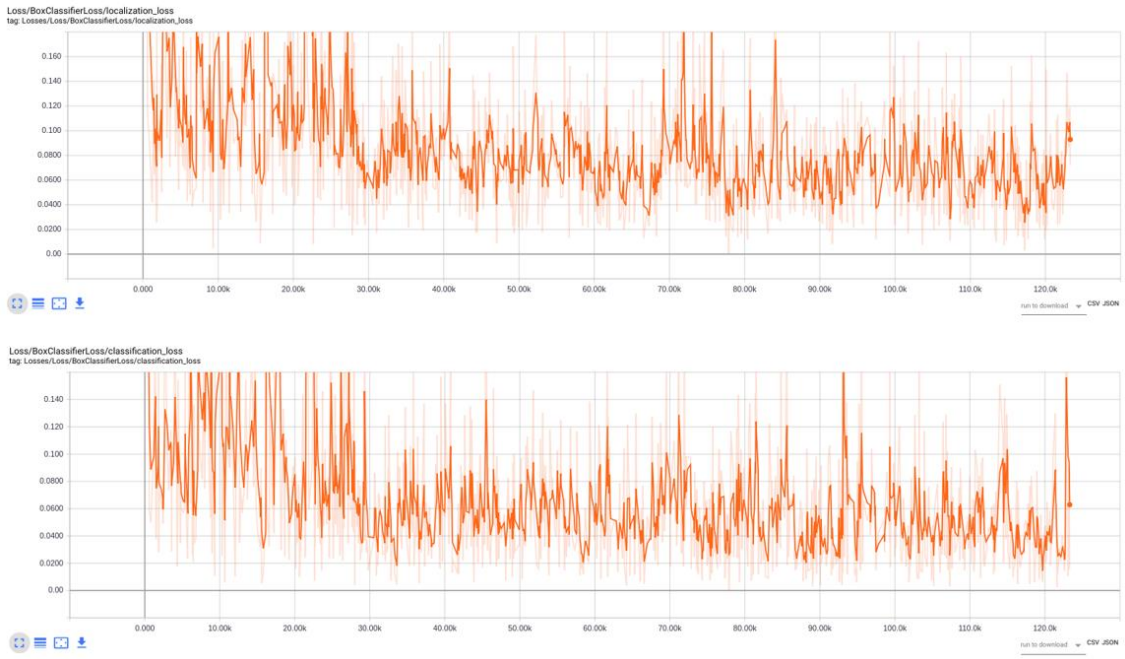

ILUSTRACIÓN 53: BOX CLASSIFICATION LOSS SEGUNDO CASO DE ESTUDIO: 19 CLASES

- ***Total Loss*:** debido a lo mostrado anteriormente, es evidente que al sumar los valores de cada una de las pérdidas, el valor final obtenido para la pérdida total disminuye, mostrando una gran mejoría a partir de las 30.0 k.

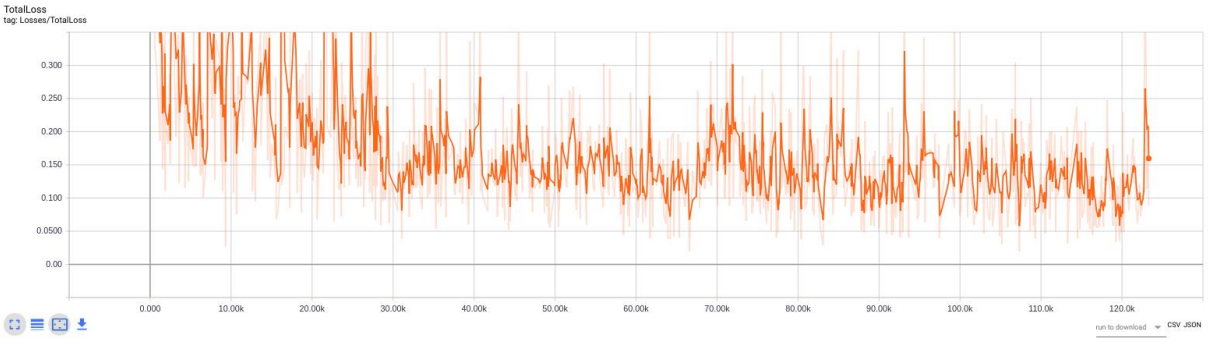

ILUSTRACIÓN 54: TOTAL LOSS SEGUNDO CASO DE ESTUDIO: 19 CLASES

4. Generación del grafico de inferencia a partir del último modelo entrenado:

Como se mencionaba anteriormente, para este modelo se ha realizado el gráfico de inferencia a partir de los datos almacenados para el *step* 123233.

Durante el entrenamiento, se realizaron diferentes gráficos de inferencia para ir viendo la calidad del clasificador, más concretamente en las siguientes etapas:

- **19866**: en esta etapa se observaba como el sistema no era capaz de reconocer todas las clases necesarias. Mientras que había clases claramente diferenciables por su forma que era capaz de distinguir (30%), las clases parecidas aún estaba muy lejos de aportar valores aceptables (70%).
- **65345**: en esta etapa ya se vio una gran mejoría, siendo el sistema de clasificar correctamente el 80% de las clases, pero fallando en el 20% restantes.
- **123233**: en esta etapa, usada para el modelo final, ya se muestra una gran mejoría, siendo capaz el sistema de reconocer todas las clases correctamente con valores altos, pero, mostrando problemas con 2 clases, aportando valores muy elevados de las mismas aun cuando en el fondo no aparece ningún objeto. El análisis específico se mostrará a continuación.

5. Validación del modelo:

- **Ejecución de la validación:**

Al igual que se hizo con el anterior modelo entrenado, la validación del modelo se realizará con todas las imágenes de validación generadas en el primer apartado, empleando el mismo programa y los pasos indicados en el modelo de 6 clases.

- **Análisis con datos e imágenes:**

Con los datos obtenidos en el archivo de salida Datos19Clases.xlsx (incluido en los archivos adjuntos) se procede a analizar los resultados obtenidos por el modelo como se hizo en el entrenamiento anterior, para fijar los valores de filtrado de cada clase para indicar correctamente a que clase pertenece cada imagen.

Se puede observar en la tabla siguiente como, si solo cogemos el valor máximo obtenido, hay ciertas clases que es capaz de clasificar con un porcentaje muy elevado o incluso del 100%, mientras que hay otras que no es capaz de acertar prácticamente nunca (clases 4, 18 y 19).

Esto puede ser debido a varios factores, no necesariamente a que esas clases no haya sido capaz de aprenderlas, sino, por ejemplo, que haya clases que las haya sobre aprendido o que en las imágenes el clasificador confunda características del fondo con piezas, y aunque sí que esté clasificando correctamente la clase correspondiente, hay otras zonas de la imagen que generan ruido y no permiten observar el resultado correcto.

TABLA 17: CLASIFICACIÓN DE LOS DATOS DE VALIDACIÓN: 19 CLASES

ID	CLASE REAL	IMAGENES A CLASIFICAR	CLASIFICADA CORRECTAMENTE		CLASIFICADA ERRONEAMENTE	
1	AVOVGEN2_FRAME	25	21	84%	4	16%
2	CITYSOULGEN2_CANOPY	16	16	100%	0	0%
3	CITYSOULGEN2_FRAME	29	29	100%	0	0%
4	CLASSICSTREET_COVER	32	8	25%	24	75%
5	CLASSICSTREET_FRAME	20	20	100%	0	0%
6	CLEARFLOOD_COVER	28	27	96%	1	4%
7	CLEARFLOOD_FRAME	32	32	100%	0	0%
8	CORELINELARGE_COVER	25	25	100%	0	0%
9	CORELINEMEDIUM_COVER	23	15	65%	8	35%
10	DIGISTREETCATENARY_HOUSING	23	23	100%	0	0%
11	IJG_COVER	18	18	100%	0	0%
12	LUMA_COVER	40	40	100%	0	0%
13	LUMA_FRAME	23	23	100%	0	0%
14	MINILUMA_COVER	15	12	80%	3	20%
15	MINILUMA_FRAME	19	15	79%	4	21%
16	STELALONG_COVER	52	51	98%	1	2%
17	STELASQUARE_COVER	36	34	94%	2	6%
18	TUBEPOINT_HOUSING	17	1	6%	16	94%
19	UNISTREET_HOUSING	23	0	0%	23	100%

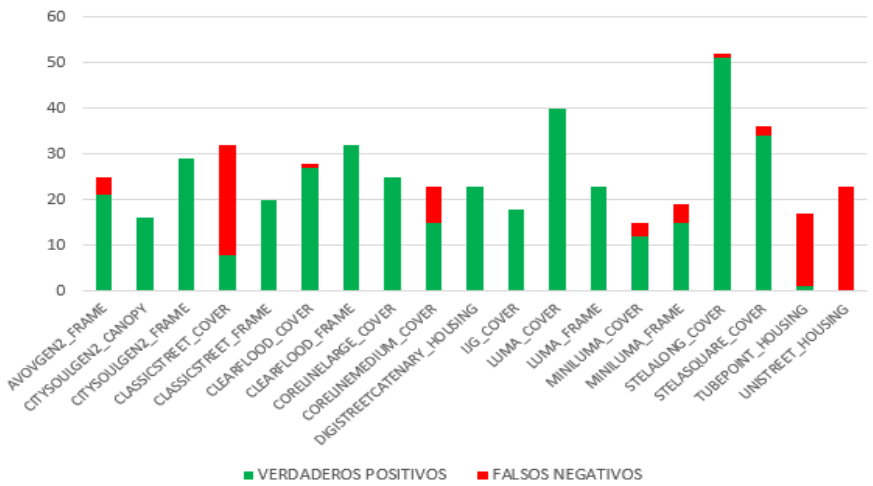

ILUSTRACIÓN 55: CLASIFICACIÓN DE LOS DATOS DE VALIDACIÓN: 19 CLASES

Para analizar mejor los problemas y diseñar correctamente el clasificador, como se hizo en el entrenamiento anterior, es necesario fijar el umbral del valor de *confidence* del clasificador. A continuación, se muestra una tabla con las clasificaciones que ha sufrido cada una de las clases, para observar si hay alguna clase que genere más problemas en las otras.

En la siguiente tabla se muestra la matriz de confusión del modelo, mostrando a la izquierda la clase real, y en cada una de las columnas la clase predicha. En la diagonal principal se muestra la cantidad de positivos reales obtenidos, mientras que tanto por encima como por debajo de esta se muestran los falsos positivos (el modelo clasifica una clase cuando no lo es), pudiendo saber también los falsos negativos (el sistema no clasifica como una clase una luminaria que sí lo es). Si todas las clasificaciones fueran correctas, todos los valores se deberían encontrar en la diagonal, sin embargo, se observa cómo hay ciertas clases como la 12 que añade ruido sobre las otras (genera falsos positivos en su propia clase), así como otras clases (18 o 19) que no es capaz de detectarlas a partir del valor máximo de entre todos los porcentajes (falsos negativos).

TABLA 18: MATRIZ DE CONFUSIÓN: 19 CLASES

ID CLASES		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	AVOVGEN2 FRAME	21					2						2							
2	CITYSOULGEN2 CANOPY		16																	
3	CITYSOULGEN2 FRAME			29																
4	CLASSICSTREET COVER				8						11		13							
5	CLASSICSTREET FRAME					20														
6	CLEARFLOOD COVER						27						1							
7	CLEARFLOOD FRAME							32												
8	CORELINELARGE COVER								25											
9	CORELINEMEDIUM COVER								3	15			5							
10	DIGISTREETCATENARY HOUSING										23									
11	IJG COVER											18								
12	LUMA COVER												40							
13	LUMA FRAME													23						
14	MINILUMA COVER												3		12					
15	MINILUMA FRAME												2	1	1	15				
16	STELALONG COVER												1				51			
17	STELASQUARE COVER												2					34		
18	TUBEPOINT HOUSING	10											6						1	
19	UNISTREET HOUSING					1						1	19	2						0

Si se obtiene un listado con los valores promedio obtenidos por cada una de las clases cuando el sistema es capaz de clasificar correctamente, podremos hacernos una idea de que valores son los que habrá que indicar en los filtrados. También se mostrará el valor promedio que obtiene la clase correcta cuando no es capaz de clasificar correctamente dicha clase, para ver si, aunque no sea capaz de reconocer la clase correcta con la mayor probabilidad, si es capaz de aportarle un valor suficientemente diferenciable como para filtrar. En la tabla siguiente se muestran los valores:

TABLA 19: VALORES PROMEDIO TP Y FN EN VALIDACIÓN: 19 CLASES

ID	CLASE REAL	PORCENTAJE PROMEDIO POSITIVOS REALES	CANTIDAD POSITIVOS	PORCENTAJE PROMEDIO FALSO NEGATIVO	CANTIDAD FALSO NEGATIVO
1	AVOVGEN2_FRAME	0,9227	21	0,5553	4
2	CITYSOULGEN2_CANOPY	0,9869	16	-	0
3	CITYSOULGEN2_FRAME	0,9815	29	-	0
4	CLASSICSTREET_COVER	0,7856	8	0,2579	24
5	CLASSICSTREET_FRAME	0,9774	20	-	0
6	CLEARFLOOD_COVER	0,9195	27	0,4780	1
7	CLEARFLOOD_FRAME	0,8719	32	-	0
8	CORELINELARGE_COVER	0,9892	25	-	0
9	CORELINEMEDIUM_COVER	0,7793	15	0,2899	8
10	DIGISTREETCATENARY_HOUSING	0,9964	23	-	0
11	IJG_COVER	0,9722	18	-	0
12	LUMA_COVER	0,9978	40	-	0
13	LUMA_FRAME	0,9837	23	-	0
14	MINILUMA_COVER	0,9739	12	0,6557	3
15	MINILUMA_FRAME	0,8377	15	0,5707	4
16	STELALONG_COVER	0,9634	51	0,7462	1
17	STELASQUARE_COVER	0,9547	34	0,6385	2
18	TUBEPOINT_HOUSING	0,4413	1	0,2286	16
19	UNISTREET_HOUSING	-	0	0,1995	23

Observando los resultados de una forma gráfica, se observa la diferencia de valores promedio entre las clasificaciones correctas y las no correctas explicadas a continuación:

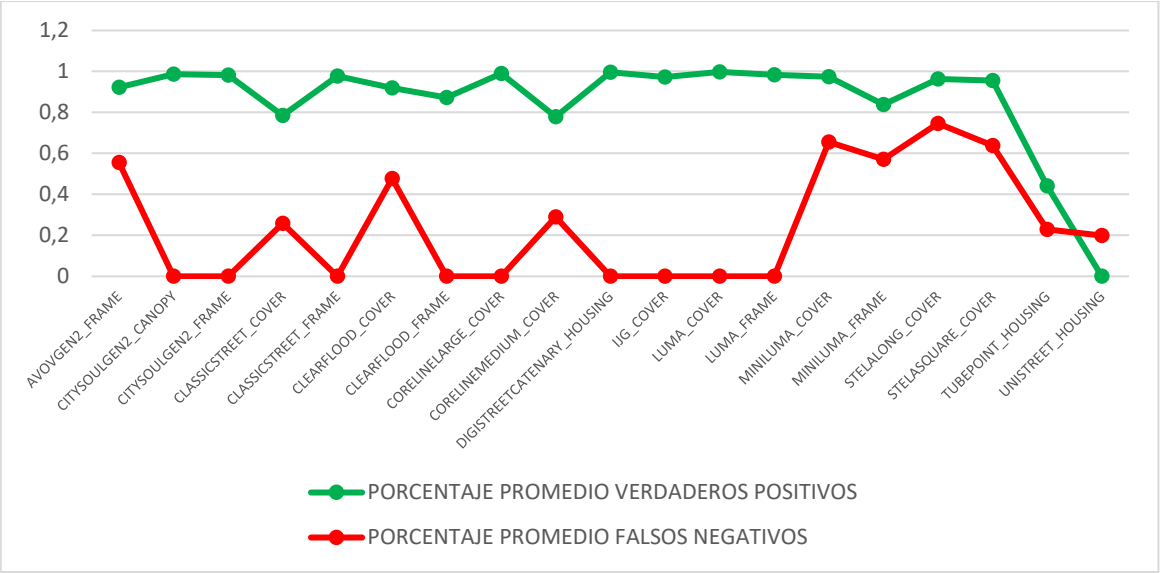

ILUSTRACIÓN 56: VALORES PROMEDIO POSITIVO Y FALSO NEGATIVO: 19 CLASES

De esta tabla se pueden sacar varias conclusiones y clasificar las clases en 4 posibles tipos para los posteriores filtrados:

- Las clases **que no se han clasificado mal nunca** presentan valores muy elevados de promedio en clasificación correcta. Es probable que sea debido a su diferencia física con respecto a otras clases. Es el caso de las clases 2, 3, 5, 7, 8, 10, 11, 12 y 13.
- Hay clases que cuando se han clasificado incorrectamente, presentan valores promedio muy elevados, y si nos fijamos en la cantidad de **veces que han fallado** se observa que habrá sido por cosas **puntuales** de la imagen. Por ejemplo, en la clase 16, se ha clasificado erróneamente 1 vez, sin embargo, el valor otorgado a la clase correcta esa vez ha sido del 74%, lo cual indica que si no se ha clasificado como esa es únicamente porque el sistema habrá encontrado ruido haciendo que otra clase adquiriese un valor superior. Estos casos en los que aun fallando los promedios superan el 45% son las clases 1, 6, 14, 15, 16, 17.
- Después, hay una clase que presenta un caso especial, como es la clase 9, donde pese a que ha sido capaz de clasificar correctamente un 65% de los casos, los valores promedio no superan valores de 80% (77,9%). Sin embargo, cuando fallan al clasificar, tampoco obtiene valores muy elevados (28%). Esta clase pese a que **cuando falla no presenta valores elevados** como las anteriores, al sí presentar **valores elevados al acertar**, habrá que tratarla de manera especial. La clase 4 se podría también asociar a este tipo de casos, puesto que presenta promedios

similares en acierto y fallo, pese a que solo es capaz de clasificarla bien en el 25% de los casos.

- El último caso es el de las clases 18 y 19, las cuales presentan **valores muy bajos al fallar**. Probablemente estas clases tendrán que tener un filtro del 0.1, y al superarlo, serán clasificadas como ellas mismas.
- Hay que tener en cuenta que la clasificación de **19 categorías tan similares** es compleja y **supone un desafío** para las técnicas existentes.

Para el caso particular del UNISTREET_HOUSING (clase 9) la cual es la única que no presenta ningún acierto, se muestra a continuación los valores obtenidos en las imágenes que correspondían a dicha clase, observando los valores alcanzados por dicha clase aunque el sistema no haya sido capaz de aportar solo con la clase de valor máximo verdaderos positivos:

TABLA 20: PORCENTAJES CLASE UNISTREET: 19 CLASES

Muestra	CLASE REAL	PORCENTAJE CLASE 19 PARA CADA MUESTRA	CLASE CON EL MÁXIMO	PORCENTAJE MAXIMO
A	UNISTREET_HOUSING	0,04962	0,96069	LUMA_COVER
B	UNISTREET_HOUSING	0,02634	0,98471	CLASSICSTREET_FRAME
C	UNISTREET_HOUSING	0,01405	0,99177	LUMA_COVER
D	UNISTREET_HOUSING	0,04222	0,96424	LUMA_FRAME
E	UNISTREET_HOUSING	0,03770	0,92512	LUMA_COVER
F	UNISTREET_HOUSING	0,09004	0,91431	LUMA_COVER
G	UNISTREET_HOUSING	0,25116	0,90557	LUMA_COVER
H	UNISTREET_HOUSING	0,37190	0,92561	LUMA_COVER
I	UNISTREET_HOUSING	0,10920	0,98169	LUMA_COVER
J	UNISTREET_HOUSING	0,15584	0,88502	LUMA_COVER
K	UNISTREET_HOUSING	0,15342	0,90002	LUMA_COVER
L	UNISTREET_HOUSING	0,45424	0,91727	LUMA_COVER
M	UNISTREET_HOUSING	0,03920	0,97455	IJG_COVER
N	UNISTREET_HOUSING	0,73723	0,94846	LUMA_COVER
O	UNISTREET_HOUSING	0,20730	0,94358	LUMA_COVER
P	UNISTREET_HOUSING	0,02126	0,98781	LUMA_COVER
Q	UNISTREET_HOUSING	0,01421	0,96284	LUMA_COVER
R	UNISTREET_HOUSING	0,62429	0,75771	LUMA_COVER
S	UNISTREET_HOUSING	0,65092	0,78667	LUMA_COVER

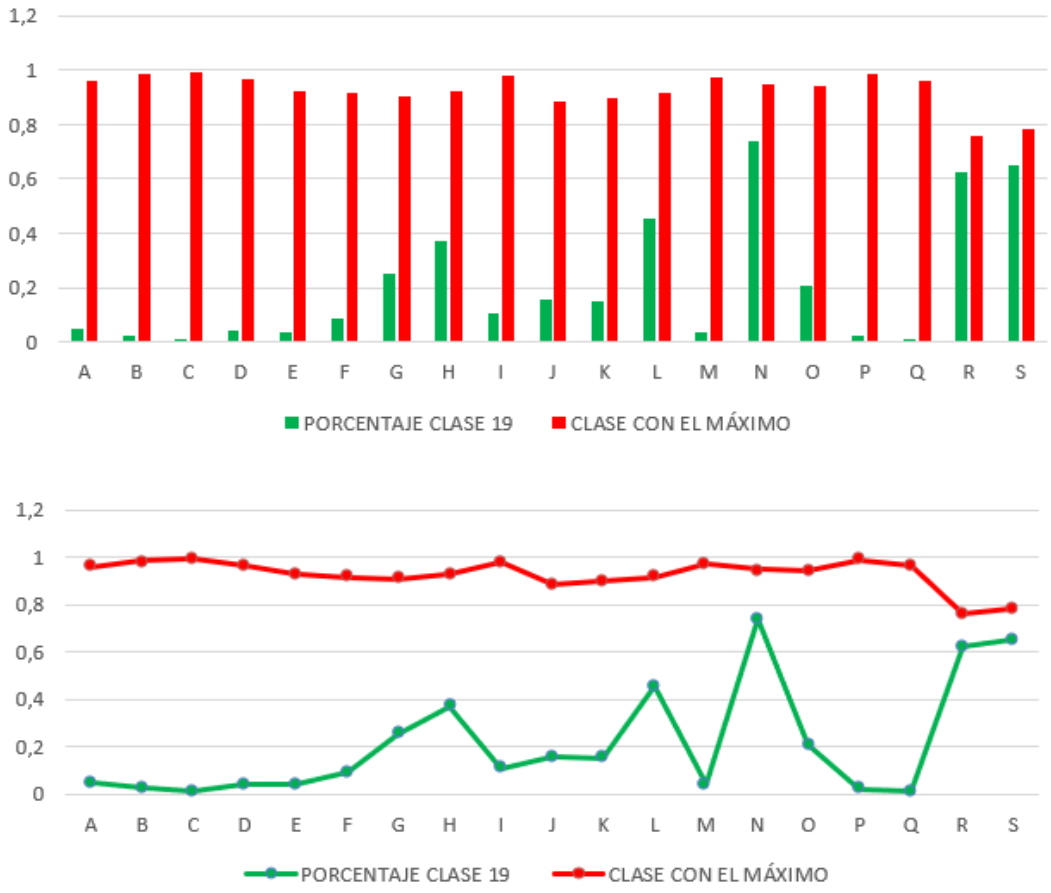

ILUSTRACIÓN 57: PORCENTAJES CLASE UNISTREET: 19 CLASES

Se observa en la tabla y en las imágenes como, pese a que no es capaz de clasificar ninguna vez correctamente, en más del 50% de los casos, obtiene valores superiores al 10% de probabilidad. Si tomamos el valor máximo de esta clase cuando no es la clase correcta, observamos que el valor máximo es 0.2232 (2.2%), valores que superan cuando es la clase correcta 16 de 19 veces (84.2 % de los casos) (datos obtenidos del archivo adjunto a la documentación Datos19Clases.xlsx). Esto implica que, si le ponemos un valor de filtrado del 2.5%, el clasificador daría clasificaciones correctas.

6. Ajuste de los filtros con datos de validación y de test:

Para terminar el proceso de modelado y pasar a la implementación, se realiza un ajuste de los filtros a emplear en el modelo final de producción para cada una de las clases. Para ello, se emplean 496 muestras divididas al 50% entre imágenes de validación y de test.

Con las imágenes de **validación** se realizará el ajuste de los filtros para cada una de las clases, obteniendo la matriz de confusión de la clasificación y ajustando dichos filtros en función de la métrica más conveniente, entre las que se encuentran:

- Precisión: proporción de ejemplos que son verdaderamente positivos.
- Recall: indica el ratio de los ejemplos positivos correctamente clasificados.

$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN}$$

ILUSTRACIÓN 58: MÉTRICAS EMPLEADAS EN VALIDACIÓN

En la ilustración anterior se encuentran las fórmulas que describen las diferentes métricas empleadas en adelante, siendo TP las clasificaciones realmente correctas, y FP - FN las clasificaciones erróneas cuando no es la clase que predecir o cuando se ha predicho mal.

En los datos de validación empezamos empleando unos filtros similares para todas las clases de 0,5. Con ello se obtiene la matriz de confusión mostrada a continuación, en la cual aparecen en cada fila la clase verdadera, y en cada columna las clases predichas entendiendo como predicha la que tiene un porcentaje mayor. En la diagonal principal se indican los verdaderos positivos, y fuera de ella los falsos negativos y los falsos positivos:

TABLA 21: MATRIZ DE CONFUSIÓN DE VALIDACIÓN CON FILTROS 0,5

ID CLASES		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	-
1	AVOVGEN2 FRAME	12																			
2	CITYSOULGEN2 CANOPY		8																		
3	CITYSOULGEN2 FRAME			15																	
4	CLASSICSTREET COVER				4						5		7								
5	CLASSICSTREET FRAME					10															
6	CLEARFLOOD COVER						13						1								
7	CLEARFLOOD FRAME							15													1
8	CORELINELARGE COVER								12												
9	CORELINEMEDIUM COVER								1	7			2								2
10	DIGISTREETCATENARY HOUSING										11										
11	IJG COVER											9									
12	LUMA COVER												20								
13	LUMA FRAME													12							
14	MINILUMA COVER												2		5						
15	MINILUMA FRAME												1			9					
16	STELALONG COVER												1				25				
17	STELASQUARE COVER												1					17			
18	TUBEPOINT HOUSING						4						2						0		2
19	UNISTREET HOUSING											1	10	1						0	
-	SIN IDENTIFICACION	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

En total observamos que las imágenes clasificadas correctamente son 204 (TP), mientras que las incorrectas han sido 44 (FN) y 39 (FP). La diferencia es debida a las imágenes que no han podido ser clasificadas. Por lo tanto, obtendríamos unas métricas totales como las siguientes:

- Precisión:

$$precisión = \frac{204}{204 + 39} * 100 = 83{,}95\%$$

- Recall:

$$recall = \frac{204}{204 + 44} * 100 = 82{,}26\%$$

Con los resultados de la tabla obtenidos con todos los filtros iguales sacamos las siguientes conclusiones:

- Hay clases que clasifican muy bien como es el caso de todas aquellas que muestran 100% en verdaderos positivos (clases 1, 2, 3…).
- Hay clases que con un umbral tan bajo generan muchos falsos positivos (clase 12) debido a que son clases que se podrían predecir con valores mucho más altos, presentando muy buen Recall (cuando es se clasifica bien) pero muy baja precisión (cuando no es también se clasifica como si lo fuera).
- Hay clases que con umbrales tan altos no son capaces de clasificarse correctamente, presentando alta precisión (o nula) y bajo Recall (clase 4).
- Al poner umbrales se puede dar el caso que ninguna clase supere el límite si se da el caso de que sea una imagen de una clase que necesite un filtro muy bajo para ser detectada.

Individualizando por clases, como en la siguiente tabla, obtendríamos los resultados de las métricas para cada una de las clases, observando los problemas descritos:

TABLA 22: RESULTADOS DE VALIDACIÓN CON FILTROS 0,5

ID CLASES		Verdaderos Positivos (TP)	Falsos Positivos (FP)	Falsos Negativos (FN)	Precisión (todo 0,5)	Recall (todo 0,5)
1	AVOVGEN2 FRAME	12	0	0	100,00%	100,00%
2	CITYSOULGEN2 CANOPY	8	0	0	100,00%	100,00%
3	CITYSOULGEN2 FRAME	15	0	0	100,00%	100,00%
4	CLASSICSTREET COVER	4	0	12	100,00%	25,00%
5	CLASSICSTREET FRAME	10	0	0	100,00%	100,00%
6	CLEARFLOOD COVER	13	4	1	76,47%	92,86%
7	CLEARFLOOD FRAME	15	0	1	100,00%	93,75%
8	CORELINELARGE COVER	12	1	0	92,31%	100,00%
9	CORELINEMEDIUM COVER	7	0	5	100,00%	58,33%
10	DIGISTREETCATENARY HOUSING	11	5	0	68,75%	100,00%
11	IJG COVER	9	1	0	90,00%	100,00%
12	LUMA COVER	20	27	0	42,55%	100,00%
13	LUMA FRAME	12	1	0	92,31%	100,00%
14	MINILUMA COVER	5	0	2	100,00%	71,43%
15	MINILUMA FRAME	9	0	1	100,00%	90,00%
16	STELALONG COVER	25	0	1	100,00%	96,15%
17	STELASQUARE COVER	17	0	1	100,00%	94,44%
18	TUBEPOINT HOUSING	0	0	8	-	0,00%
19	UNISTREET HOUSING	0	0	12	-	0,00%

Para entender como afectar aumentar o disminuir el valor del filtro en las métricas, se hace una prueba aumentando todos los filtros a 0,9 y otra disminuyendo a 0,1, obteniendo los siguientes resultados:

- Precisión:

$$precisión\ 0{,}1 = \frac{207}{207 + 41} * 100 = \ 83{,}47\%$$

$$precisión\ 0{,}9 = \frac{166}{166 + 15} * 100 = \ 91{,}71\%$$

- Recall:

$$recall\ 0{,}1 = \frac{207}{207 + 41} * 100 = \ 83{,}47\%$$

$$recall\ 0{,}9 = \frac{166}{166 + 82} * 100 = \ 66{,}94\%$$

TABLA 23: RESULTADOS DE VALIDACIÓN MODIFICANDO FILTROS

ID CLASES		Precisión (filtros 0,1)	Recall (filtros 0,1)	Precisión (filtros 0,5)	Recall (filtros 0,5)	Precisión (filtros 0,9)	Recall (filtros 0,9)
1	AVOVGEN2 FRAME	100,00%	100,00%	100,00%	100,00%	100,00%	66,67%
2	CITYSOULGEN2 CANOPY	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
3	CITYSOULGEN2 FRAME	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
4	CLASSICSTREET COVER	100,00%	25,00%	100,00%	25,00%	-	0,00%
5	CLASSICSTREET FRAME	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
6	CLEARFLOOD COVER	72,22%	92,86%	76,47%	92,86%	100,00%	57,14%
7	CLEARFLOOD FRAME	100,00%	100,00%	100,00%	93,75%	100,00%	75,00%
8	CORELINELARGE COVER	92,31%	100,00%	92,31%	100,00%	100,00%	100,00%
9	CORELINEMEDIUM COVER	100,00%	66,67%	100,00%	58,33%	-	0,00%
10	DIGISTREETCATENARY HOUSING	68,75%	100,00%	68,75%	100,00%	78,57%	100,00%
11	IJG COVER	90,00%	100,00%	90,00%	100,00%	87,50%	77,78%
12	LUMA COVER	41,67%	100,00%	42,55%	100,00%	64,52%	100,00%
13	LUMA FRAME	92,31%	100,00%	92,31%	100,00%	100,00%	100,00%
14	MINILUMA COVER	100,00%	71,43%	100,00%	71,43%	100,00%	71,43%
15	MINILUMA FRAME	100,00%	90,00%	100,00%	90,00%	100,00%	20,00%
16	STELALONG COVER	100,00%	96,15%	100,00%	96,15%	24,72%	84,62%
17	STELASQUARE COVER	100,00%	94,44%	100,00%	94,44%	100,00%	77,78%
18	TUBEPOINT HOUSING	100,00%	12,50%	-	0,00%	-	0,00%
19	UNISTREET HOUSING	-	0,00%	-	0,00%	-	0,00%

Con las modificaciones anteriores se observa que:

- Al **disminuir el valor del filtro del confidence score**, aunque en el total se mantiene la precisión como en los valores de 0,5, la precisión individual no mejora en ningún caso, puesto que para clasificar un valor como bueno hace falta que el sistema obtenga precisiones menores. En este caso, **no aparecen imágenes que no ha sido capaz de clasificar,** y por lo tanto, aparecen más verdaderos positivos, correspondientes a las imágenes clasificadas correctamente con los filtros de 0,5 pero que habían sido clasificadas con valores precisión del 30%. Por otro lado, el Recall global mejora porque al disminuir el umbral, casi todo lo que aparece es capaz de clasificarlo, haciendo que clases que necesitan un umbral muy bajo ya presenten verdaderos positivos (clase 18), haciendo posible medir sus métricas y observando lo bajas que son.
- Al **aumentar el valor del filtro sobre el confidence score** se observa una restricción de la clasificación. Esto es debido a que al ser más restrictivo, las clases que suelen predominar por haber aprendido mejor sus patrones mejoran su precisión, sin embargo, el resto de las clases que necesitan valores de filtro inferiores no son capaces de clasificar correctamente y disminuye su Recall. A diferencia de antes, ahora aparecen **67 imágenes sin identificación**.

Una vez analizados los diferentes casos y entendido el impacto de modificar los filtros, utilizando de apoyo los análisis del punto anterior y mediante ensayo se fijan nuevos filtros para cada una de las clases y se reanalizan todas las imágenes de validación, obteniendo los resultados mostrados en la tabla comparativa siguiente, en la cual se muestran individualizando por clases los resultados obtenidos, así como las métricas:

- Precisión:

$$precisión = \frac{201}{201 + 9} * 100 = \ 95{,}71\%$$

- Recall:

$$recall = \frac{201}{201 + 47} * 100 = \ 81{,}05\%$$

En primer lugar se observa como mejoran ambas métricas al poner filtros específicos por clase, haciendo que aquellas que clasifican con valores muy altos tengas filtros igual de altos, mientras que las que clasifican con valores muy pequeños, sean valores cercanos a estos los que las hagan clasificar. Por otro lado, los verdaderos positivos disminuyen ligeramente, pero sin ser relevante.

TABLA 24: RESULTADOS DE VALIDACIÓN CON FILTROS ESPECÍFICOS MÁS RESTRICTIVOS

ID CLASES		Filtros	Verdaderos Positivos (TP)	Falsos Positivos (FP)	Falsos Negativos (FN)	Precisión	Recall
1	AVOVGEN2 FRAME	0,65	12	0	0	100,00%	100,00%
2	CITYSOULGEN2 CANOPY	0,8	8	0	0	100,00%	100,00%
3	CITYSOULGEN2 FRAME	0,8	15	0	0	100,00%	100,00%
4	CLASSICSTREET COVER	0,4	5	0	11	100,00%	31,25%
5	CLASSICSTREET FRAME	0,8	10	0	0	100,00%	100,00%
6	CLEARFLOOD COVER	0,9	8	0	6	100,00%	57,14%
7	CLEARFLOOD FRAME	0,8	12	0	4	100,00%	75,00%
8	CORELINELARGE COVER	0,8	12	0	0	100,00%	100,00%
9	CORELINEMEDIUM COVER	0,6	7	0	5	100,00%	58,33%
10	DIGISTREETCATENARY HOUSING	0,95	11	2	0	84,62%	100,00%
11	IJG COVER	0,8	9	1	0	90,00%	100,00%
12	LUMA COVER	0,98	20	3	0	86,96%	100,00%
13	LUMA FRAME	0,98	10	0	2	100,00%	83,33%
14	MINILUMA COVER	0,8	6	0	1	100,00%	85,71%
15	MINILUMA FRAME	0,78	5	0	5	100,00%	50,00%
16	STELALONG COVER	0,8	25	0	1	100,00%	96,15%
17	STELASQUARE COVER	0,8	17	0	1	100,00%	94,44%
18	TUBEPOINT HOUSING	0,35	4	0	4	100,00%	50,00%
19	UNISTREET HOUSING	0,05	5	3	7	62,50%	41,67%

En cuanto a las métricas individuales se observa como ahora todas las clases obtienen muy buenos valores de precisión comparado con los casos anteriores, viéndose el Recall afectado en cuanto a que obtiene peores valores, pero no de manera exagerada.

Terminando de ver cómo afectan las modificaciones en los filtros, se rebajan los filtros de las clases 4, 6, 15 y 18 que presentaban un bajo Recall, obteniendo los siguientes resultados:

- Precisión:

$$precisión = \frac{210}{210 + 11} * 100 = 95{,}02\%$$

- Recall:

$$recall = \frac{211}{211 + 37} * 100 = 85{,}08\%$$

TABLA 25: RESULTADOS DE VALIDACIÓN CON FILTROS MENOS RESTRICTIVOS

	ID CLASES	Filtros	Verdaderos Positivos (TP)	Falsos Positivos (FP)	Falsos Negativos (FN)	Precisión	Recall
1	AVOVGEN2 FRAME	0,65	12	0	0	100,00%	100,00%
2	CITYSOULGEN2 CANOPY	0,8	8	0	0	100,00%	100,00%
3	CITYSOULGEN2 FRAME	0,8	15	0	0	100,00%	100,00%
4	CLASSICSTREET COVER	0,2	8	0	8	100,00%	50,00%
5	CLASSICSTREET FRAME	0,8	10	0	0	100,00%	100,00%
6	CLEARFLOOD COVER	0,8	12	2	2	85,71%	85,71%
7	CLEARFLOOD FRAME	0,8	12	0	4	100,00%	75,00%
8	CORELINELARGE COVER	0,8	12	0	0	100,00%	100,00%
9	CORELINEMEDIUM COVER	0,6	7	0	5	100,00%	58,33%
10	DIGISTREETCATENARY HOUSING	0,95	11	2	0	84,62%	100,00%
11	IJG COVER	0,8	9	1	0	90,00%	100,00%
12	LUMA COVER	0,98	20	3	0	86,96%	100,00%
13	LUMA FRAME	0,98	10	0	2	100,00%	83,33%
14	MINILUMA COVER	0,8	6	0	1	100,00%	85,71%
15	MINILUMA FRAME	0,72	7	0	3	100,00%	70,00%
16	STELALONG COVER	0,8	25	0	1	100,00%	96,15%
17	STELASQUARE COVER	0,8	17	0	1	100,00%	94,44%
18	TUBEPOINT HOUSING	0,35	5	0	3	100,00%	62,50%
19	UNISTREET HOUSING	0,05	5	3	7	62,50%	41,67%

- Al analizar los datos, en primer lugar se observa una mejoría de verdaderos positivos, aumentando de 201 a 210, causado por la disminución de los valores de filtro en las clases que necesitan menos porcentaje.
- En cuanto a la precisión, se observa una ligera disminución (95,71% a 95,02%) causada porque al disminuir el valor de filtro, el modelo es capaz de clasificar en esas clases de forma más fácil.
- Observando el Recall, se observa una mejoría (81,05% a 85,08%), como consecuencia de que mientras antes de cada vez que aparece una clase con un valor de porcentaje bajo se perdía, ahora en cambio no se pierde si ese valor bajo se encuentra por debajo del filtro antiguo y por encima del filtro nuevo inferior.

Como conclusión, a la hora de ponerlo en producción habrá que analizar que conviene más, si obtener una mejor precisión con umbrales altos a costa de saber que habrá menos imágenes clasificadas, o disminuir el umbral, para perder precisión clasificando más imágenes mal pero sabiendo que serán más las imágenes que se clasificarán. Esto puede hacerse de forma individualizada, una clase puede tener un filtrado de score diferente a otra,

porque sea más confusa o fácil de aprender por el modelo, y la elección de este umbral se convierte en un paso fundamental del sistema.

Para terminar, se realiza un análisis con los datos obtenidos tras ejecutar la clasificación de las **imágenes de test** para asegurar que con datos nuevos se mantienen las métricas.

En los datos obtenidos, se observa que se han detectado 221 de las 248 imágenes, un 89,11% de las imágenes, de las cuales se han detectado correctamente 215, siendo 27 las imágenes sin identificación cuyos porcentajes no han superado ningún umbral.

- Precisión:

$$precisión = \frac{215}{215 + 6} * 100 = 97{,}29\%$$

- Recall:

$$recall = \frac{215}{215 + 33} * 100 = 86{,}69\%$$

TABLA 26: DATOS DE TEST FINALES CON FILTROS

	ID CLASES	Verdaderos Positivos (TP)	Falsos Positivos (FP)	Falsos Negativos (FN)	Precisión	Recall
1	AVOVGEN2 FRAME	10	0	3	100,00%	76,92%
2	CITYSOULGEN2 CANOPY	8	0	0	100,00%	100,00%
3	CITYSOULGEN2 FRAME	14	0	0	100,00%	100,00%
4	CLASSICSTREET COVER	11	0	5	100,00%	68,75%
5	CLASSICSTREET FRAME	10	1	0	90,91%	100,00%
6	CLEARFLOOD COVER	13	4	1	76,47%	92,86%
7	CLEARFLOOD FRAME	14	0	2	100,00%	87,50%
8	CORELINELARGE COVER	13	0	0	100,00%	100,00%
9	CORELINEMEDIUM COVER	7	0	4	100,00%	63,64%
10	DIGISTREETCATENARY HOUSING	12	0	0	100,00%	100,00%
11	IJG COVER	9	0	0	100,00%	100,00%
12	LUMA COVER	19	1	1	95,00%	95,00%
13	LUMA FRAME	9	0	2	100,00%	81,82%
14	MINILUMA COVER	7	0	1	100,00%	87,50%
15	MINILUMA FRAME	6	0	3	100,00%	66,67%
16	STELALONG COVER	25	0	1	100,00%	96,15%
17	STELASQUARE COVER	17	0	1	100,00%	94,44%
18	TUBEPOINT HOUSING	3	0	6	100,00%	33,33%
19	UNISTREET HOUSING	8	0	3	100,00%	72,73%

En las métricas individuales se observa el alto nivel de precisión obtenido para todas las clases, así como valores muy elevados para el Recall en todas salvo en la clase 18, que podría disminuirse su filtro en caso de comprobar que no afecta a la precisión de ninguna otra clasificación.

4.4. Implementación en producción con dispositivos IoT:

En los apartados anteriores se ha explicado el proceso de obtención de los datos empleados para la creación del clasificador. Una vez hecho, es hora de diseñar el entorno en el que se ejecutará dicho modelo con la finalidad de obtener los datos lo mejor posibles, y almacenarlos para que sean de utilidad para la compañía. Para ello, se seguirá la estructura mostrada en la siguiente imagen, que explica el funcionamiento del sistema y las relaciones entre dispositivos:

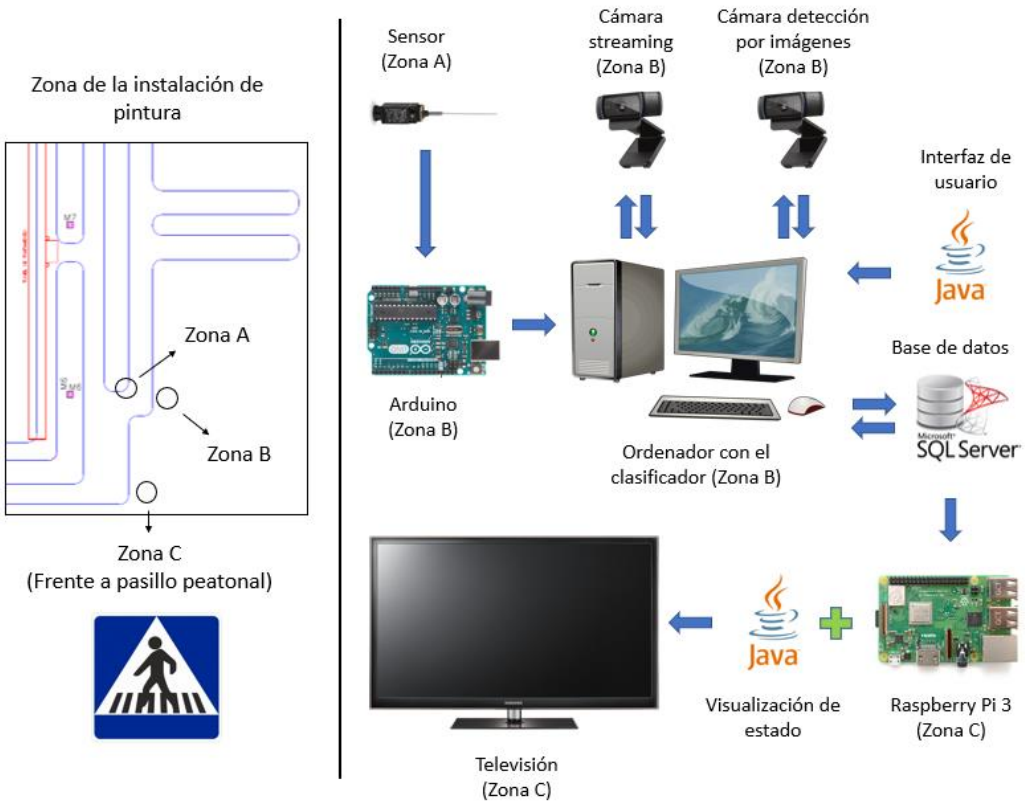

ILUSTRACIÓN 59: ESQUEMA DE LA INSTALACIÓN FINAL

Como se explica en la imagen, la instalación estará ubicada en la zona de la instalación de pintura fijada en el apartado 4.1. En esa ubicación se diferencian 3 zonas:

- **Zona A**: ubicación del sensor de perchas, situado en la propia cadena de pintura.
- **Zona B:** ubicación del software y hardware encargado de recibir las señales del sensor, realizar el análisis de las imágenes a través del interfaz de usuario y almacenar los datos en la base de datos de la fábrica.
- **Zona C:** ubicación del monitor de visualización público con el proceso de detección (esta ubicación puede variar en función de los requerimientos de la factoría).

a) Zona A: Sensorización de paso de perchas con Arduino uno:

Uno de los aspectos importante es el momento en el cual el modelo tiene que detectar los objetos que circulan por la instalación. Hasta ahora, simplemente se ha creado un modelo capaz de clasificar imágenes de entrada, sin embargo, el objetivo es identificar que material va en cada una de las perchas que circulan por la instalación, por lo que es imprescindible detectar cuando una percha nueva está pasando por la zona deseada.

Para ello, se va a emplear un sensor instalado en la entrada de la curva donde se instalará el detector y de donde se han tomado las imágenes de fondo para crear el clasificador (recordar el apartado 4.1 a)).

Para conectarlo al ordenador que tendrá el clasificador y transmitir la información desde el sensor, se empleará una placa de Arduino.

El diseño consta de 2 partes, explicadas a continuación:

1. **Desarrollo hardware:**

   En esta primera parte se va a describir todo lo necesario para el desarrollo hardware del detector, incluyendo tanto los elementos empleados como las conexiones correspondientes. Los dispositivos empleados son:

- Arduino Uno con cable *usb*.
- Sensor de paso *Telemecanique* XCK-P106
- Resistencia de 560Ω ± 5% (Verde/Azul/Marrón/Dorado).
- 4 conectores WAGO.
- 3 leds de colores (verde, rojo y amarillo)
- Cable de 8 hilos trenzados en pares de a 2 de 4 metros de longitud.
- Cables básicos de microelectrónica.

   Con todos los elementos disponibles, se realizan las conexiones correspondientes como se indican a continuación:

- Polo positivo de los leds a las siguientes señales digitales del Arduino:
  a) Led verde -> Señal 2
  b) Led rojo -> Señal 3
  c) Led amarillo -> Señal 4

- Polos negativos de los leds a una toma de tierra (GND).
- Señal de 5V (Vin) conectada a la señal A0 y A1 (Vout) del Arduino a través de una resistencia y el interruptor para controlar los pulsos que indican paso de señal con un divisor de tensión, donde R1 es la resistencia de 560Ω y R2 el interruptor normalmente abierto (NO) para A0 y normalmente cerrado (NC) para A1:

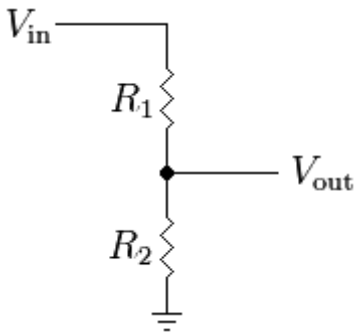

ILUSTRACIÓN 60: DIVISOR DE TENSIÓN PARA EL INTERRUPTOR

De forma más gráfica se muestra a continuación una imagen con el diagrama del conexionado:

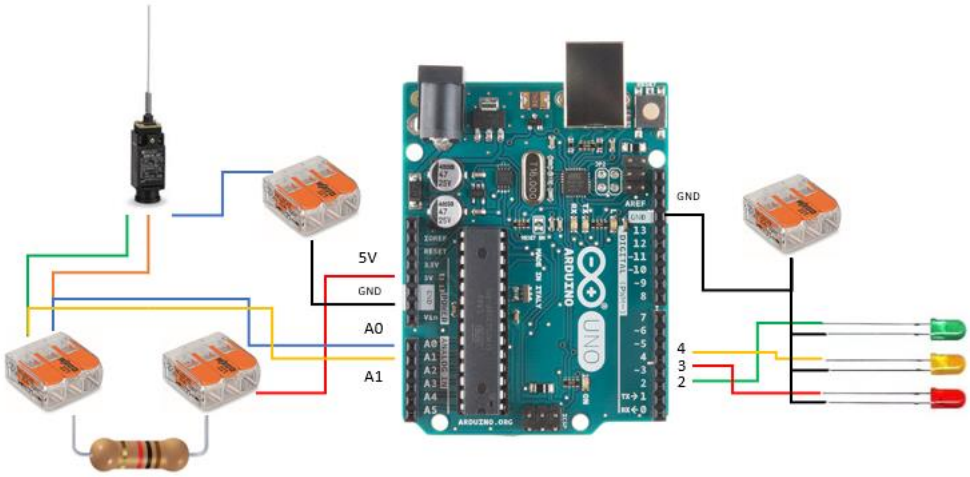

ILUSTRACIÓN 61: SENSOR DE PERCHAS: DIAGRAMA DEL CONEXIONADO HARDWARE DEL SENSOR

Una vez instalado en su ubicación, justo después del inicio de la curva en la que irá instalado el reconocedor de imágenes muestra el siguiente aspecto:

ILUSTRACIÓN 62: SENSOR DE PERCHAS: IMÁGENES REALES DE LA INSTALACIÓN

2. Desarrollo software:

Una vez diseñada la parte hardware, la otra parte importante es el control software del sensor. Para ello, se ha diseñado a través del IDE de Arduino, toda la lógica que seguirá el sensor, especificada en el siguiente diagrama:

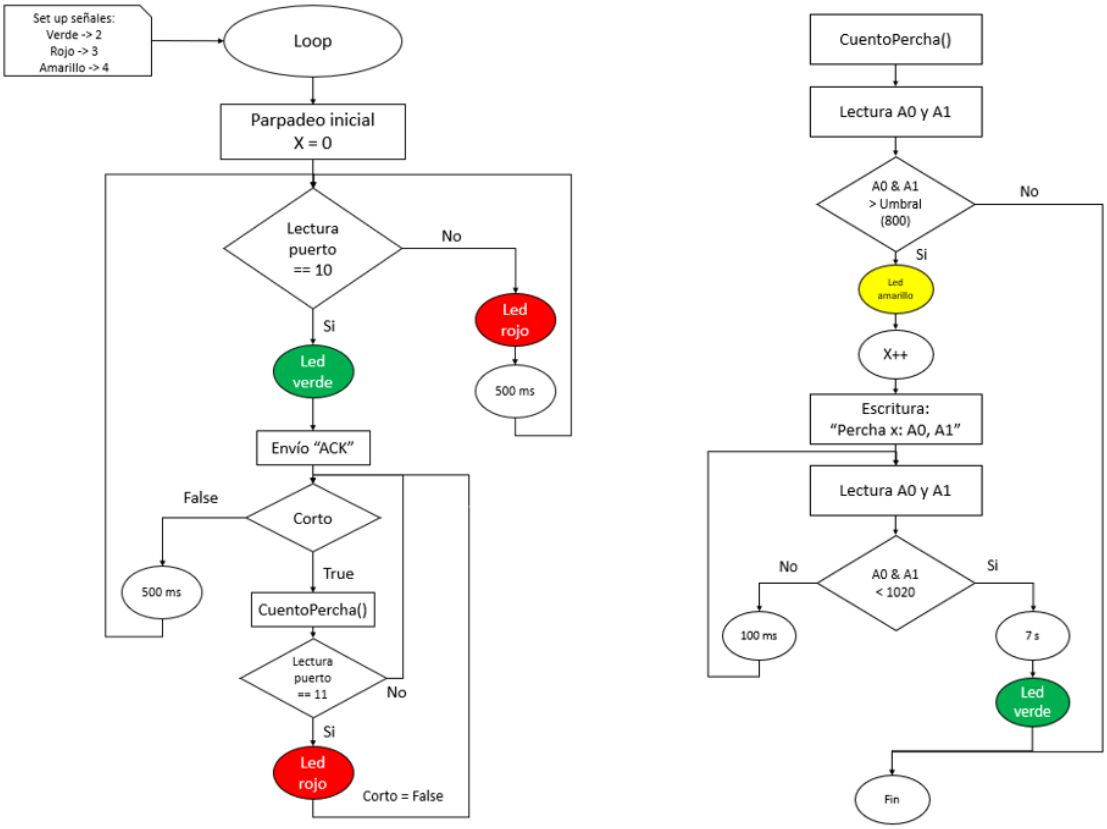

ILUSTRACIÓN 63: SENSOR DE PERCHAS: LÓGICA SOFTWARE

Como se muestra en el diagrama, el objetivo del sensor es comunicarse con el ordenador con el que estará conectado trasmitiendo la información necesaria de acuerdo con la siguiente lógica:

- Lectura del puerto serie hasta encontrar la señal enviada por el ordenador.
- En cuanto la recibe (10), envía un “ACK” al ordenador para transmitirle que ha recibido la información.
- Entra en un bucle continuo de lectura de perchas hasta que el ordenador le indique que tiene que parar (11).
- En cada lectura de percha, si el valor de las señales analógicas supera el valor de umbral (800 -> 3,9V), se indica que se ha detectado una percha. El sensor realiza lecturas hasta que compruebe que ambas señales analógicas vuelven a descender (< 1020 ≈ 5V).
- Si el sensor recibe un comando de paro desde el ordenador (11), deja de leer y pasa al estado inicial de lectura hasta recibir de nuevo un 10.

El código mostrado se encuentra incluido en los anexos (Apartado A11).

b) Zona B: Interfaz de usuario (aplicación Java) y almacenamiento:

Para facilitar el uso del modelo, se diseñará un interfaz de usuario que permita manejar el sistema de una forma fácil y sencilla desde el ordenador situado en la zona B indicada en la ilustración de la instalación. Para ello, se creará un interfaz de usuario en Java, con doble funcionalidad:

- Apertura de una ventana para hacer detección en tiempo real de todo lo que pase por delante de la cámara.
- Detección y almacenamiento en la base de datos de la fábrica del contenido de las perchas justo en el momento que el sensor instalado indique que tiene que realizar la detección.

ILUSTRACIÓN 64: INTERFAZ DE USUARIO: VISTA INICIAL

Esta doble funcionalidad permitirá tanto almacenar datos, como visualizar en tiempo real el funcionamiento de la detección. A continuación, se explican más en detalle cada una de las funcionalidades.

- **Identificación en tiempo real:**

  Esta funcionalidad servirá simplemente para permitir al usuario ejecutar el código "TFM_streaming.py" (Apartado A15) cuya finalidad es abrir una ventana, cargar el modelo almacenado en la carpeta:

  "Tensorflow/models/research/object_detection/TFM_model"

  y a través de una de las 2 webcam de las que dispondrá el ordenador, realizar una clasificación en tiempo real de todo lo que pase en frente de la instalación.

- **Extracción de características, filtrado, identificación y almacenamiento:**

  Esta funcionalidad, tiene como objetivo capturar imágenes cuando se lo indique el sensor de perchas, analizar cada una de las imágenes a través del clasificador entrenado llamando a un programa similar al empleado durante la validación "TFM_analizarImagen.py" (Apartado A13) y devolver al programa la imagen captada a través de la webcam, y los porcentajes otorgados por el modelo a cada una de las clases para un posterior análisis.

Este programa, a través de los datos obtenidos, será el encargado de analizar los porcentajes, y a partir de los valores de filtro puestos a cada una de las clases, indicar que objeto es realmente el de la imagen analizada.

A estas dos funcionalidades explicadas anteriormente, se podrá acceder a través del interfaz de usuario mostrado en la siguiente ilustración, la cual constará de diferentes zonas:

1. **Estado del sensor:** en la parte superior izquierda se mostrará el estado de la comunicación con el Arduino que está conectado con el sensor de las perchas. Nada más iniciar la aplicación, el programa intentará establecer conexión con el Arduino buscando en los diferentes puertos disponibles en el ordenador si en alguno hay algún Arduino conectado. En caso de conectarse, se indicará en verde en el apartado de estado. En caso contrario, se indicará en rojo.

ILUSTRACIÓN 65: INTERFAZ DE USUARIO: ESTADO DEL SENSOR

2. **Lectura de perchas:** en caso de que la comunicación sea correcta entre el Arduino con el sensor y la aplicación, esta entrará en un modo de escucha continua, esperando a recibir señales de paso de perchas desde el sensor. En caso de recibir una señal por el puerto serie del ordenador, la aplicación almacenará el registro y ejecutará el código “CamaraUSB.py” (Apartado A12) el cual se encargará de:
   a) Realizar una captura de imagen con la webcam 1 conectada al ordenador otorgándole el nombre único acorde a la siguiente nomenclatura:

      yyyyMMdd_hhmmss.png

      Siendo *yyyy* el año, *MM* el mes, *dd* el día, *hh* la hora, *mm* los minutos y *ss* los segundos en que se detectó la percha y se pidió hacer una imagen.

   b) Almacenar esa imagen en la carpeta “FotosOriginales” de la app y en la carpeta “sceneA” de la dirección

      “Tensorflow/models/research/object_detection”.

   c) Ejecutar el código “TFM_analizarImagen.py” el cual se encargará de tomar la imagen y encontrar los diferentes objetos presentes a través del modelo almacenado en dicha dirección.
   d) Almacenará la imagen devuelta en la carpeta “FotosAnalisis” de la aplicación.

e) Almacenará el archivo pA.csv con las probabilidades otorgadas para cada clase por el modelo.

f) Almacenamiento en una tabla de la base de datos SQL Server de la fábrica la información de la percha, la hora y la clasificación.

Realizados todos estos pasos, la aplicación mostrará las 2 imágenes en su pantalla (tanto la original como la analizada con los "Bounding Box"), tomará el archivo pA.csv, comprobará los límites puestos a cada una de las clases, los cuales han sido definidos previamente, y mostrará por pantalla todos los datos del archivo .csv así como la clase detectada por el modelo.

En la siguiente imagen se puede ver un ejemplo de ejecución donde la imagen ha sido clasificada por el modelo, e indica la clase que puede ser de acuerdo con los filtros establecidos en ese momento.

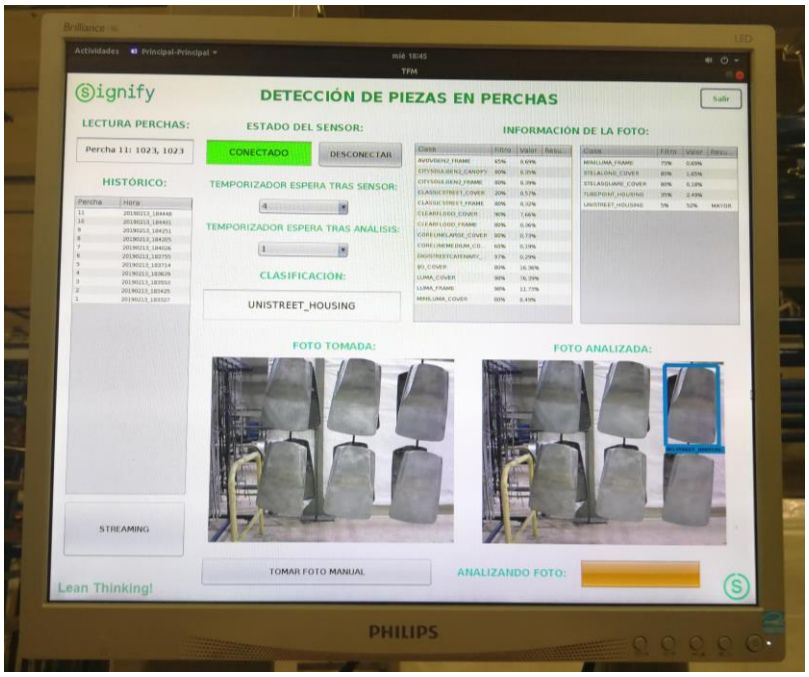

ILUSTRACIÓN 66: INTERFAZ DE USUARIO: EJEMPLO DE EJECUCIÓN

3. **Streaming:** la otra funcionalidad mencionada anteriormente es la de ejecutar una ventana para realizar la clasificación en tiempo real. Al pulsar sobre el botón "Streaming", la aplicación ejecuta el código "CamaraStreaming.py" (Apartado A14) la cual se encarga de ejecutar el código "TFM_streaming.py" (Apartado A15) situado en la dirección donde se encuentra el modelo de TensorFlow:

"Tensorflow/models/research/object_detection".

ILUSTRACIÓN 67: VISUALIZACIÓN DEL STREAMING CON LA CLASIFICACIÓN EN TIEMPO REAL

4. **Foto manual:** la aplicación permite también la posibilidad de ejecutar la captura y reconocimiento de imágenes sin necesidad del sensor, a través del botón “TOMAR FOTO MANUAL”, el cual ejecuta la misma secuencia generada en cada evento de percha, pero añadiendo al nombre de la imagen el prefijo “FotoManual”.

c) Zona C: Visualización de cara al público:

Terminada la parte de adquisición de datos, hay que gestionar la parte de visualización de cara al público. Esta parte es muy importante puesto que para que el proyecto continúe y se tenga en cuenta, hace falta que sea visible para cualquier persona que pase por la instalación, mostrando la nueva zona inteligente de la fábrica y añadiendo valor a la compañía ante futuras visitas demostrando la capacidad de desarrollo interno de la factoría.

Para ello, los requerimientos son los siguientes:

1. **Desarrollo hardware:**
   - Raspberry Pi: para la visualización de los datos, se configuró una Raspberry Pi modelo 3 con S.O. Raspbian, con toma de datos vía ethernet y alimentación a través del alimentador oficial de Raspberry Pi conectado a la corriente.

- Televisión o monitor de ordenador: para visualizar el S.O. de la Raspberry Pi se empleó vía HDMI o VGA una televisión o un monitor de ordenador.

2. Desarrollo software:

- Aplicación JAVA: para la visualización del estado del clasificador de perchas, se diseñó un interfaz de usuario, el cual, muestra el estatus del clasificador en tiempo real accediendo cada 10 segundos a la base de datos de la fábrica donde se almacenan los datos desde el ordenador, y mostrando el resultado y un gráfico de la clasificación de las perchas durante cada jornada laboral.

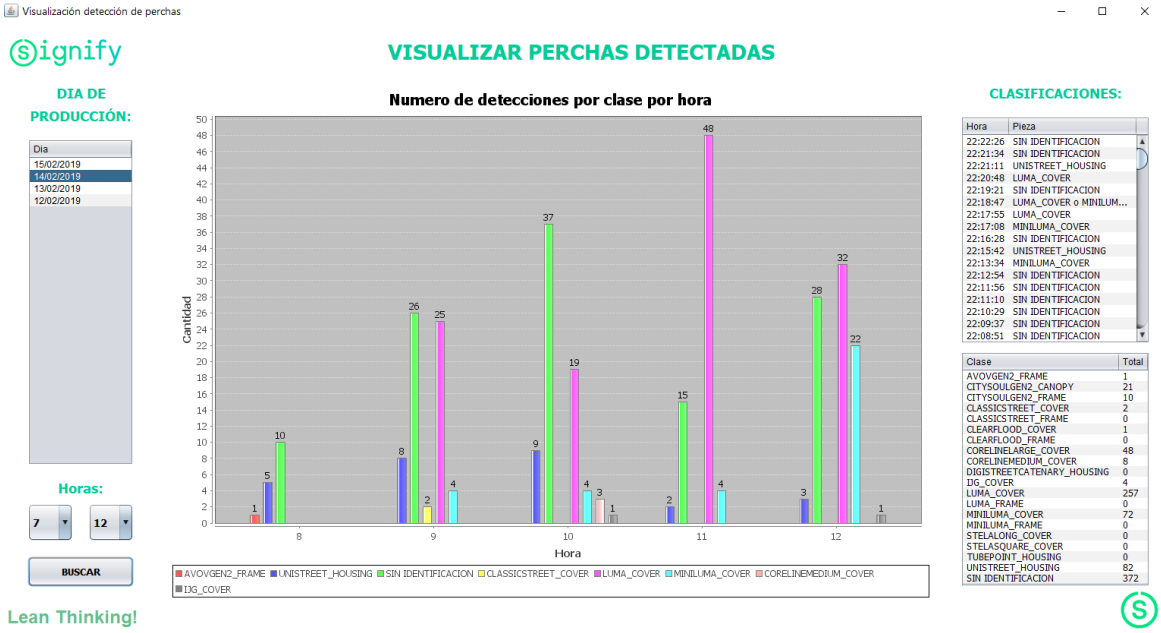

ILUSTRACIÓN 68: VISUALIZACIÓN DE LA JORNADA LABORAL DEL CLASIFICADOR

Lo bueno de este sistema de visualización es que es replicable tantas veces como se quiera. Simplemente sería necesario grabar la imagen de la Raspberry Pi en una nueva tarjeta SD e instalar el Hardware en la zona de la factoría deseada.

d) Instalación final y puesta en marcha de la versión inicial:

1. Requerimientos del sistema operativo:

Para poder continuar, es necesario tener instalados una serie de funcionalidades en el sistema que pretenda replicar los siguientes apartados, entre los que se encuentran:

- **Sistema Ubuntu (Linux):**

Puede ser tanto el único sistema operativo del computador donde se vaya a ejecutar como una partición del disco duro. En el apartado A16 de los anexos se explican los pasos realizados para la instalación de un sistema Ubuntu como una partición del disco duro de un computador con Windows 10, así como los seguidos para la instalación del sistema Ubuntu en un ordenador sin particiones de disco, empleado para la instalación y puesta en producción.

- **TensorFlow:**

Para poder ejecutar los siguientes apartados, es necesario tener implantado en el computador la librería TensorFlow, así como una versión de **Python 3.6 o inferior** (la versión 3.7 a día 22/12/2018 no dispone de versión de TensorFlow compatible). En el apartado A17 y Apartado A18 de los anexos se explican los pasos seguidos para su instalación.

- **API Object Detection:**

Una vez instalado TensorFlow, es necesario realizar la instalación de la API *Object Detection*, la cual será la base para la creación del detector de imágenes. En el apartado A18 de los anexos se explican los pasos realizados para su instalación.

2. Instalación de software necesario:

Para realizar la instalación en el PC que se encargará de ejecutar el modelo clasificador, es necesario realizar las siguientes instalaciones:

- Instalación Ubuntu (Apartado A16).
- Instalación de Python 3.6 con Anaconda (Apartado A17).
- Instalación de TensorFlow y la API object-detection (Apartado A18).
- Instalación de OpenCV (Apartado A19).
- Instalación de Arduino (Apartado A20).
- Instalación de Java (Apartado A21)

- Instalación de Netbeans en Ubuntu (Apartado A22). Este punto no es imprescindible.

3. Instalación de hardware necesario:

Una vez instalado el software, el siguiente paso es conectar todo el hardware que será empleado por el sistema, entre las que se incluye:

- **Arduino uno:**

Para que funcione el sensor correctamente, es necesario conectar el Arduino uno al puerto serie y otorgarle permisos de acceso al usuario. Para ello, se conecta el usb del Arduino a uno de los puertos y en un terminal se teclea lo siguiente:

```
ls -l /dev | grep ACM
```

Con esto introducido, se observará el puerto al que está conectado el Arduino, así como los permisos disponibles, a través de una línea parecida a la siguiente:

```
crw-rw-- 1 root dialout 166, 0 sep 25 14:42 ttyACM0
```

Para otorgar permisos a ese puerto a todos los usuarios hay que introducir la siguiente línea:

```
sudo chmod 777 /dev/ttyACM0
```

Una vez hecho esto, si introducimos otra vez la primera línea de comandos observaremos algo similar a lo que se muestra a continuación:

```
crwxrwxrwx 1 root dialout 166, 0 sep 25 14:42 ttyACM0
```

Gracias a esto, ya podremos acceder al Arduino uno desde cualquier usuario del sistema.

- **Cámaras usb:**

Este paso es muy sencillo, simplemente conectando las cámaras usb a los puertos del ordenador funcionarían.

- **Raspberry Pi:**

Pese a que finalmente se decidió optar por un ordenador por cuestiones de velocidad de procesamiento y calidad de detección puesto que en una Raspberry habría que usar otro modelo pre-entrenado durante el entrenamiento, el sistema está diseñado para ser totalmente integrado en una Raspberry Pi, tanto el interfaz de usuario en Java, las conexiones con Arduino, como todas las librerías necesarias. Siguiendo los pasos indicados para la instalación de Ubuntu en un ordenador y copiando los archivos relativos al interfaz,

se podría instalar y poner en funcionamiento con un modelo entrenado que pudiese soportar el procesador de la Raspberry Pi.

4. Instalación del programa java y los códigos Python:

Una vez instalado el software y el hardware necesario, es necesario instalar el programa en Java que servirá de interfaz de usuario, así como los códigos en Python que se encargan de realizar las fotos y analizar las imágenes.

Para ello, se crea una carpeta denominada DetectorPerchas en la dirección */home/signifyvalladolid* y en dicha dirección se pegan los siguientes archivos:

- Carpeta Camara, que incluirá los archivos:
    a) CamaraStreaming.py
    b) CamaraUSB.py
    c) Nombre.txt
- Carpeta FotoAnalisis
- Carpeta FotosOriginales
- Carpeta lib
- Carpeta SalidaAnalisis
- Archivo Filtros.csv
- Archivo TFM.jar

Con todos estos archivos creados, el siguiente paso es crear un ejecutable en el escritorio (Apartado A23).

Ejecutada esa última línea de código, aparecerá una ventana como la siguiente, en la cual habrá que indicar en Tipo, el campo “aplicación desde terminal”, en Nombre el nombre deseado, y en Comando, el archivo script.sh:

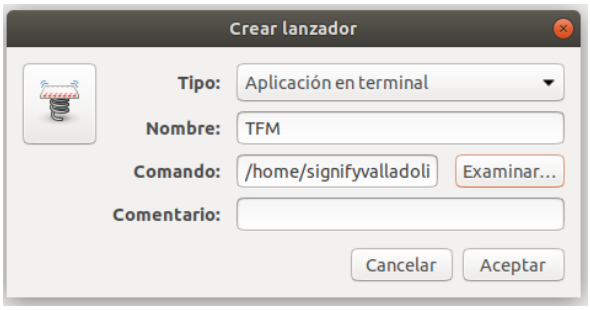

ILUSTRACIÓN 69: CREACIÓN DEL EJECUTADOR DE LA APLICACIÓN

5. Instalación final y puesta en marcha:

Finalizado todo el proceso de creación del modelo y software, así como instalado el hardware, se terminó la instalación del entorno como se muestra en la siguiente imagen:

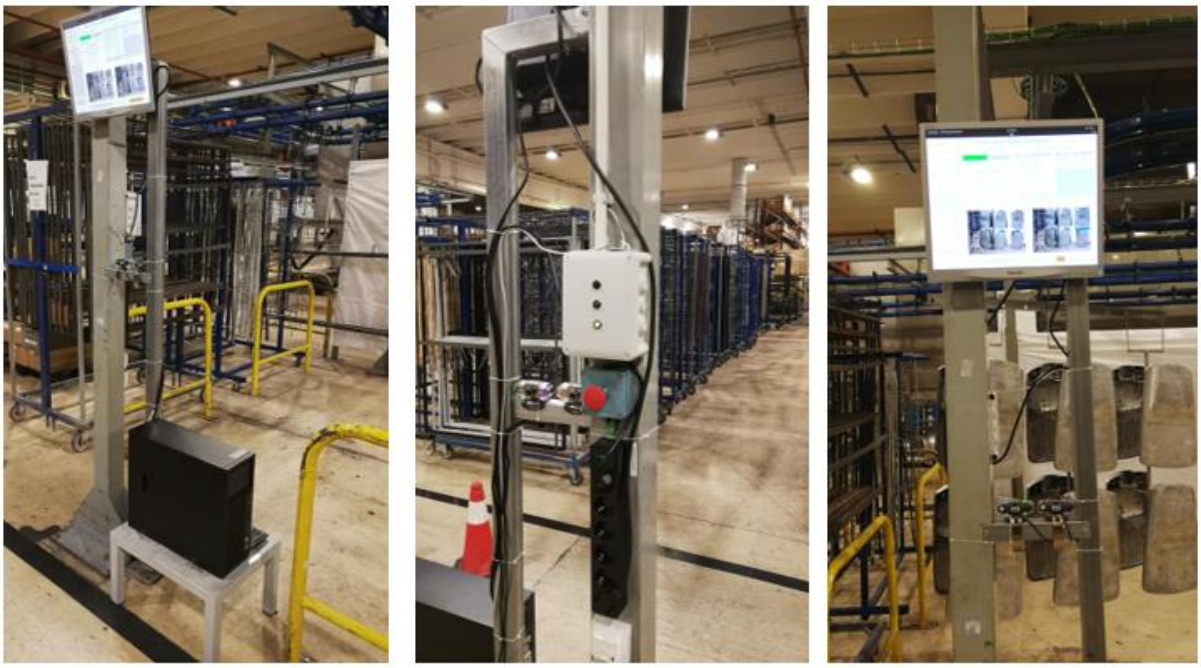

ILUSTRACIÓN 70: INSTALACIÓN FINAL

Durante las primeras semanas se realizará un análisis de las fotos tomadas para comprobar si los parámetros de los filtros son los adecuados o es necesario modificarlos. A continuación se muestran algunas imágenes tomadas durante el primer día de puesta en marcha (dentro del periodo de prueba inicial), mostrando en todas ellas clasificaciones superiores al 87%, y en muchos casos cercanas al 100%, validando que la metodología seguida durante la creación de modelo propuesto en el TFM, generando muestras suficientes a partir de técnicas de aumentación de datos, han sido adecuadas:

- CITYSOULGEN2_CANOPY: 99%
- CITYSOULGEN2_FRAME: 89%
- CORELINELARGE_COVER: 98%
- CORELINEMEDIUM_COVER: 88%
- IJG_COVER: 89%
- LUMA_COVER: 99%
- MINILUMA_COVER: 97%
- STELALONG_COVER: 99%

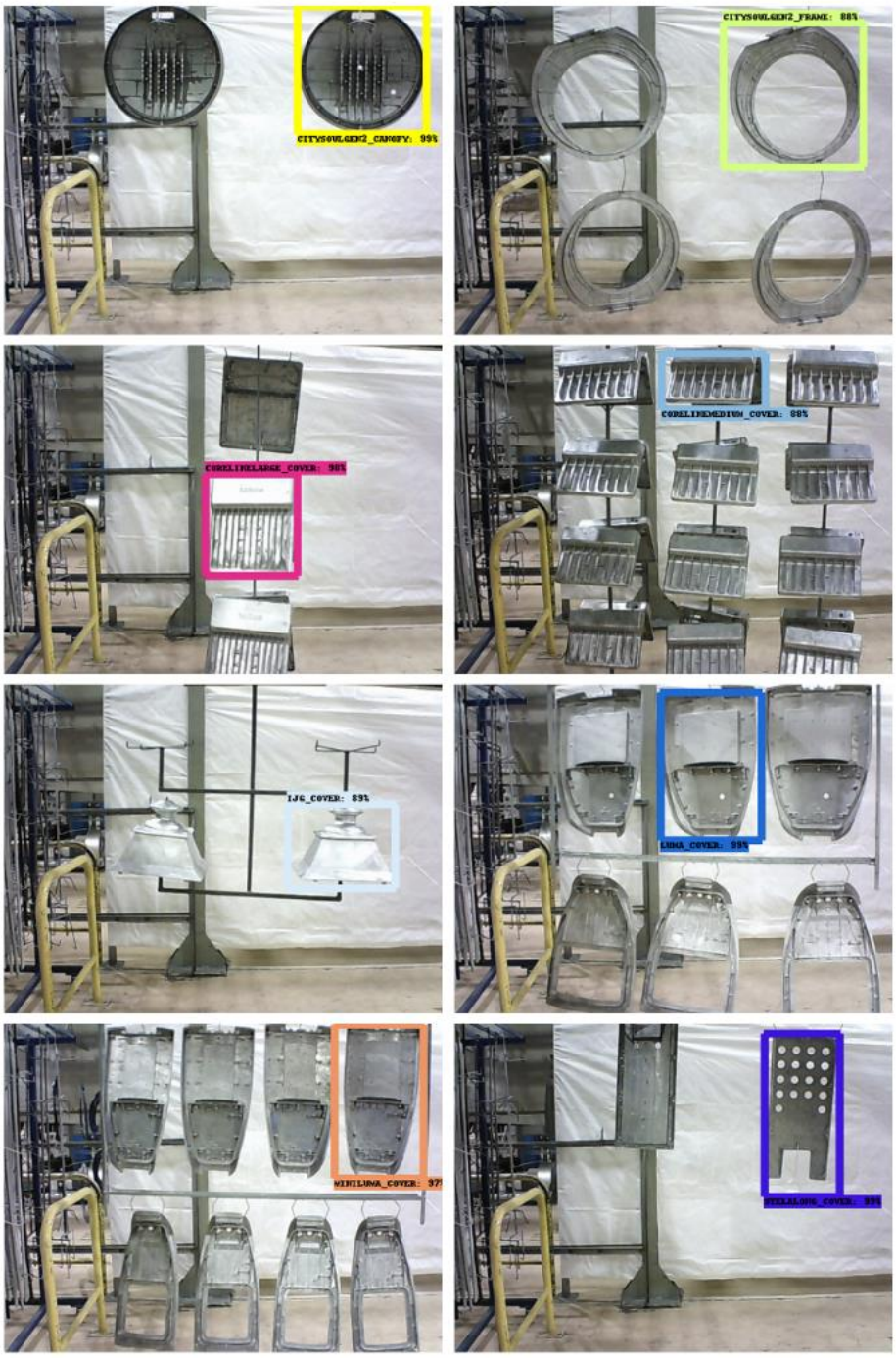

ILUSTRACIÓN 71: IMÁGENES CLASIFICADAS EN TIEMPO REAL EN ENTORNO DE PRODUCCIÓN

5. Conclusiones y líneas futuras:

5.1. Conclusiones y contribuciones del trabajo:

Desde hace varios años, la empresa Signify Manufacturing Spain S.L. ha estado investigando sobre soluciones industriales de visión artificial con la finalidad de detectar las perchas que circulan por su instalación sin éxito, debido al gran costo económico de las aplicaciones ofrecidas por las empresas líderes en este tipo de soluciones, como el gran costo de recolección de las imágenes necesarias para emplearlas como datos de comparación en los modelos ofrecidos por estas compañías.

Con la finalidad de encontrar una solución potente y económica que proporcione los requerimientos de la compañía, se propuso el diseño e implementación de un sistema de detección de bajo coste y de bajo coste de implementación y actualización ante modificaciones. Como añadido, la compañía requería que el sistema fuese lo más visual posible, permitiendo mostrar a las visitas o a los propios empleados de la fábrica el potencial de innovación de la compañía, disponiendo de un entorno inteligente dentro de la factoría.

Este sistema diseñado, presenta soluciones a estas tres condiciones propuestas.

Para solucionar el problema de recolectar grandes cantidades de datos, se propuso emplear técnicas de Data Augmentation, tomando como punto de partida las investigaciones realizadas por Medina Quero, J. (tutor del TFM) y otros en su investigación *"Straightforward Recognition of Daily Objects in Smart Environments from Wearable Vision Sensor"* (Medina Quero, 2018), en la cual proponen una solución de diseño de modelos de detección de objetos a través de unas pocas fotos de los objetos a detectar y un video del fondo, generando imágenes a partir de estas entradas aplicando sobre ellas transformaciones para generar las entradas necesarias del modelo de una manera rápida.

En cuanto al bajo coste económico de implantación requerido, en comparación con las ofertas industriales comerciales, se ha propuesto la utilización de dispositivos de bajo coste como Arduino, sensores comerciales, Raspberry Pi, webcams y un ordenador reutilizado de antiguos proyectos, sin la necesidad de caros dispositivos hardware para tratamiento de imágenes. Los gastos hardware totales empleados durante la elaboración del proyecto no superan los 200€, sin contar el coste del ordenador puesto que se reutilizó uno en desuso. En caso de tenerlo en cuenta, el coste asciende a 800€, muy lejos de las soluciones comerciales que rondan los 5000€ en hardware (sin contar la parte software y de soporte).

Finalizando con el último requerimiento de la compañía, el cual indicaba la necesidad de crear un entorno inteligente y visual dentro de la factoría para mostrar el potencial innovador, se ha implementado el clasificador en una instalación que combina dispositivos IoT. Para la parte de captación de imágenes y clasificación se ha empleado un sensor industrial y una placa Arduino conectadas a un ordenador con S.O. Linux, el cual dispone de un interfaz de usuario diseñado en la elaboración del TFM, que a través de las dos cámaras webcam obtiene las clasificaciones a través del modelo diseñado con la API Object Detection de TensorFlow.

Para la parte de visualización, se ha diseñado un interfaz de usuario compatible con Raspberry Pi, el cual obtiene los datos almacenados por el ordenador en la base de datos SQL Server de la fábrica, permitiendo así total flexibilidad para instalar el sistema en la ubicación deseada por la compañía.

Desde el punto de vista de las innovaciones propuestas por el trabajo, centrándonos en los datos obtenidos a través del filtrado del confidence, se concluye que la capacidad del sistema de permitir modificar los valores de filtrado de cada una de las clases ayudará en gran medida a dotar al sistema de flexibilidad ante ligeros cambios en unas u otras piezas, permitiendo modificar el filtro de clases individualmente sin afectar al conjunto, decidiendo si es más conveniente obtener mejores valores de precisión subiendo dicho valor, u optando por obtener mejor Recall al disminuirlo.

Observando los datos de salida obtenidos sobre imágenes reales, se observa la buena capacidad clasificatoria del modelo diseñado siendo un modelo tan complicado de realizar para la gran cantidad de clases presentes con formas similares, sin el uso de gran cantidad de imágenes reales, sino que solo ha sido requisito inicial disponer de unas 20-25 muestras de cada tipo de luminaria. Como conclusión se puede afirmar que la metodología empleada para generar datos con Data Augmentation es válida para aplicarla en entornos de producción.

Por otro lado, es importante destacar la alta labor de ingeniería desplegada ante la necesidad que ha surgido durante el desarrollo del trabajo de unificar diferentes técnicas:

- Para el análisis previo de los datos obteniéndolos directamente de la base de datos de la fábrica con programación tradicional basados en lenguaje Java y SQL, así como su posterior análisis a través de herramientas estadísticas como Minitab.
- Para la generación de los datos aumentados con herramientas basadas en Python y diferentes librerías como OpenCV.

- Para su implementación final empleando sistemas operativos diferentes como Windows 10 y Linux, dispositivos IoT como Arduino y Raspberry Pi, y software específico basado en lenguaje Java para permitir al usuario final usar el sistema de una forma fácil.

5.2. Líneas de trabajo futuro:

Finalizada la versión piloto del proyecto, aparecen diferentes vías de trabajo futuras para mejorar el sistema de detección como para emplear los datos obtenidos para mejorar el proceso de pintura de la fábrica.

- **Mejora del sistema de captación:**

Una de las primeras etapas en la mejora de la versión piloto es la de ampliar el modelo de detección de objetos para incluir todos los objetos que pueden circular por la instalación. Como se comentó en la parte de desarrollo, de las 154 posibles combinaciones detectadas a partir de los 447 materiales diferentes, se obtiene un total de 78 combinaciones posibles que por tamaño y por su especificidad (el resto de las combinaciones son materiales que comparten diferentes familias) son las que realmente es necesario detectar.

De todas estas posibilidades, la versión piloto es capaz de detectar las 22 más importantes (90% de la producción). La inclusión del resto de luminarias es claramente el siguiente punto de trabajo.

- **Empleo de los datos obtenidos para mejorar el proceso:**

Otra de las posibles líneas de trabajo que aparecen tras la realización de este proyecto es el uso de los datos para mejorar la instalación de pintura, diseñando un sistema de identificación de perchas a lo largo de la instalación empleando dispositivos IoT con sensores como TAG's u otro tipo de sensores, que sean capaces de informar a los pintores, que luminaria está incluida en cada percha justo antes de ser pintadas, reduciendo así el valor no añadido del trabajo de los pintores identificando visualmente el tipo de luminaria presente.

Disponer de los datos de las luminarias presentes en cada percha abre otra vía de desarrollo como es programar de manera automática las máquinas de la instalación, comunicando sus PLC's con la base de datos de la fábrica, siendo capaz de decir en cada momento el programa necesario que tiene que tener cada parte de la instalación

(curvas del horno, programas de pintado…), eliminando así el valor no añadido generado por los operarios de dicha instalación.

Este desarrollo contribuiría además al requisito de la compañía de tener un entorno inteligente, ampliándolo desde el actual diseñado con la clasificación a través de visión artificial, hasta una instalación completa de pintado inteligente.

- **Mejora del sistema IoT:**

Una mejora clara al sistema de dispositivos IoT diseñado es la eliminación de cables, permitiendo al sistema más flexibilidad a la hora de desear mover la instalación en el futuro. Para ello, se podrían emplear módulos Zigbee de Arduino (Faludi, 2010) o de Raspberry Pi para comunicar ambos dispositivos con el ordenador, si necesidad de emplear cables para la transmisión de información, ni redes Wifi, puesto que uno de los problemas presentados durante la implantación del proyecto por parte de la compañía ha sido el uso de la red interna con dispositivos Linux, los cuales no se encuentran dentro del dominio de la empresa.

ILUSTRACIÓN 72: MÓDULO ZIGBEE DE ARDUINO

Referencias

Abadi, M. B. (2016, November). *Tensorflow: a system for large-scale machine learning. .* In OSDI (Vol. 16, pp. 265-283).

Adobe. (21/02/2019). *Photoshop*. Obtenido de https://www.photoshop.com/

Aguirre, Á. G. (2014). *Sistema embebido de bajo costo para visión artificial.* Scientia et technica, 19(2), 163-173.

Amro. (21/02/2019). *GitHub: GrabCut segmentation demo*. Obtenido de http://amroamroamro.github.io/mexopencv/opencv/grabcut_demo_gui.html

Anaconda. (21/02/2019). *Download Anaconda Distribution*. Obtenido de https://www.anaconda.com/download/

BackgroundBurner. (21/02/2019). *Eliminar los fondos a partir de imágenes*. Obtenido de https://burner.bonanza.com/

Banzi, M. &. (2014). *Getting started with Arduino: the open source electronics prototyping platform.* Maker Media, Inc..

Basogain, X. O. (2007). *Realidad Aumentada en la Educación: una tecnología emergente.* Escuela Superior de Ingeniería de Bilbao, EHU. .

Bay, H. E. (2008). *Speeded-up robust features (SURF).* Computer vision and image understanding, 110(3), 346-359.

Bradski, G. &. (2008). *Learning OpenCV: Computer vision with the OpenCV library.* "O'Reilly Media, Inc.".

Brownlee, J. (2017, December). *A Gentle Introduction to Transfer Learning for Deep Learning*. Obtenido de https://machinelearningmastery.com/transfer-learning-for-deep-learning/

Brownlee, J. (2018). *Train Faster, Reduce Overfitting, and Make Better Predictions.* Better Deep Learning.

Canva. (21/02/2019). *Diseña lo que quieras*. Obtenido de https://www.canva.com/

Cárdenas, J. A.-O. (2015). *Diseño de un algoritmo de corrección automática de posición para el proceso de perforado PCB, empleando técnicas de visión artificial. .* Revista de Investigación, Desarrollo e Innovación, 5(2), 107-118.

Castroviejo, R. C. (2008). *Caracterización y cuantificación autmatizadas de menas metálicas mediante visión artificial: proyecto cameva.* In Actos del XIII Congreso Latino Americano de Geologia.

Ciresan, D. C. (2011, July). *Flexible, high performance convolutional neural networks for image classification.* In IJCAI Proceedings-International Joint Conference on Artificial Intelligence (Vol. 22, No. 1, p. 1237).

Ciresan, D., Meier, U., Masci, J., Gambardella, L., & Schmidhuber, J. H. (2011). *Performance Neural Networks for Visual Object Classification.* arXiv:1102.0183.

ClippingMagic. (21/02/2019). *Instantly Remove Image Backgrounds Online*. Obtenido de https://clippingmagic.com/

Cognex. (21/02/2019). *Cognex*. Obtenido de https://www.cognex.com/

Condori Arias, E. F. (2013). *Reconocimiento y clasificación de objetos usando inteligencia artificial basada en SVM y visión estereoscópica.* Universidad Nacional de Ingeniería.

Constante, P. G. (2016). Artificial Vision Techniques to Optimize Strawberry's Industrial Classification. *IEEE Latin America Transactions,* , 14(6), 2576-2581.

Davies, E. R. (2004). *Machine vision: theory, algorithms, practicalities.* Elsevier.

De Dios Santos, J. (2018). *Detecting Pikachu on Android using Tensorflow Object Detection*. Obtenido de https://towardsdatascience.com/detecting-pikachu-on-android-using-tensorflow-object-detection-15464c7a60cd

Download the Arduino IDE. (21/02/2019). Obtenido de https://www.arduino.cc/en/Main/Software

Dwibedi, D. M. (2017, October). *Cut, paste and learn: Surprisingly easy synthesis for instance detection.* In The IEEE international conference on computer vision (ICCV).

Dwivedi, P. (21/02/2019). *Analyze a Soccer game using Tensorflow Object Detection and OpenCV*. Obtenido de https://towardsdatascience.com/analyse-a-soccer-game-using-tensorflow-object-detection-and-opencv-e321c230e8f2

EdjeElectronics. (21/02/2019). *How To Train an Object Detection Classifier for Multiple Objects Using TensorFlow (GPU) on Windows 10*. Obtenido de https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10

Faludi, R. (2010). *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing.* " O'Reilly Media, Inc.".

Fawzi, A. S. (2016). *Adaptive data augmentation for image classification.* In 2016 Ieee International Conference On Image Processing (Icip) (No. EPFL-CONF-218496, pp. 3688-3692). Ieee.

Fukushima, K. &. (1982). *Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition.* Springer, Berlin, Heidelberg: In Competition and cooperation in neural nets (pp. 267-285).

Galipienso, M. I. (2003). *Inteligencia artificial: modelos, técnicas y áreas de aplicación.* Editorial Paraninfo.

Georgakis, G. M. (2017). *Synthesizing training data for object detection in indoor scenes.* arXiv preprint arXiv:1702.07836.

Gidaris, S. &. (2015). *Object detection via a multi-region and semantic segmentation-aware cnn model.* In Proceedings of the IEEE International Conference on Computer Vision (pp. 1134-1142).

Girshick, R. (2015). *Fast r-cnn.* In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

GitHub: Installation Object detection. (21/02/2019). Obtenido de https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md

GitHub: Tensorflow Models. (21/02/2019). Obtenido de https://github.com/tensorflow/models

GitHub: Tensorflow Object Detection API. (21/02/2019). Obtenido de https://github.com/tensorflow/models/tree/master/research/object_detection

Goodfellow, I. B. (2016). *Deep learning (adaptive computation and machine learning series).* Adaptive Computation and Machine Learning series, 800.

Gross, B. (1992). La inteligencia artificial y su aplicación en la enseñanza. págs. Comunicación, lenguaje y educación, 4(13), 73-80.

Gubbi, J. B. (2013). *Internet of Things (IoT): A vision, architectural elements, and future directions.* Future generation computer systems, 29(7), 1645-1660.

He, W. Y. (2014). *Developing vehicular data cloud services in the IoT environment.* IEEE Transactions on Industrial Informatics, 10(2), 1587-1595.

Huang, J. R. (2017). *Tensorflow object detection api.* Obtenido de Code: github. com/tensorflow/models/tree/master/object detection.

Hui, J. (2018). *Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)*. Obtenido de https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

Intelligent. (2018). *Deep learning & Convolutional Neuronal Network: qué es y en qué consiste*. Obtenido de https://itelligent.es/es/deep-learning-convolutional-neuronal-network-cnn-consiste/

Karpathy, A. (2016). *Cs231n convolutional neural networks for visual recognition.* Neural networks, 1.

Kelly, S. D. (2013). *Towards the implementation of IoT for environmental condition monitoring in homes.* IEEE Sensors Journal, 13(10), 3846-3853.

Krizhevsky, A. S. (2012). *Imagenet classification with deep convolutional neural networks.* In Advances in neural information processing systems (pp. 1097-1105).

Krizhevsky, A. S. (2012). *Imagenet classification with deep convolutional neural networks.* In Advances in neural information processing systems (pp. 1097-1105).

LeCun, Y. B. (1998). *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86(11), 2278-2324.

Lee, I. &. (2015). *The Internet of Things (IoT): Applications, investments, and challenges for enterprises. .* Business Horizons, 58(4), 431-440.

Liu, W. A. (2016, October). *Ssd: Single shot multibox detector.* In European conference on computer vision (pp. 21-37). Springer, Cham.

McCarthy, J. (1998). *What is artificial intelligence?.*

McLaughlin, N. D. (2015, August). *Data-augmentation for reducing dataset bias in person re-identification.* In Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on (pp. 1-6). IEEE.

Medina Quero, J. C. (2018). *Straightforward Recognition of Daily Objects in Smart Environments from Wearable Vision Sensor.*

Merchán, F. G. (2016). Mejoras en el Entrenamiento de Esquemas de Deteccion de Sonrisas basados en AdaBoost. *I+D Tecnológico, 10(2), 17-30.*

Microsoft. (21/02/2019). *Quitar el fondo de una imagen*. Obtenido de https://support.office.com/es-es/article/quitar-el-fondo-de-una-imagen-c0819a62-6844-4190-8d67-6fb1713a12bf

Olafenwa, M. (2018). *Object Detection with 10 lines of code*. Obtenido de https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606

Olivas, E. S. (2009). *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques.* IGI Global.

OpenCV. (21/02/2019). *Install OpenCV-Python in Ubuntu*. Obtenido de https://docs.opencv.org/3.4.1/d2/de6/tutorial_py_setup_in_ubuntu.html

Oracle. (21/02/2019). *Java SE Development Kit 11 Downloads*. Obtenido de https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html

Pastor, J. (2018). *Qué es la inteligencia artificial*. Obtenido de https://www.xataka.com/robotica-e-ia/que-inteligencia-artificial

Peters, T. (2010). *The zen of python.* In Pro Python (pp. 301-302). Apress.

PhotoScissors. (21/02/2019). *Instantly Remove Background From Your Photos Online!* Obtenido de https://online.photoscissors.com/

Pollo Cattaneo, M. F. (2016, May). *Implementación de sistemas inteligentes para la asistencia a alumnos y docentes de la carrera de Ingeniería en Sistemas de Información.* In XVIII Workshop de Investigadores en Ciencias de la Computación (WICC 2016, Entre Ríos, Argentina).

Raj, B. (2018). *A Simple Guide to the Versions of the Inception Network*. Obtenido de https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Raj, B. (2018). *Data Augmentation | How to use Deep Learning when you have Limited Data—Part 2*. Obtenido de https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Raj, B. (21/02/2019). *How to play Quidditch using the TensorFlow Object Detection API*. Obtenido de https://medium.freecodecamp.org/how-to-play-quidditch-using-the-tensorflow-object-detection-api-b0742b99065d

Redmon, J. D. (2016). *You only look once: Unified, real-time object detection.* In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

Ren, S. H. (2015). *Faster r-cnn: Towards real-time object detection with region proposal networks.* In Advances in neural information processing systems (pp. 91-99).

Richardson, M. &. (2012). *Getting started with raspberry PI.* " O'Reilly Media, Inc.".

Rosebrock, A. (2017). *Keras and deep learning on the Raspberry Pi*. Obtenido de https://www.pyimagesearch.com/2017/12/18/keras-deep-learning-raspberry-pi/

Rother, C. K. (2004). *Grabcut: Interactive foreground extraction using iterated graph cuts.* In ACM transactions on graphics (TOG) (Vol. 23, No. 3, pp. 309-314). ACM. Obtenido de OpenCV.

SACHAN, A. (21/02/2019). *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD*. Obtenido de CV-Tricks.com: https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

Sanchez, L. M. (2014). *SmartSantander: IoT experimentation over a smart city testbed.* Computer Networks, 61, 217-238.

Schalkoff, R. J. (1997). *Artificial neural networks (Vol. 1). .* New York: McGraw-Hill.

Signify. (2018). *La plataforma de IoT Interact*. Obtenido de https://www.signify.com/es-es/innovation/iot-platform

Signify. (21/02/2019). *CityTouch: Sistema de gestión de la iluminación*. Obtenido de http://www.lighting.philips.com.ar/systems/lighting-systems/citytouch

Signify. (21/02/2019). *Introducing LiFi*. Obtenido de https://www.signify.com/global/innovation/lifi

Szegedy, C. I. (2017). *Inception-v4, inception-resnet and the impact of residual connections on learning.* In AAAI (Vol. 4, p. 12).

Szegedy, C. V. (2016). *Rethinking the inception architecture for computer vision.* In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).

Tagliaferri, L. (2018, September). *How To Install Anaconda on Ubuntu 18.04 [Quickstart]*. Obtenido de https://www.digitalocean.com/community/tutorials/how-to-install-anaconda-on-ubuntu-18-04-quickstart

Techopedia. (21/02/2019). *Aumento de datos*. Obtenido de https://www.techopedia.com/definition/28033/data-augmentation

Tensorflow. (21/02/2019). Obtenido de https://www.tensorflow.org/

Tensorflow. (21/02/2019). *Install TensorFlow*. Obtenido de https://www.tensorflow.org/install/

Tensorflow. (21/02/2019). *Tensorboard: Visualizing Learning*. Obtenido de https://www.tensorflow.org/guide/summaries_and_tensorboard

Trivedi, C. (21/02/2019). *Using Tensorflow Object Detection to control first-person shooter games*. Obtenido de https://towardsdatascience.com/using-tensorflow-object-detection-to-control-first-person-shooter-games-c2c7f1daf2e9

VenkatesH, B. (2018). *TensorFlow object detection with custom objects*. Obtenido de https://medium.com/coinmonks/tensorflow-object-detection-with-custom-objects-34a2710c6de5

Vilasenor, M. M. (2015). *Clasificación de objetos en imágenes usando SIFT.*

Zeiler, M. D. (2014, September). *Visualizing and understanding convolutional networks.* Springer, Cham.: In European conference on computer vision (pp. 818-833).

Zhu, X. L. (2017). *Data augmentation in emotion classification using generative adversarial networks. .* arXiv preprint arXiv:1711.00648.

Anexos:

A1. Datos del análisis previo a la creación del modelo:

Todos los datos relativos al análisis se encuentran en el archivo TFM_DatosAnalisis.xlsx adjunto en la documentación de este trabajo.

A2. RenombrarImagenes_Final.py:

Código Python empleado para renombrar las imágenes al formato deseado:

```
#!/usr/bin/env python

'''
================================================================================
Programa que renombra las imagenes
README FIRST:
    Al iniciar, el programa buscará todas las imagenes que se encuentren
en el directorio actual. Todo aquello que tenga .png se renombrara con el nombre de
la carpeta actual
================================================================================
'''

print(__doc__)

import cv2
import os
from os import scandir, getcwd
import shutil

#Se elimina la carpeta de salida si existe
try:
    shutil.rmtree("Renombrados")
except:
    print("No existe ya")

#Se crea la carpeta de salida vacia
#os.makedirs("Renombrados", exist_ok=True)

#Se obtiene el nombre de la carpeta
directorio = os.getcwd().split("\\")
nombre_carpeta = directorio[len(directorio)-1]
print(nombre_carpeta)

#Se cogen todos los nombres de los archivos de la carpeta
def ls(ruta = getcwd()):
    return [arch.name for arch in scandir(ruta) if arch.is_file()]

lista_arq = ls(getcwd())

#Se recorren todos los archivos, y si contienen .png se renombra
contador = 0
for i in range(int(len(lista_arq))):
    if ".png" in lista_arq[i]:
        contador = contador+1
        #imagen = cv2.imread(lista_arq[i])
```

```
        numero = "_0000"
        if contador < 10:
            numero = "_000" + str(contador)
        elif contador < 100:
            numero = "_00" + str(contador)
        elif contador < 1000:
            numero = "_0" + str(contador)
        shutil.copyfile(lista_arq[i],nombre_carpeta + numero + ".png")
        print(lista_arq[i] + " -> " + nombre_carpeta + numero + ".png")
        os.remove(lista_arq[i])

print("FINALIZADO EL PROCESO")
```

A3. Ejemplo desarrollo detección:

Todos los datos relativos a la creación de la información desarrollada en el apartado 4.3 d) y e) se incluyen en la carpeta TFM_EjemploDesarrolloDetección adjunto en la documentación de este trabaj

A4. EjecuciónSuperponerImagenEnVideo.txt:

```
(base) C:\Users\user\Desktop\SalidaBordes>python SuperponerImagenEnVideo.py

================================================================================
Programa que superpone piezas en frames de un video de fondo
README FIRST:
    Al iniciar, tras cargar librerias e inicializar funciones, realiza las
    siguientes acciones:
      1 - Crea la carpeta Salida
      2 - Coge todos los archivos que hay en la carpeta actual
      3 - Comprueba si existe el video en formato mp4
      4 - Se crean los frames del video
      5 - Recorre una a una todas las imagenes en formato .png y va
          superponiendolas sobre los frames del video
================================================================================

Archivo csv no existe
No existe ya
No existe ya
No existe ya
No existe ya
Video: vid7.mp4
Creando... Video/F000.png
Creando... Video/F030.png
Creando... Video/F060.png
Creando... Video/F090.png
Creando... Video/F120.png
Creando... Video/F150.png
Creando... Video/F180.png
Creando... Video/F210.png
Creando... Video/F240.png
Creando... Video/F270.png
Creando... Video/F300.png
Creando... Video/F330.png
Creando... Video/F360.png
Creando... Video/F390.png
Creando... Video/F420.png
Creando... Video/F450.png
Creando... Video/F480.png
Creando... Video/F510.png
Creando... Video/F540.png
LUMA_FRAME_0003.png
Creando imagen... train/0.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F000.png
Creando imagen... train/1.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F030.png
Creando imagen... train/2.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F060.png
Creando imagen... train/3.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F090.png
Creando imagen... train/4.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F120.png
Creando imagen... train/5.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F150.png
Creando imagen... train/6.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F180.png
Creando imagen... train/7.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F210.png
Creando imagen... train/8.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F240.png
Creando imagen... train/9.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F270.png
Creando imagen... train/10.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F300.png
Creando imagen... train/11.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F330.png
Creando imagen... train/12.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F360.png
Creando imagen... train/13.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F390.png
Creando imagen... train/14.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F420.png
Creando imagen... train/15.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F450.png
Creando imagen... train/16.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F480.png
Creando imagen... train/17.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F510.png
Creando imagen... train/18.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_0_F540.png
Creando imagen... train/19.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_90_F000.png
Creando imagen... train/20.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_90_F030.png
Creando imagen... train/21.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_90_F060.png
...
Creando imagen... train/176.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F210.png
Creando imagen... train/177.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F240.png
Creando imagen... train/178.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F270.png
Creando imagen... train/179.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F300.png
Creando imagen... train/180.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F330.png
Creando imagen... train/181.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F360.png
Creando imagen... train/182.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F390.png
Creando imagen... train/183.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F420.png
Creando imagen... train/184.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F450.png
Creando imagen... train/185.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F480.png
Creando imagen... train/186.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F510.png
Creando imagen... train/187.png h: 480, w: 270 -> Real: train/LUMA_FRAME_0003_RANDOM9_F540.png
Creando archivo csv...
HECHO TRAIN!
Creando archivo csv...
HECHO TEST!
```

A5. EjecucciónTFRecord.txt:

```
(base)
C:\Users\user\Desktop\SalidaBordes>python
generate_tfrecord.py
train/0.png
train/1.png
train/10.png
train/100.png
train/101.png
train/102.png
train/103.png
train/104.png
train/105.png
train/106.png
train/107.png
train/108.png
train/109.png
train/11.png
train/110.png
train/111.png
train/112.png
train/113.png
train/114.png
train/115.png
train/116.png
train/117.png
train/118.png
train/119.png
train/12.png
train/120.png
train/121.png
train/122.png
train/123.png
train/124.png
train/125.png
train/126.png
train/127.png
train/128.png
train/129.png
train/13.png
train/130.png
train/131.png
train/132.png
train/133.png
train/134.png
train/135.png
train/136.png
train/137.png
train/138.png
train/139.png
train/14.png
train/140.png
train/141.png
train/142.png
train/143.png
train/144.png
train/145.png
train/146.png
train/147.png
train/148.png
train/149.png
train/15.png
train/150.png
train/151.png
...
train/75.png
train/76.png
train/77.png
train/78.png
train/79.png
train/8.png
train/80.png
train/81.png
train/82.png
train/83.png
train/84.png
train/85.png
train/86.png
train/87.png
train/88.png
train/89.png
train/9.png
train/90.png
train/91.png
train/92.png
train/93.png
train/94.png
train/95.png
train/96.png
train/97.png
train/98.png
train/99.png
Successfully created the TFRecords:
train.record
test/175.png
test/44.png
test/69.png
Successfully created the TFRecords:
test.record
```

A6. Object-detection.pbtxt:

```
item {
  id: 1
  name: 'AVOVGEN2_FRAME'
}
item {
  id: 2
  name: 'CITYSOULGEN2_CANOPY'
}
item {
  id: 3
  name: 'CITYSOULGEN2_FRAME'
}
item {
  id: 4
  name: 'CLASSICSTREET_COVER'
}
item {
  id: 5
  name: 'CLASSICSTREET_FRAME'
}
item {
  id: 6
  name: 'CLEARFLOOD_COVER'
}
item {
  id: 7
  name: 'CLEARFLOOD_FRAME'
}
item {
  id: 8
  name: 'CORELINELARGE_COVER'
}
item {
  id: 9
  name: 'CORELINEMEDIUM_COVER'
}
item {
  name: 'UNISTREET_HOUSING'
}
  id: 10
  name: 'DIGISTREETCATENARY_HOUSING'
}
item {
  id: 11
  name: 'IJG_COVER'
}
item {
  id: 12
  name: 'LUMA_COVER'
}
item {
  id: 13
  name: 'LUMA_FRAME'
}
item {
  id: 14
  name: 'MINILUMA_COVER'
}
item {
  id: 15
  name: 'MINILUMA_FRAME'
}
item {
  id: 16
  name: 'STELALONG_COVER'
}
item {
  id: 17
  name: 'STELASQUARE_COVER'
}
item {
  id: 18
  name: 'TUBEPOINT_HOUSING'
}
item {
  id: 19
```

A7. Obj-inception.config:

```
# Faster R-CNN with Inception v2, configured for Oxford-IIIT Pets Dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
# should be configured.

model {
  faster_rcnn {
    num_classes: 19
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.01
        }
      }
    }
    first_stage_nms_score_threshold: 0.0
    first_stage_nms_iou_threshold: 0.7
    first_stage_max_proposals: 300
    first_stage_localization_loss_weight: 2.0
    first_stage_objectness_loss_weight: 1.0
    initial_crop_size: 14
    maxpool_kernel_size: 2
    maxpool_stride: 2
    second_stage_box_predictor {
      mask_rcnn_box_predictor {
        use_dropout: false
        dropout_keep_probability: 1.0
        fc_hyperparams {
          op: FC
          regularizer {
            l2_regularizer {
              weight: 0.0
            }
          }
          initializer {
```

```
            variance_scaling_initializer {
              factor: 1.0
              uniform: true
              mode: FAN_AVG
            }
          }
        }
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 900000
            learning_rate: .00002
          }
          schedule {
            step: 1200000
            learning_rate: .000002
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }

  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "/media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/10-
Tensforflow/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_
28/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
 data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
```

```
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
 data_augmentation_options {
    random_pixel_value_scale {
    }
  }
data_augmentation_options {
    random_adjust_brightness {
    }
  }
  data_augmentation_options {
    random_adjust_contrast {
    }
  }
  data_augmentation_options {
    random_adjust_saturation {
    }
  }
  data_augmentation_options {
    random_jitter_boxes {
    }
  }
  data_augmentation_options {
    random_crop_image {
    }
  }
}


train_input_reader: {
  tf_record_input_reader {
    input_path: "/media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/12-DATOS_BUENOS-
Imagenes_con_fondo/train.record"
  }
  label_map_path: "/media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/14-
Object_detection_config2/object-detection.pbtxt"
}

eval_config: {
  num_examples: 2000
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "/media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/12-DATOS_BUENOS-
Imagenes_con_fondo/test.record"
  }
  label_map_path: "/media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/14-
Object_detection_config2/object-detection.pbtxt"
  shuffle: false
  num_readers: 1
}
```

A8. Resultado de las 31 primeras etapas del primer entrenamiento:

```
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
W1224 12:21:25.428649 140161595688768 tf_logging.py:125] From
/home/david/anaconda3/lib/python3.6/site-
packages/tensorflow/contrib/slim/python/slim/learning.py:737: Supervisor.__init__ (from
tensorflow.python.training.supervisor) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
2018-12-24 12:21:26.410077: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
INFO:tensorflow:Restoring parameters from /media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/10-
Tensforflow/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.ck
pt
I1224 12:21:27.941079 140161595688768 tf_logging.py:115] Restoring parameters from
/media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/10-
Tensforflow/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.ck
pt
INFO:tensorflow:Running local_init_op.
I1224 12:21:28.542055 140161595688768 tf_logging.py:115] Running local_init_op.
INFO:tensorflow:Done running local_init_op.
I1224 12:21:28.778316 140161595688768 tf_logging.py:115] Done running local_init_op.
INFO:tensorflow:Starting Session.
I1224 12:21:33.976604 140161595688768 tf_logging.py:115] Starting Session.
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
I1224 12:21:34.135071 140159832004352 tf_logging.py:115] Saving checkpoint to path
training/model.ckpt
INFO:tensorflow:Starting Queues.
I1224 12:21:34.138917 140161595688768 tf_logging.py:115] Starting Queues.
INFO:tensorflow:global_step/sec: 0
I1224 12:21:36.284593 140159613925120 tf_logging.py:159] global_step/sec: 0
INFO:tensorflow:Recording summary at step 0.
I1224 12:21:41.200845 140159605532416 tf_logging.py:115] Recording summary at step 0.
INFO:tensorflow:global step 1: loss = 1.9927 (12.242 sec/step)
I1224 12:21:46.488050 140161595688768 tf_logging.py:115] global step 1: loss = 1.9927 (12.242
sec/step)
INFO:tensorflow:global step 2: loss = 2.0384 (3.275 sec/step)
I1224 12:21:49.875236 140161595688768 tf_logging.py:115] global step 2: loss = 2.0384 (3.275
sec/step)
INFO:tensorflow:global step 3: loss = 1.9582 (3.761 sec/step)
I1224 12:21:53.636834 140161595688768 tf_logging.py:115] global step 3: loss = 1.9582 (3.761
sec/step)
INFO:tensorflow:global step 4: loss = 1.7209 (3.594 sec/step)
I1224 12:21:57.231038 140161595688768 tf_logging.py:115] global step 4: loss = 1.7209 (3.594
sec/step)
INFO:tensorflow:global step 5: loss = 1.6420 (3.174 sec/step)
I1224 12:22:00.406568 140161595688768 tf_logging.py:115] global step 5: loss = 1.6420 (3.174
sec/step)
...
I1224 12:23:31.901461 140161595688768 tf_logging.py:115] global step 31: loss = 1.5640 (4.308
sec/step)
INFO:tensorflow:global_step/sec: 0.259743
I1224 12:23:35.633095 140159613925120 tf_logging.py:159] global_step/sec: 0.259743
INFO:tensorflow:Recording summary at step 31.
I1224 12:23:36.994746 140159605532416 tf_logging.py:115] Recording summary at step 31.
```

A9. Resultado Export_Inference_Graph.py:

```
python3 object_detection/export_inference_graph.py --input_type image_tensor --
pipeline_config_path=/media/david/TOSHIBA\ EXT/WINDOWS10/Master/7-TFM/21-
PruebaValidacionModelo/obj_inception.config --trained_checkpoint_prefix /media/david/TOSHIBA\
EXT/WINDOWS10/Master/7-TFM/21-PruebaValidacionModelo/Training/model.ckpt-3896 --
output_directory /media/david/TOSHIBA\ EXT/WINDOWS10/Master/7-TFM/21-
PruebaValidacionModelo/Inference_Graph/
WARNING:tensorflow:From /media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/10-
Tensforflow/models/research/object_detection/predictors/heads/box_head.py:93: calling
reduce_mean (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be
removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
W1224 16:20:58.442481 140428570916672 tf_logging.py:125] From /media/david/TOSHIBA
EXT/WINDOWS10/Master/7-TFM/10-
Tensforflow/models/research/object_detection/predictors/heads/box_head.py:93: calling
reduce_mean (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be
removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From /media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/10-
Tensforflow/models/research/object_detection/exporter.py:328: get_or_create_global_step (from
tensorflow.contrib.framework.python.ops.variables) is deprecated and will be removed in a
future version.
Instructions for updating:
Please switch to tf.train.get_or_create_global_step
W1224 16:21:00.166271 140428570916672 tf_logging.py:125] From /media/david/TOSHIBA
EXT/WINDOWS10/Master/7-TFM/10-Tensforflow/models/research/object_detection/exporter.py:328:
get_or_create_global_step (from tensorflow.contrib.framework.python.ops.variables) is
deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.get_or_create_global_step
WARNING:tensorflow:From /media/david/TOSHIBA EXT/WINDOWS10/Master/7-TFM/10-
Tensforflow/models/research/object_detection/exporter.py:482: print_model_analysis (from
tensorflow.contrib.tfprof.model_analyzer) is deprecated and will be removed after 2018-01-01.
Instructions for updating:
Use `tf.profiler.profile(graph, run_meta, op_log, cmd, options)`. Build `options` with
`tf.profiler.ProfileOptionBuilder`. See README.md for details
W1224 16:21:00.177620 140428570916672 tf_logging.py:125] From /media/david/TOSHIBA
EXT/WINDOWS10/Master/7-TFM/10-Tensforflow/models/research/object_detection/exporter.py:482:
print_model_analysis (from tensorflow.contrib.tfprof.model_analyzer) is deprecated and will be
removed after 2018-01-01.
Instructions for updating:
Use `tf.profiler.profile(graph, run_meta, op_log, cmd, options)`. Build `options` with
`tf.profiler.ProfileOptionBuilder`. See README.md for details
264 ops no flops stats due to incomplete shapes.
Parsing Inputs...
Incomplete shape.

=========================Options=============================
-max_depth                  10000
-min_bytes                  0
-min_peak_bytes             0
-min_residual_bytes         0
-min_output_bytes           0
-min_micros                 0
-min_accelerator_micros     0
-min_cpu_micros             0
-min_params                 0
-min_float_ops              0
-min_occurrence             0
-step                       -1
-order_by                   name
-account_type_regexes       _trainable_variables
-start_name_regexes         .*
-trim_name_regexes          .*BatchNorm.*
-show_name_regexes          .*
```

```
-hide_name_regexes
-account_displayed_op_only  true
-select                     params
-output                     stdout:

==================Model Analysis Report======================
Incomplete shape.

Doc:
scope: The nodes in the model graph are organized by their names, which is hierarchical like
filesystem.
param: Number of parameters (in the Variable).

Profile:
node name | # parameters
_TFProfRoot (--/12.87m params)
  Conv (--/2.65m params)
    Conv/biases (512, 512/512 params)
    Conv/weights (3x3x576x512, 2.65m/2.65m params)
  FirstStageBoxPredictor (--/36.94k params)
    FirstStageBoxPredictor/BoxEncodingPredictor (--/24.62k params)
      FirstStageBoxPredictor/BoxEncodingPredictor/biases (48, 48/48 params)
      FirstStageBoxPredictor/BoxEncodingPredictor/weights (1x1x512x48, 24.58k/24.58k params)
    FirstStageBoxPredictor/ClassPredictor (--/12.31k params)
      FirstStageBoxPredictor/ClassPredictor/biases (24, 24/24 params)
      FirstStageBoxPredictor/ClassPredictor/weights (1x1x512x24, 12.29k/12.29k params)
  FirstStageFeatureExtractor (--/4.25m params)
    FirstStageFeatureExtractor/InceptionV2 (--/4.25m params)
      FirstStageFeatureExtractor/InceptionV2/Conv2d_1a_7x7 (--/2.71k params)
        FirstStageFeatureExtractor/InceptionV2/Conv2d_1a_7x7/BatchNorm (--/0 params)
        FirstStageFeatureExtractor/InceptionV2/Conv2d_1a_7x7/depthwise_weights (7x7x3x8,
1.18k/1.18k params)
        FirstStageFeatureExtractor/InceptionV2/Conv2d_1a_7x7/pointwise_weights (1x1x24x64,
1.54k/1.54k params)
      FirstStageFeatureExtractor/InceptionV2/Conv2d_2b_1x1 (--/4.10k params)
        FirstStageFeatureExtractor/InceptionV2/Conv2d_2b_1x1/BatchNorm (--/0 params)
        FirstStageFeatureExtractor/InceptionV2/Conv2d_2b_1x1/weights (1x1x64x64, 4.10k/4.10k
params)
      FirstStageFeatureExtractor/InceptionV2/Conv2d_2c_3x3 (--/110.59k params)
        FirstStageFeatureExtractor/InceptionV2/Conv2d_2c_3x3/BatchNorm (--/0 params)
        FirstStageFeatureExtractor/InceptionV2/Conv2d_2c_3x3/weights (3x3x64x192,
110.59k/110.59k params)
      FirstStageFeatureExtractor/InceptionV2/Mixed_3b (--/218.11k params)
        FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_0 (--/12.29k params)
          FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_0/Conv2d_0a_1x1 (--/12.29k
params)
            FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_0/Conv2d_0a_1x1/BatchNorm
(--/0 params)
            FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_0/Conv2d_0a_1x1/weights
(1x1x192x64, 12.29k/12.29k params)
        FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_1 (--/49.15k params)
          FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_1/Conv2d_0a_1x1 (--/12.29k
params)
            FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_1/Conv2d_0a_1x1/BatchNorm
(--/0 params)
            FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_1/Conv2d_0a_1x1/weights
(1x1x192x64, 12.29k/12.29k params)
          FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_1/Conv2d_0b_3x3 (--/36.86k
params)
            FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_1/Conv2d_0b_3x3/BatchNorm
(--/0 params)
            FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_1/Conv2d_0b_3x3/weights
(3x3x64x64, 36.86k/36.86k params)
        FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_2 (--/150.53k params)
          FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_2/Conv2d_0a_1x1 (--/12.29k
params)
            FirstStageFeatureExtractor/InceptionV2/Mixed_3b/Branch_2/Conv2d_0a_1x1/BatchNorm
...
```

```
(--/0 params)
            SecondStageFeatureExtractor/InceptionV2/Mixed_5c/Branch_2/Conv2d_0b_3x3/weights
(3x3x192x224, 387.07k/387.07k params)
          SecondStageFeatureExtractor/InceptionV2/Mixed_5c/Branch_2/Conv2d_0c_3x3 (--/451.58k
params)
            SecondStageFeatureExtractor/InceptionV2/Mixed_5c/Branch_2/Conv2d_0c_3x3/BatchNorm
(--/0 params)
            SecondStageFeatureExtractor/InceptionV2/Mixed_5c/Branch_2/Conv2d_0c_3x3/weights
(3x3x224x224, 451.58k/451.58k params)
        SecondStageFeatureExtractor/InceptionV2/Mixed_5c/Branch_3 (--/131.07k params)
          SecondStageFeatureExtractor/InceptionV2/Mixed_5c/Branch_3/Conv2d_0b_1x1 (--/131.07k
params)
            SecondStageFeatureExtractor/InceptionV2/Mixed_5c/Branch_3/Conv2d_0b_1x1/BatchNorm
(--/0 params)
            SecondStageFeatureExtractor/InceptionV2/Mixed_5c/Branch_3/Conv2d_0b_1x1/weights
(1x1x1024x128, 131.07k/131.07k params)

======================End of Report==========================
264 ops no flops stats due to incomplete shapes.
Parsing Inputs...
Incomplete shape.

=========================Options============================
-max_depth                  10000
-min_bytes                  0
-min_peak_bytes             0
-min_residual_bytes         0
-min_output_bytes           0
-min_micros                 0
-min_accelerator_micros     0
-min_cpu_micros             0
-min_params                 0
-min_float_ops              1
-min_occurrence             0
-step                       -1
-order_by                   float_ops
-account_type_regexes       .*
-start_name_regexes         .*
-trim_name_regexes          .*BatchNorm.*,.*Initializer.*,.*Regularizer.*,.*BiasAdd.*
-show_name_regexes          .*
-hide_name_regexes
-account_displayed_op_only  true
-select                     float_ops
-output                     stdout:

==================Model Analysis Report======================
Incomplete shape.

Doc:
scope: The nodes in the model graph are organized by their names, which is hierarchical like
filesystem.
flops: Number of float operations. Note: Please read the implementation for the math behind
it.

...

  Preprocessor/map/while/ResizeToRange/mul (1/1 flops)

SecondStagePostprocessor/BatchMultiClassNonMaxSuppression/map/while/MultiClassNonMaxSuppressio
n/Greater (1/1 flops)
  BatchMultiClassNonMaxSuppression/map/while/Less (1/1 flops)

SecondStagePostprocessor/BatchMultiClassNonMaxSuppression/map/while/MultiClassNonMaxSuppressio
n/ChangeCoordinateFrame/truediv (1/1 flops)

SecondStagePostprocessor/BatchMultiClassNonMaxSuppression/map/while/MultiClassNonMaxSuppressio
n/ChangeCoordinateFrame/sub_1 (1/1 flops)

SecondStagePostprocessor/BatchMultiClassNonMaxSuppression/map/while/MultiClassNonMaxSuppressio
n/ChangeCoordinateFrame/sub (1/1 flops)
```

```
  SecondStagePostprocessor/BatchMultiClassNonMaxSuppression/map/while/Less_1 (1/1 flops)
  SecondStagePostprocessor/BatchMultiClassNonMaxSuppression/map/while/Less (1/1 flops)
  SecondStageDetectionFeaturesExtract/mul (1/1 flops)
  Preprocessor/map/while/add_1 (1/1 flops)
  Preprocessor/map/while/add (1/1 flops)
  Preprocessor/map/while/ResizeToRange/truediv_1 (1/1 flops)
  Preprocessor/map/while/ResizeToRange/truediv (1/1 flops)
  Preprocessor/map/while/ResizeToRange/mul_3 (1/1 flops)
  Preprocessor/map/while/ResizeToRange/mul_2 (1/1 flops)
  Preprocessor/map/while/ResizeToRange/mul_1 (1/1 flops)

SecondStagePostprocessor/BatchMultiClassNonMaxSuppression/map/while/MultiClassNonMaxSuppressio
n/Minimum (1/1 flops)
  Preprocessor/map/while/ResizeToRange/Minimum (1/1 flops)
  Preprocessor/map/while/ResizeToRange/Maximum (1/1 flops)
  Preprocessor/map/while/ResizeToRange/Greater (1/1 flops)
  Preprocessor/map/while/Less_1 (1/1 flops)
  Preprocessor/map/while/Less (1/1 flops)
  GridAnchorGenerator/zeros/Less (1/1 flops)
  GridAnchorGenerator/mul_8 (1/1 flops)
  GridAnchorGenerator/mul_7 (1/1 flops)
  GridAnchorGenerator/assert_equal/Equal (1/1 flops)
  GridAnchorGenerator/add_4 (1/1 flops)
  GridAnchorGenerator/add_3 (1/1 flops)
  FirstStageFeatureExtractor/GreaterEqual_1 (1/1 flops)
  FirstStageFeatureExtractor/GreaterEqual (1/1 flops)
  Decode/transpose_1/sub (1/1 flops)
  Decode/transpose/sub (1/1 flops)

======================End of Report==========================
2018-12-24 16:21:04.542645: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
```

A10. Object_detection_video.py:

```
######## Video Object Detection Using Tensorflow-trained Classifier
#########
#
# Author: Evan Juras
# Date: 1/16/18
# Description:
# This program uses a TensorFlow-trained classifier to perform object
detection.
# It loads the classifier uses it to perform object detection on a video.
# It draws boxes and scores around the objects of interest in each frame
# of the video.

## Some of the code is copied from Google's example at
##
https://github.com/tensorflow/models/blob/master/research/object_detection/
object_detection_tutorial.ipynb

## and some is copied from Dat Tran's example at
##
https://github.com/datitran/object_detector_app/blob/master/object_detectio
n_app.py

## but I changed it to make it more understandable to me.

# Import packages
import os
import cv2
import numpy as np
import tensorflow as tf
import sys

# This is needed since the notebook is stored in the object_detection
folder.
sys.path.append("..")

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'ModelTrain'
VIDEO_NAME = 'vid1.mp4'

# Grab path to current working directory
CWD_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is
used
# for object detection.
PATH_TO_CKPT =
os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'.','object-detection.pbtxt')

# Path to video
PATH_TO_VIDEO = os.path.join(CWD_PATH,VIDEO_NAME)
```

```
# Number of classes the object detector can identify
NUM_CLASSES = 19

# Load the label map.
# Label maps map indices to category names, so that when our convolution
# network predicts `5`, we know that this corresponds to `king`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

print(label_map)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.Session(graph=detection_graph)

# Define input and output tensors (i.e. data) for the object detection
classifier

# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was
detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes =
detection_graph.get_tensor_by_name('detection_classes:0')

# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Open video file
#video = cv2.VideoCapture(PATH_TO_VIDEO)
#print(PATH_TO_VIDEO)
fi = open("pA.tsv","w")

for x in sorted(os.listdir("sceneA")):
    # Acquire frame and expand frame dimensions to have shape: [1, None,
None, 3]
    # i.e. a single-column array, where each item in the column has the
pixel RGB value
    print(os.path.join("sceneA", x))
    frame = cv2.imread(os.path.join("sceneA", x))
    frame_expanded = np.expand_dims(frame, axis=0)
    #print(frame)
```

```
    # Perform the actual detection by running the model with the image as
input
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes,
num_detections],
        feed_dict={image_tensor: frame_expanded})


    #print(len(scores[0]))
    #print(scores)
    #print(len(classes[0]))
    #print(classes)
    maxv = {}
    for i in range(1, NUM_CLASSES+1):
      maxv[i]=0
    for i in range(0, len(scores[0])):
      maxv[classes[0][i]]=max(maxv[classes[0][i]],scores[0][i])
    print(maxv)
    for i in range(1, NUM_CLASSES+1):
      fi.write(str(maxv[i])+"\t")
    fi.write("\n")
    # Draw the results of the detection (aka 'visulaize the results')
    vis_util.visualize_boxes_and_labels_on_image_array(
        frame,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
        use_normalized_coordinates=True,
        line_thickness=5,
        min_score_thresh=0.4)
    cv2.imwrite(os.path.join("test_timeline", x), frame)

    #print(frame)
    # All the results have been drawn on the frame, so it's time to display
it.
    cv2.imshow('Object detector', frame)
    cv2.imwrite('Salida/' + str(x), frame)

    # Press 'q' to quit
    if cv2.waitKey(1) == ord('q'):
        break
# Clean up
#video.release()
cv2.destroyAllWindows()
```

A11. Código software Arduino:

```
int ite=0;
int Vuelta=0;
int SVerde=2;
int SRojo=3;
int SAma=4;
int Umbral=800;

void setup() {
  // put your setup code here, to run
once:
  Serial.begin(9600);
  pinMode(SVerde, OUTPUT);
  pinMode(SRojo, OUTPUT);
  pinMode(SAma, OUTPUT);
}

void loop() {

  //LeerSenales();
  if(Vuelta==0)
  {
    ParpadeoInicial();
  }

  int Leo = Serial.read();
  if(Leo == 10)
  {
    digitalWrite(SRojo, LOW);
    digitalWrite(SAma, LOW);
    digitalWrite(SVerde, HIGH);
    Serial.println("ACK");
    bool Corto = true;
    while(Corto)
    {
      CuentoPerchas();
      int Recibo = Serial.read();
      if(Recibo == 11)
      {
        Corto = false;
        digitalWrite(SVerde, LOW);
        digitalWrite(SRojo, HIGH);
        digitalWrite(SAma, LOW);
      }
    }
  }
  else
  {
    digitalWrite(SVerde, LOW);
    digitalWrite(SRojo, HIGH);
    digitalWrite(SAma, LOW);
  }

  delay(500);
  Vuelta++;
}

void ParpadeoInicial()
{

  digitalWrite(SVerde, HIGH);
  digitalWrite(SRojo, HIGH);
  digitalWrite(SAma, HIGH);
  delay(500);
  digitalWrite(SVerde, LOW);
  digitalWrite(SRojo, LOW);
  digitalWrite(SAma, LOW);
  delay(500);
  digitalWrite(SVerde, HIGH);
  digitalWrite(SRojo, HIGH);
  digitalWrite(SAma, HIGH);
  delay(500);
  digitalWrite(SVerde, LOW);
  digitalWrite(SRojo, LOW);
  digitalWrite(SAma, LOW);
  delay(500);
}

void CuentoPerchas()
{
  // put your main code here, to run
repeatedly:
  int lectura=analogRead(A0);
  int lectura2=analogRead(A1);
  if(lectura>Umbral &&
lectura2>Umbral)
  {
    digitalWrite(SAma, HIGH);
    digitalWrite(SVerde, LOW);
    ite=ite+1;
    Serial.print("Percha ");
    Serial.print(ite);
    Serial.print(": ");
    Serial.print(lectura);
    Serial.print(", ");
    Serial.println(lectura2);
    lectura=analogRead(A0);
    lectura2=analogRead(A1);
    while(lectura<1020 &&
lectura2<1020)
    {
      delay(100);
      lectura=analogRead(A0);
      lectura2=analogRead(A1);
    }
    delay(7000);
    digitalWrite(SAma, LOW);
    digitalWrite(SVerde, HIGH);
  }
}
```

A12. CamaraUSB.py:

```
import cv2
import time
import os
import shutil

try:
    file = open("Camara/Nombre.txt","r")
    Nombre = file.read()
except:
    file = open("Nombre.txt","r")
    Nombre = file.read()

Nombre = Nombre.strip()
print(Nombre)

folder = "/home/signifyvalladolid/Tensorflow/models/research/object_detection/"
folder_act = "/home/signifyvalladolid/DetectorPerchas/"

for the_file in os.listdir(folder + "sceneA"):
	try:
		file_path = os.path.join(folder + "sceneA", the_file)
		os.unlink(file_path)
	except Exception as e:
		print(e)

for the_file in os.listdir(folder_act + "FotosOriginales"):
	try:
		file_path = os.path.join(folder_act + "FotosOriginales", the_file)
		#os.unlink(file_path)
	except Exception as e:
		print(e)

for the_file in os.listdir(folder_act + "FotosAnalisis"):
	try:
		file_path = os.path.join(folder_act + "FotosAnalisis", the_file)
		#os.unlink(file_path)
	except Exception as e:
		print(e)

for i in range(1):

	try:
		cap = cv2.VideoCapture(1)
		leido, frame = cap.read()
		frame = cv2.resize(frame, (0,0), fx=0.70, fy=0.70)
		if leido == True:
			cv2.imwrite(folder + "sceneA/" + Nombre + ".png", frame)
			cv2.imwrite(folder_act + "FotosOriginales/" + Nombre + ".png", frame)
			#cv2.imwrite(folder + "sceneA/" + Nombre + "_" + str(i) + ".png", frame)

			#cv2.imwrite(folder_act + "FotosOriginales/" + Nombre + "_" + str(i) +
".png", frame)
		else:
			print("Error")

		cap.release()
		time.sleep(2)
	except:
		print("Hola")

os.chdir(folder)
os.system("python3 TFM_analizarImagen.py")
```

A13. TFM_analizarimagen.py:

```
######## Video Object Detection Using Tensorflow-trained Classifier #########
#
# Author: Evan Juras
# Date: 1/16/18
# Description:
# This program uses a TensorFlow-trained classifier to perform object detection.
# It loads the classifier uses it to perform object detection on a video.
# It draws boxes and scores around the objects of interest in each frame
# of the video.

## Some of the code is copied from Google's example at
##
https://github.com/tensorflow/models/blob/master/research/object_detection/object_d
etection_tutorial.ipynb

## and some is copied from Dat Tran's example at
##
https://github.com/datitran/object_detector_app/blob/master/object_detection_app.py

## but I changed it to make it more understandable to me.

# Import packages
import os
import cv2
import numpy as np
import tensorflow as tf
import sys

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'TFM_model'

# Grab path to current working directory
CWD_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'','object-detection.pbtxt')

# Path to execute java
PATH_JAVA = "/home/signifyvalladolid/DetectorPerchas/"

# Number of classes the object detector can identify
NUM_CLASSES = 19

# Load the label map.
# Label maps map indices to category names, so that when our convolution
# network predicts `5`, we know that this corresponds to `king`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
```

```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.Session(graph=detection_graph)

# Define input and output tensors (i.e. data) for the object detection classifier

# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')

# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Open video file
fi = open(PATH_JAVA + "SalidaAnalisis/pA.csv","w")

fi.write("Nombre imagen" + ";")
for i in categories:
    fi.write(i['name'] + ";")

fi.write("\n")

th = [0.65, 0.8, 0.8, 0.2, 0.8, 0.9, 0.8, 0.8, 0.6, 0.97, 0.8, 0.98, 0.98, 0.8,
0.75, 0.8, 0.8, 0.35, 0.05]

for x in sorted(os.listdir("sceneA")):
    # Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]
    # i.e. a single-column array, where each item in the column has the pixel RGB
value
    print(os.path.join("sceneA", x))
    frame = cv2.imread(os.path.join("sceneA", x))
    frame_expanded = np.expand_dims(frame, axis=0)

    # Perform the actual detection by running the model with the image as input
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes, num_detections],
        feed_dict={image_tensor: frame_expanded})

    fi.write(x + ";")
    maxv = {}
    for i in range(1, NUM_CLASSES+1):
      maxv[i]=0
```

```
for i in range(0, len(scores[0])):
   maxv[classes[0][i]]=max(maxv[classes[0][i]],scores[0][i])
print(maxv)
for i in range(1, NUM_CLASSES+1):
   fi.write(str(maxv[i])+";")
fi.write("\n")


maxv = {}
maxbxs = {}
for i in range(1, NUM_CLASSES+1):
   maxv[i]=0
for i in range(0, len(scores[0])):
   if maxv[classes[0][i]]<scores[0][i] :
          maxv[classes[0][i]]=scores[0][i]
          maxbxs[classes[0][i]]=boxes[0][i]

maxc2 = []
maxbxs2 = []
scores2 = []
maxc2.append(5)
scores2.append(0.1)
maxbxs2.append([0,0,1,1])
maxc2.append(5)
scores2.append(0.1)
maxbxs2.append([0,0,1,1])
for i in range(1, NUM_CLASSES+1):
   if maxv[i]>th[i-1] :
          maxc2.append(i)
          scores2.append(maxv[i])
          maxbxs2.append(maxbxs[i])

maxc3 = []
maxbxs3 = []
scores3 = []
maxc3.append(maxc2)
maxbxs3.append(maxbxs2)
scores3.append(scores2)

# Draw the results of the detection (aka 'visulaize the results')
try:
       vis_util.visualize_boxes_and_labels_on_image_array(
          frame,
          np.squeeze(maxbxs3),
          np.squeeze(maxc3).astype(np.int32),
          np.squeeze(scores3),
          category_index,
          use_normalized_coordinates=True,
          line_thickness=8,
          min_score_thresh=0.1)
       cv2.imwrite(os.path.join("test_timelineBB", x), frame)
except:
   print('Error')

# All the results have been drawn on the frame, so it's time to display it.
cv2.imwrite(PATH_JAVA + 'FotosAnalisis/' + str(x), frame)
```

A14. CamaraStreaming.py:

```
import os

folder =
"/home/signifyvalladolid/Tensorflow/models/research/object_detection/"

print("Abriendo streaming")

os.chdir(folder)

os.system("python3 TFM_streaming.py")
```

A15. TFM_streaming.py:

```
######## Picamera Object Detection Using Tensorflow Classifier #########
#
# Author: Evan Juras
# Date: 4/15/18
# Description:
# This program uses a TensorFlow classifier to perform object detection.
# It loads the classifier uses it to perform object detection on a Picamera feed.
# It draws boxes and scores around the objects of interest in each frame from
# the Picamera. It also can be used with a webcam by adding "--usbcam"
# when executing this script from the terminal.

## Some of the code is copied from Google's example at
##
https://github.com/tensorflow/models/blob/master/research/object_detection/object_d
etection_tutorial.ipynb

## and some is copied from Dat Tran's example at
##
https://github.com/datitran/object_detector_app/blob/master/object_detection_app.py

## but I changed it to make it more understandable to me.


# Import packages
import os
import cv2
import numpy as np
import tensorflow as tf
import argparse
import sys

# Set up camera constants
IM_WIDTH = 1280
IM_HEIGHT = 720
#IM_WIDTH = 640    Use smaller resolution for
#IM_HEIGHT = 480   slightly faster framerate

# Select camera type (if user enters --usbcam when calling this script,
# a USB webcam will be used)
camera_type = 'usb'
parser = argparse.ArgumentParser()
parser.add_argument('--usbcam', help='Use a USB webcam instead of picamera',
                    action='store_true')
args = parser.parse_args()
if args.usbcam:
    camera_type = 'usb'

# This is needed since the working directory is the object_detection folder.
sys.path.append('..')

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'TFM_model'

# Grab path to current working directory
CWD_PATH = os.getcwd()
```

```
# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'','object-detection.pbtxt')

# Number of classes the object detector can identify
NUM_CLASSES = 19

## Load the label map.
# Label maps map indices to category names, so that when the convolution
# network predicts `5`, we know that this corresponds to `airplane`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.Session(graph=detection_graph)


# Define input and output tensors (i.e. data) for the object detection classifier

# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')

# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()
font = cv2.FONT_HERSHEY_SIMPLEX

# Initialize camera and perform object detection.
# The camera has to be set up and used differently depending on if it's a
# Picamera or USB webcam.

# I know this is ugly, but I basically copy+pasted the code for the object
# detection loop twice, and made one work for Picamera and the other work
# for USB.
```

```
th = [0.65, 0.8, 0.8, 0.4, 0.8, 0.9, 0.8, 0.8, 0.6, 0.95, 0.8, 0.98, 0.98, 0.8,
0.75, 0.8, 0.8, 0.35, 0.05]

### Picamera ###
if camera_type == 'usb':
    # Initialize USB webcam feed
    camera = cv2.VideoCapture(0)
    ret = camera.set(3,IM_WIDTH)
    ret = camera.set(4,IM_HEIGHT)

    while(True):

        t1 = cv2.getTickCount()

        # Acquire frame and expand frame dimensions to have shape: [1, None, None,
3]
        # i.e. a single-column array, where each item in the column has the pixel
RGB value
        ret, frame = camera.read()
        frame_expanded = np.expand_dims(frame, axis=0)

        # Perform the actual detection by running the model with the image as input
        (boxes, scores, classes, num) = sess.run(
            [detection_boxes, detection_scores, detection_classes, num_detections],
            feed_dict={image_tensor: frame_expanded})

    maxv = {}
    maxbxs = {}
    for i in range(1, NUM_CLASSES+1):
      maxv[i]=0
    for i in range(0, len(scores[0])):
      if maxv[classes[0][i]]<scores[0][i] :
            maxv[classes[0][i]]=scores[0][i]
            maxbxs[classes[0][i]]=boxes[0][i]

    maxc2 = []
    maxbxs2 = []
    scores2 = []
    maxc2.append(5)
    scores2.append(0.1)
    maxbxs2.append([0,0,1,1])
    maxc2.append(5)
    scores2.append(0.1)
    maxbxs2.append([0,0,1,1])
    for i in range(1, NUM_CLASSES+1):
      if maxv[i]>th[i-1] :
            maxc2.append(i)
            scores2.append(maxv[i])
            maxbxs2.append(maxbxs[i])

    maxc3 = []
    maxbxs3 = []
    scores3 = []
    maxc3.append(maxc2)
    maxbxs3.append(maxbxs2)
    scores3.append(scores2)

      # Draw the results of the detection (aka 'visulaize the results')
      vis_util.visualize_boxes_and_labels_on_image_array(
            frame,
            np.squeeze(boxes),
```

```
            np.squeeze(classes).astype(np.int32),
            np.squeeze(scores),
            category_index,
            use_normalized_coordinates=True,
            line_thickness=8,
            min_score_thresh=0.85)

      cv2.putText(frame,"FPS:
{0:.2f}".format(frame_rate_calc),(30,50),font,1,(255,255,0),2,cv2.LINE_AA)

      # All the results have been drawn on the frame, so it's time to display it.
      cv2.imshow('Object detector', frame)

      t2 = cv2.getTickCount()
      time1 = (t2-t1)/freq
      frame_rate_calc = 1/time1

      # Press 'q' to quit
      if cv2.waitKey(1) == ord('q'):
            break

   camera.release()

cv2.destroyAllWindows()
```

A16. Instalación Ubuntu junto con Windows 10:

1. Grabar imagen Ubuntu en un USB
2. Iniciar el ordenador desde el USB
3. Instalar Ubuntu
    a. Seleccionar idioma
    b. Actualizaciones y otro software
    c. Tipo de instalación:
        i. Borrar disco e instalar Ubuntu (Solo sistema Ubuntu).
        ii. Crear partición del disco duro (Junto a Windows 10).
    d. Seleccionar zona horaria
    e. Registro de usuario:
        i. Nombre: SignifyValladolid
        ii. Equipo: SignifyValladolid
        iii. Nombre de usuario: signifyvalladolid
        iv. Contraseña: philipsindal

A17. Instalación de Python 3.6 con Anaconda:

Ir a la página de Anaconda (Anaconda, 21/02/2019) y bajar la última versión disponible (en mi caso 3.7). Los pasos seguidos para la instalación en Ubuntu vienen recogidos en la página web *"How To Install Anaconda on Ubuntu 18.04 [Quickstart]"* (Tagliaferri, 2018, September).

Instalar a través de:

```
bash Anaconda3-5.2.0- … .sh
```

Una vez descargado, actualizar a la última versión compatible con TensorFlow a día 23/01/2019 poniendo el comando:

```
conda install Python=3.6
```

A18. Instalación de TensorFlow y API “Object detection”:

- **TensorFlow:**

Para la instalación se han seguido los pasos indicados en la página oficial (Tensorflow, Install TensorFlow, 21/02/2019).

1. Instalar la última versión de TensorFlow con pip (en mi caso la 1.12 para Python 3.6):

```
pip install tensorflow
```

- **Object detection:**

Para la instalación se han seguido los pasos indicados en el manual oficial de la API (GitHub: Installation Object detection, 21/02/2019).

1. Descargar los archivos oficiales de la página (GitHub: Tensorflow Object Detection API, 21/02/2019).
   Y guardarlos en la carpeta principal dentro de la carpeta TensorFlow.
2. Instalar las siguientes librerías:

```
sudo apt-get install protobuf-compiler python-pil python-lxml python-tk
pip install --user Cython
pip install --user contextlib2
pip install --user jupyter
pip install --user matplotlib
pip install --user pillow
pip install --user lxml
```

3. Instalar COCOAPI y copiar los protocolos:

```
sudo apt install git
git clone https://github.com/cocodataset/cocoapi.git
cd cocoapi/PythonAPI
make
cp -r pycocotools /home/signifyvalladolid/Tensorflow/models/research/
```

4. Configuración de Protobuf:

```
Cd /home/signifyvalladolid/Tensorflow/models/research
protoc object_detection/protos/*.proto --python_out=.
wget -O protobuf.zip
https://github.com/google/protobuf/releases/download/v3.0.0/protoc-3.0.0-linux-
x86_64.zip
unzip protobuf.zip
```

5. Añadir las librerías al path (necesario cada vez que se intente ejecutar):

```
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

6. Testear:

```
python object_detection/builders/model_builder_test.py
```

A19. Instalación de OpenCV en Linux:

Para la instalación de OpenCV es necesario seguir los siguientes pasos:

1. Intentar instalar con anaconda:

```
conda install -c conda-forge opencv
```

2. Para una mejor instalación, hay que seguir los siguientes pasos indicados en la página web de openCV (OpenCV, 21/02/2019). Si falla, probar versión 3.1 (solo usar de los siguientes comandos los 2 primeros y o el 3º o el 4º):

```
conda remove opencv
conda update conda
conda install -c menpo opencv3 "(OpenCV 3.1)"
conda install –channel menpo opencv "(Last OpenCV)"
```

A20. Instalación de Arduino en Linux:

Para la instalación de Arduino es necesario seguir los siguientes pasos:

1. Ir al apartado descargas de la página oficial de Arduino (Download the Arduino IDE, 21/02/2019).
2. Descargar la versión para Linux.
3. Ir a la carpeta donde se ha descargado y extraer los archivos.
4. Abrir un terminal que apunte a esa carpeta y escribir:

```
./install.sh
```

5. Ir al escritorio y comprobar que se ha creado el acceso directo al IDE.

A21. Instalación de Java en Linux:

Para la instalación de java en Linux es necesario seguir los siguientes pasos:

1. Abrir un terminal y escribir:

```
sudo apt install default-jre
```

2. Comprobar que se ha instalado correctamente:

```
java -version
```

3. Instalar el JDK, a través de la página oficial (Oracle, 21/02/2019)

A22. Instalación de Netbeans en Linux:

Para tener instalado el software de desarrollo Netbeans en Linux es necesario seguir los siguientes pasos:

1. Abrir un terminal y descargar la versión de netbeans:

```
wget -c http://download.netbeans.org/netbeans/8.2/final/bundles/netbeans-8.2-linux.sh
```

2. Dar permisos de ejecución al archivo descargado:

```
chmod +x netbeans-8.2-linux.sh
```

3. Ejecutar el instalador:

```
./netbeans-8.2-linux.sh
```

4. Seguir los pasos indicados en el instalador.

A23. Crear ejecutable en el escritorio de Linux:

Para crear un ejecutable en Linux del programa, en primer lugar, es necesario crear un archivo txt con extensión .sh. Para ello, se abre un terminal y se escribe lo siguiente:

```
cd /home/signifyvalladolid
nano script.sh
```

Dentro del script habrá que escribir las siguientes líneas:

```
#!/bin/bash
export
PYTHONPATH=$PYTHONPATH:/home/signifyvalladolid/anaconda3/lib/pyth
on36.zip
export
PYTHONPATH=$PYTHONPATH:/home/signifyvalladolid/anaconda3/lib/pyth
on3.6
export
PYTHONPATH=$PYTHONPATH:/home/signifyvalladolid/anaconda3/lib/pyth
on3.6/lib-dynload
export
PYTHONPATH=$PYTHONPATH:/home/signifyvalladolid/.local/lib/python3
.6/site-packages
export
PYTHONPATH=$PYTHONPATH:/home/signifyvalladolid/anaconda3/lib/pyth
on3.6/site-packages
cd /home/signifyvalladolid/DetectorPerchas/
java -jar TFM_V1.jar
```

Una vez generado el archivo, es necesario otorgarle los permisos de ejecución a través del siguiente comando:

```
sudo chmod +x script.sh
```

Realizados todos los pasos, el último punto es instalar la aplicación que nos ayudará a crear lanzadores en Linux, a través de los siguientes comandos:

```
sudo apt install gnome-panel
gnome-desktop-item-edit ~/Escritorio --create-new
```

Anexo: Artículo de investigación en español

Anexo: Artículo de investigación en inglés

Desarrollo e implementación IoT de un sistema de reconocimiento de imágenes a nivel industrial

David Sanz Cabreros

Universidad Internacional de la Rioja, Logroño (España)

Fecha: 17/02/2019

Resumen

El objetivo de este proyecto es diseñar un reconocedor de objetos basado en la API "Object Detection" de TensorFlow con la finalidad de aplicarlo a nivel industrial en una planta de fabricación de luminarias para mejorar uno de sus procesos de fabricación. La mejora que se propone con respecto a los modelos actuales de detección de objetos del mercado a nivel industrial es la generación de dicho reconocedor con técnicas de aumentación de datos, donde a partir de unas pocas imágenes de entrada de las diferentes piezas a detectar y de un video del fondo donde estarán localizadas. Esto permitirá crear un modelo suficientemente potente como para detectar los objetos deseados, suprimiendo la parte de generación de cantidades enormes de datos de entrada y facilitando así su desarrollo y actualización. Tras su implementación física, la transmisión de información para su visualización se realizará mediante dispositivos IoT.

Palabras Clave

Object Detection

TensorFlow

Data Augmentation

Visión artificial industrial

Dispositivos IoT

I. Introducción

Dentro de la visión artificial, el reconocimiento de objetos en imágenes y su identificación es una de las áreas más atractivas y que mayor desarrollo han tenido en los últimos años.

En el sector industrial son muchas las compañías que optan por este tipo de sistemas para mejorar sus procesos productivos, la calidad y aportar valor añadido a sus productos.

Es en esa área de la industria en el cual se centra este trabajo final de máster, que tiene como finalidad desarrollar un sistema de visión artificial, basado en el reconocimiento de imágenes, con el objetivo de ser implementado en el proceso productivo de pintura de la fábrica Signify Manufacturing Spain S.L., localizada en Valladolid, con la finalidad de mejorar dicho proceso, generando nueva información para la planta sobre el material producido, y aportando datos necesarios para el cálculo de aprovechamiento, eficiencia y calidad del proceso en el que será implementado.

La parte innovadora que pretende este proyecto es la de diseñar un sistema de reconocimiento de imágenes implementado directamente en un entorno industrial, sin grandes gastos económicos de herramientas comerciales, y de una forma ágil y asequible siendo capaz de diseñar un sistema suficientemente potente de reconocimiento de imágenes a partir de una pequeña cantidad de imágenes de entrada. Además, propone la creación de un entorno inteligente dentro de la factoría a través de dispositivos IoT obteniendo la información generada por el clasificador y visualizándolo en la parte deseada de la instalación vía internet sin necesidad de cableados.

II. Estado del Arte

Dentro de la inteligencia artificial, existen diferentes áreas o casos de uso como son el aprendizaje automático, el procesamiento del lenguaje natural, la robótica o la visión artificial entre otros.

A. Visión Artificial:

De entre todas estas disciplinas situadas dentro de la inteligencia artificial [11], la aplicación objetivo de este trabajo se encuentra enmarcada dentro del área de la visión artificial [7]. Por eso mismo, será la técnica que será explicada más en detalle en este apartado. Entre las herramientas empleadas en este proyecto, se encuentran:

- **Object Detection API:** Para el desarrollo de aplicaciones típicas como es el caso del reconocimiento de objetos en imágenes, reconocimiento de voz… TensorFlow [1] aporta una serie de API's [13] que permiten usar modelos ya existentes.
  "Object Detection" [18] es una de estas API's que como menciona en la página oficial [14] permite "localizar e identificar múltiples objetos en una sola imagen".
- **OpenCV:** esta biblioteca desarrollada por Intel es sin ninguna duda una de las más importantes a nivel mundial en visión [3], incluye multitud de funciones de visión artificial, tanto para el reconocimiento de imágenes, como para su tratamiento.
- **Aumentación de datos:** una de las técnicas más empleadas en los últimos años en el mundo del Deep Learning, en especial para las aplicaciones de visión artificial es la aumentación de datos.
  Para solucionarlo el problema de la cantidad de datos necesarias, surgió la técnica denominada en ingles Data Augmentation la cual consiste en "agregar valor a los datos básicos al agregar información derivada de fuentes internas y externa" [39]. Agregando pequeñas modificaciones a las imágenes como traslaciones, rotaciones, "flips", escalados…, se pueden generar muchas imágenes. En artículos como el presentado por Bharath Raj [31] o en libros como el de McLaughlin, N, [27], el de Fawzi A. [9] o en el de Zhu, X [40] explican el funcionamiento de esta técnica.
  El objetivo es aportar a la red neuronal, diferentes imágenes de un mismo objeto con pequeñas modificaciones, que obliguen a la red a ser capaz de generalizar conceptos en vez de memorizarlos:

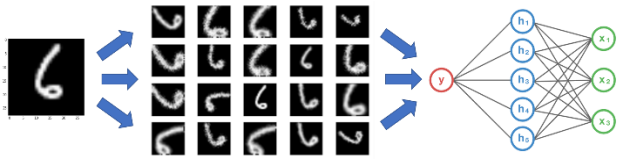

Fig. 1: Red neuronal con Data Augmentation

- **Transfer Learning:** El Transfer Learning o aprendizaje por transferencia, es un método de aprendizaje automático en el cual se emplean modelos pre-entrenados para una tarea, reutilizándolo para nuevas tareas, con la finalidad de reducir el tiempo de cómputo requerido para desarrollar modelos de redes neuronales [4] [5] [15].

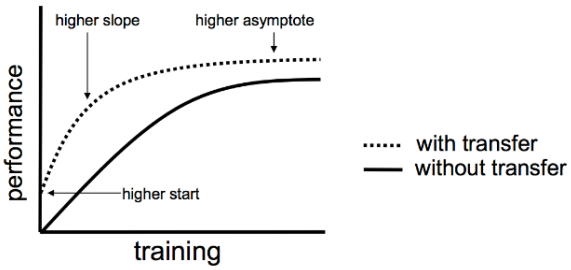

Fig. 2: Comparación de entrenamiento con y sin Transfer Learning

- **Dispositivos IoT**: Los dispositivos IoT (internet de las cosas) son dispositivos que se basan el concepto IoT, el cual se refiere a una interconexión digital de objetos cotidianos con internet. El uso de este tipo de dispositivos y su conexión a través de internet es una de las tecnologías más en auge en la actualidad. Como se define en el libro "Internet of Things (IoT): A vision, architectural elements, and future directions." [16]. Crear ambientes inteligentes donde los dispositivos se comunican entre sí es uno de los avances más potentes de los últimos años y uno de los retos más importantes que tendrán que afrontar las empresas en los próximos años para mantenerse a la vanguardia del sector industrial [25]. Son muchos los proyectos basados en estos dispositivos desarrollados en los últimos años [17] [22] [35].

B. Detección de objetos con Deep Learning:

Las redes neuronales [36] son un tipo de modelo computacional muy empleado actualmente, sobre todo en visión artificial, donde con el auge de las redes neuronales profundas, conocidas como Deep Learning, han mostrado muy buenos resultados en las tareas de clasificación de objetos en imágenes.

Entre los diferentes tipos de redes existentes, las más usuales son las mostradas a continuación:

- Redes convolucionales (CNN): es una de las tecnologías que revolucionó el mundo del Deep Learning, y más concretamente de la visión artificial y el área del reconocimiento de objetos [6] [10] [24]. La característica principal de estas redes es su diferente estructura de capas en comparación con las redes feedforward tradicionales [20] [21] [23] [39], entre las que se encuentran:
  - Capas convolucionales.
  - Capas Max Pooling.
  - Capas Fully Connected.

- R-CNN y Faster R-CNN: Como mejoras a los modelos de redes convolucionales, surgieron modelos como las "Faster R-CNN" cuyo objetivo era reducir el tiempo de ejecución de las redes de neuronas empleadas para detección de objetos [12] [33].

- "You Only Look Once" (YOLO): esta es una de las últimas contribuciones aportadas en la que "presentan un nuevo enfoque para la detección de objetos. A diferencia de los modelos anteriores, su modelo encuadra la detección de objetos como un problema de regresión a cuadros delimitadores separados espacialmente y probabilidades de clase asociadas" [32].
- "Single Shot MultiBox Detector" (SSD): Como mejora a la velocidad de las Faster R-CNN presentadas anteriormente, en 2016 aparecieron las SSD (Single Sol Multibox Detector), cuyo objetivo es realizar un reconocimiento de imágenes lo más cercano posible al tiempo real [26].

Comparando estas 4 tipo de redes como se hace en la Figura 3, se puede observar las mejoras que presentan unas frente a otras.

Fig. 3: YOLO vs SSD vs Faster R-CNN: Precisión vs velocidad

- Inception network: una de las tecnologías que se empleará en el desarrollo del trabajo, es la diseñada para la red Inception (GoogleNet) [30] [37].
  La primera versión de esta tecnología pretendía resolver problemas como la variación en la ubicación de la información entre imágenes de objetos similares, los problemas de *overfitting* presentados por las redes muy profundas, las cuales son capaces de memorizar objetos, o el alto costo computacional de las redes neuronales muy profundas. Para ello, propuso emplear filtros con múltiples tamaños operando en el mismo nivel, con la finalidad de obtener "más ancho" en lugar de "más profundo".
  Las diferentes versiones posteriores a esta (v2 y v3) siguieron mejorando problemas como reducir los cuellos de botella producidos al reducir las dimensiones de la entrada, o comprobar si los clasificadores auxiliares contribuían lo suficiente al proceso final como se pensaba, o si actuaban como regularizadores.
  La versión 4 de Inception, así como la versión Inception-ResNet, tenía como objetivo hacer los módulos más uniformes, para lo que modificaron el "vástago" (*stem*), introduciendo además los "Bloques de Reducción" (*Reduction Blocks*) empleados para cambiar el ancho y la altura de las cuadriculas.
  Para entender un poco mejor las ventajas de los diferentes extractores de características presentados, se empleará una imagen del articulo realizado por Jonathan Hui [19].

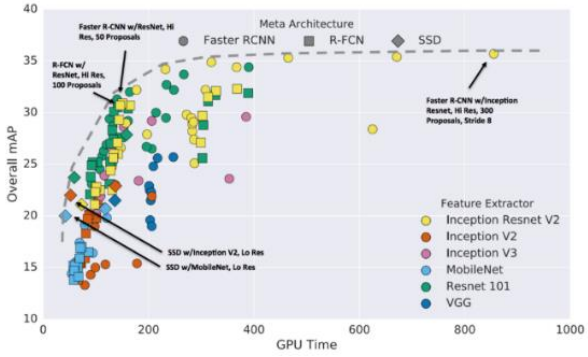

Fig. 4: Inception: Velocidad vs Precisión

III. Contribución

El objetivo de este trabajo final de máster consiste en desarrollar una metodología de reconocimiento de placas que será integrada en una prueba piloto de un sistema de detección y clasificación en tiempo real a través de visión artificial, integrando el desarrollo software del algoritmo de detección de objetos en dispositivos IoT con sensores visuales, abaratando el precio de los sistemas actuales disponibles en el mercado. El trabajo se podría dividir en 5 apartados:

1. Desplegar las herramientas existentes de Object Recognition y adaptarla al reconocimiento de placas de luminarias.
2. Usar técnicas de Deep Learning con aumentación de datos y transferencia de aprendizaje.
3. Incluir dispositivos IoT que acceda a la información para crear un entorno visual inteligente dentro de la factoría.
4. Diseñar una solución empleando sistemas de bajo coste para disminuir gastos.
5. Evaluar el sistema de reconocimiento de placas a través de las imágenes recolectadas en el entorno real de la industria.

Los pasos e ideas desarrolladas durante el desarrollo del TFM se basan en las ideas propuestas en el trabajo de investigación "Straightforward Recognition of Daily Objects in Smart Environments from Wearable Vision Sensor" [28] desarrollado por Medina Quero, Javier (tutor del TFM), en el cual "muestran un método novedoso que facilita el reconocimiento visual de objetos cotidianos en entornos inteligentes a partir de sensores de visión wearables". Dicha metodología se ha integrado en el reconocimiento de este proyecto, que abarca un ámbito de aplicación y evaluación mayor con dispositivos IoT creando en la factoría un espacio visual inteligente para el reconocimiento de piezas industriales. A continuación, se muestra una imagen de dicho trabajo en el que se puede observar las ideas claves mostradas:

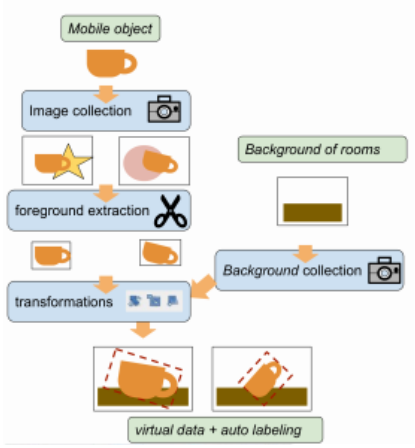

Fig. 5: Generación de imágenes de entrada a partir de imágenes individuales

A. Identificación del entorno y datos a clasificar:

En primer lugar, es necesario analizar el entorno del sistema donde se pretende implementar la aplicación, así como entender las clases de luminarias a clasificar por el modelo.

En cuanto a la ubicación, el sistema estará ubicado en una zona de la cadena de pintura de la factoría en la cual no se pueden acumular las piezas que circulan por dicha cadena una detrás de otra. En dicha zona se instaló una tela blanca que servirá de fondo para mejorar la precisión del modelo de clasificación.

En la parte de análisis de datos, tras identificar todos los posibles elementos que pueden circular por la instalación (154 posibles combinaciones), se obtuvo un total de 19 clases principales en función de la producción de la planta, mostradas en la Tabla 1.

B. Desarrollo del sistema de detección:

Con los datos a clasificar identificados, el siguiente paso es el de diseñar el sistema de detección tomando como idea base el documento mencionado en el estado del arte [28].

1. Captación de imágenes a clasificar:

Para cada una de las clases identificadas se realizan entre 20-25 capturas frontales y desde diferentes ángulos similares a las de la Figura 6:

Fig. 6: Captación de imágenes a clasificar

2. Eliminación del fondo de las imágenes:

Con todas las imágenes captadas, almacenadas y nombradas correctamente, el siguiente objetivo es el de eliminar el fondo de estas, dejando imágenes de cada una de las piezas con su forma física, sobre un fondo trasparente o de un color de pixeles fijado correctamente.

Para ello, se diseñó un código en Python encargado de pasar de una en una todas las imágenes deseadas llamando a la herramienta Grabcut [34]. En la Figura 7 se muestra el proceso seguido con una imagen para eliminar el fondo de la pieza deseada:

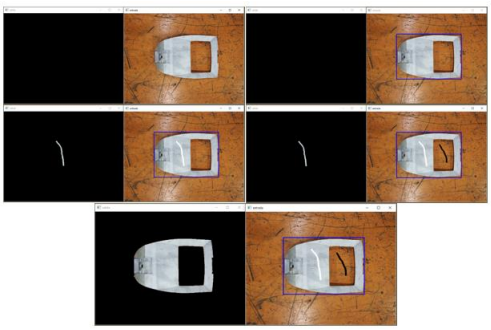

Fig. 7: Eliminación del fondo de una imagen

3. Captación del entorno:

Capturadas y segmentadas todas las imágenes de las diferentes clases, el siguiente paso consiste en obtener imágenes del fondo. Para esto, se realizó un video de 20 segundos tomado con una cámara móvil normal, de la que se obtienen 15 frames de la zona seleccionada como se muestra en la Figura 8.

Fig. 8: Frames del entorno obtenidos a partir del video grabado

4. Generación de imágenes de entrenamiento y test:

Para la generación de las imágenes se diseñó un código en Python encargado de superponer cada una de las imágenes de las piezas sobre todas y cada una de las imágenes de fondo eliminando los píxeles negros generados en el apartado 2 al segmentar las imágenes.

En esta parte de aumentación de datos, es importante generar la mayor cantidad de imágenes diferentes posibles para dotar al sistema de generalización y reducir así el posible overfitting en las imágenes de entrenamiento durante dicho entrenamiento. Para ello, se aplican las siguientes transformaciones:

- Traslación: movimientos de la imagen de la pieza ±45 pixeles con respecto del centro de la imagen del fondo.
- Rotación: giros de cada imagen de las piezas ±45º con respecto al centro de la imagen de la pieza, a mayores de la rotación de 45º, 90º y 180º aplicada sobre todas las imágenes.
- "Flip": efecto espejo sobre la imagen de la pieza.
- Escalado: aumentación y disminución en un rango de ±50% con el fin de hacer más o menos grandes las imágenes

Cada imagen generada es clasificada por el sistema entre imagen de entrenamiento o de test en una proporción de 2 imágenes de test por cada 8 de entrenamiento.

En la Figura 9 se muestran algunas imágenes generadas a la salida del sistema.

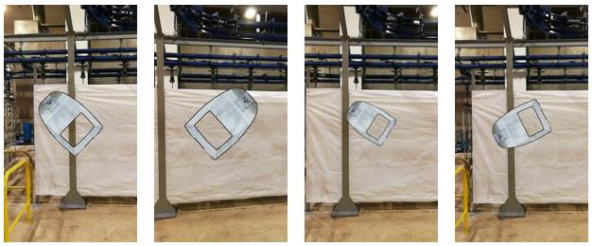

Fig. 9: Imágenes de salida del programa de aumentación

5. Compresión de archivos TFRecord para el entrenamiento:

Una vez se dispone de todas las imágenes superpuestas y clasificadas en entrenamiento y test, el siguiente paso consiste en generar a partir de los archivos .csv los archivos correspondientes en formato ".record" que se emplearán en la API "Object Detection" de TensorFlow como entradas y salidas.

***6.* Selección, ajuste y despliegue del modelo de entrenamiento:**

Con todas las imágenes de entrada disponibles, la API Object Detection de TensorFlow instalada, y los archivos .record creados, el siguiente paso es definir los parámetros necesarios del modelo para realizar el entrenamiento.

Para ello, es necesario tener los siguientes archivos:

- Object-detection.pbtxt: Este archivo, es el encargado de definir la relación entre las diferentes clases y el ID correspondiente de cada una de ellas.
- Obj_inception.config: Este archivo, tiene como finalidad definir todos los parámetros del modelo con el cual se va a realizar el entrenamiento de la red, entre los que se incluyen:
    - Numero de clase: 19.
    - Modelo predefinido empleado en el "Transfer Learning": faster_rcnn_inception_v2
    - Ubicación del archivo.record de entrenamiento y de test.

Con todos estos pasos realizados para preparar el entrenamiento, ya se está en disposición de entrenar las redes neuronales deseadas y evaluar los resultados. Estos pasos se explicarán en el apartado IV.

IV. Resultados o Evaluación

Con la metodología definida y comprobado su funcionamiento, en este apartado se explican los dos ensayos o evaluaciones realizadas para comprobar el funcionamiento de las redes neuronales entrenadas con el sistema de generación de datos aumentados planteado durante el proyecto.

Para ello, se realizan dos evaluaciones, una con menos clases para obtener de forma más rápida un modelo con el que probar, y en caso de obtener buenos resultados, expandirlo a las 19 clases indicadas en la parte de análisis.

A. Primer caso de estudio de la red neuronal (6 clases):

En primer lugar, se realizó el entrenamiento de una red neuronal sencilla, con tan solo 6 clases físicamente diferenciables entre sí con el fin de detectar los posibles problemas a tener en cuenta más adelante, así como para validar la metodología empleada con aumentación de datos, y el software instalado para realizar el Transfer Learning.

De todas las clases presentes en la Tabla 1, se seleccionaron las clases 1, 2, 8, 10, 13 y 19.

Los datos de dicho entrenamiento desde la generación de datos aumentados hasta la obtención del modelo final fueron los siguientes:

- 118 imágenes originales de las piezas sin fondo.
- 19 frames tomados de un video de 25 segundos.
- 21922 imágenes de entrenamiento y 385 de validación generadas en 1 hora y 56 minutos.
- Entrenamiento: 23675 iteraciones y un total de 24 horas de entrenamiento en un ordenador con procesador Intel Core i5 con CPU 2.50, S.O. Ubuntu (Linux) y memoria RAM de 8GB.

Los resultados obtenidos con las imágenes de test para cada una de las clases y la matriz de confusión correspondiente fueron las mostradas en la Figura 9 y 10 correspondientemente.

ID	CLASE REAL	IMAGENES A CLASIFICAR	VERDADEROS POSITIVOS		FALSOS NEGATIVOS	
1	AVOVGEN2_FRAME	36	36	100%	0	0%
2	CITYSOULGEN2_CANOPY	69	69	100%	0	0%
3	CORELINELARGE_COVER	78	72	92.3%	6	7.7%
4	DIGISTREETCATENARY_HOUSING	68	59	86.76%	9	13.24%
5	LUMA_FRAME	73	45	61.64%	28	38.36%
6	UNISTREET_HOUSING	61	61	100%	0	0%

Fig. 10: Datos de validación: primer caso de estudio

ID CLASES		1	2	3	4	5	6
1	AVOVGEN2_FRAME	36					
2	CITYSOULGEN2_CANOPY		69				
3	CORELINELARGE_COVER	3		72			3
4	DIGISTREETCATENARY_HOUSING				59		9
5	LUMA_FRAME	6	1			45	21
6	UNISTREET_HOUSING						61

Fig. 11: Matriz de confusión: primer caso de estudio

A partir de este estudio, aparte de validar que el sistema de generación de datos aumentados para el entrenamiento era válido, se obtuvieron una serie de conclusiones a tener en cuenta:

- Revisar la cara vista de cada luminaria en la zona deseada de acuerdo con la hoja de proceso de la zona de pintura.
- Analizar una a una las diferentes clasificaciones, y fijar sus valores del filtro de confidence score para añadirlos en el programa final.

B. Segundo y último caso de estudio (19 clases):

Para el segundo caso de estudio, se emplearon todas las clases indicadas en la Tabla 1, modificando previamente aquellas imágenes originales que mostraban partes de las luminarias que no se verían en la realidad, reduciendo así el posible ruido generado en la red neuronal.

Los datos de este caso de estudio son los siguientes:

- 199 imágenes originales de las piezas sin fondo de las 19 clases.
- 15 frames tomados de un video de 20 segundos.
- 19181 imágenes de entrenamiento y 496 de validación generadas en 3.15 horas.
- Entrenamiento: 123233 iteraciones y un total de 4 días y 20 horas de entrenamiento con el mismo hardware.

Los resultados obtenidos en este caso de estudio son los mostrados en la figura 12, en las que se muestran los datos reales:

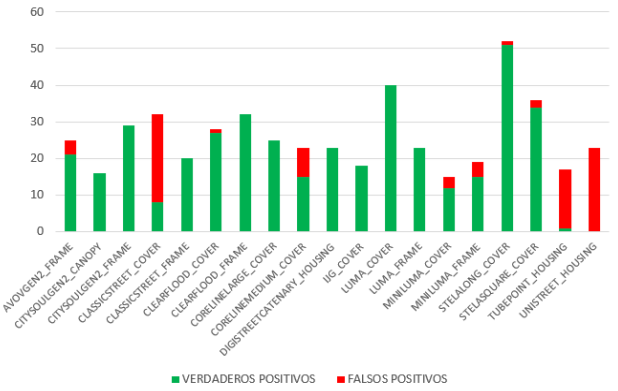

Fig. 12: Datos de validación: segundo caso de estudio

A partir de estos datos y de la matriz de confusión, se obtuvo un porcentaje promedio por clase de las clasificaciones realmente positivas (TP) y los falsos negativos obtenidos en la clasificación errónea (FN), que son los mostrados en la Figura 13.

ID	CLASE REAL	PORCENTAJE PROMEDIO POSITIVOS REALES	CANTIDAD POSITIVOS	PORCENTAJE PROMEDIO FALSO NEGATIVO	CANTIDAD FALSO NEGATIVO
1	AVOVGEN2_FRAME	0,9227	21	0,5553	4
2	CITYSOULGEN2_CANOPY	0,9869	16	-	0
3	CITYSOULGEN2_FRAME	0,9815	29	-	0
4	CLASSICSTREET_COVER	0,7856	8	0,2579	24
5	CLASSICSTREET_FRAME	0,9774	20	-	0
6	CLEARFLOOD_COVER	0,9195	27	0,4780	1
7	CLEARFLOOD_FRAME	0,8719	32	-	0
8	CORELINELARGE_COVER	0,9892	25	-	0
9	CORELINEMEDIUM_COVER	0,7793	15	0,2899	8
10	DIGISTREETCATENARY_HOUSING	0,9964	23	-	0
11	IJG_COVER	0,9722	18	-	0
12	LUMA_COVER	0,9978	40	-	0
13	LUMA_FRAME	0,9837	23	-	0
14	MINILUMA_COVER	0,9739	12	0,6557	3
15	MINILUMA_FRAME	0,8377	15	0,5707	4
16	STELALONG_COVER	0,9634	51	0,7462	1
17	STELASQUARE_COVER	0,9547	34	0,6385	2
18	TUBEPOINT_HOUSING	0,4413	1	0,2286	16
19	UNISTREET_HOUSING	-	0	0,1995	23

Fig. 13: Valores promedio TP y FN en validación: segundo caso de estudio

Con estos datos se obtuvieron las siguientes conclusiones:

- Las clases **que no se han clasificado mal nunca** presentan valores **muy elevados** de promedio en clasificación correcta.
- Hay clases que cuando se han clasificado incorrectamente, presentan valores promedio muy elevados pero **han fallado** pocas veces.
- La clase 4 y la 9 presentan un caso especial, puesto que **cuando falla no presenta valores elevados** pero sí presenta **valores elevados al acertar**.
- El último caso es el de las clases 18 y 19, las cuales presentan **valores muy bajos al fallar**.
- Hay que tener en cuenta que la clasificación de **19 categorías tan similares** es compleja y **supone un desafío** para las técnicas existentes.

C. Ajuste de los filtros con datos de validación y test:

Con los datos obtenidos en el apartado anterior, se realizó un ajuste de los filtros antes de poner el modelo en producción.

Con las imágenes de **validación** se realizará el ajuste de los filtros para cada una de las clases, obteniendo la matriz de confusión de la clasificación y ajustando dichos filtros en función de la métrica más conveniente.

- Filtros a 0.5:

$$precisión = \frac{204}{204+39} * 100 = 83{,}95\%$$

$$recall = \frac{204}{204+44} * 100 = 82{,}26\%$$

- Filtros a 0.1:

$$precisión\ 0{,}1 = \frac{207}{207+41} * 100 = 83{,}47\%$$

$$recall\ 0{,}1 = \frac{207}{207+41} * 100 = 83{,}47\%$$

- Filtros a 0.9:

$$precisión\ 0{,}9 = \frac{166}{166+15} * 100 = 91{,}71\%$$

$$recall\ 0{,}9 = \frac{166}{166+82} * 100 = 66{,}94\%$$

Con las pruebas anteriores se observa:

- **Al disminuir el valor del filtro del confidence score**, la precisión individual no mejora en ningún caso, puesto que para clasificar un valor como bueno hace falta que el sistema obtenga precisiones menores. En este caso, **no aparecen imágenes que no ha sido capaz de clasificar**, y por lo tanto, aparecen más verdaderos positivos. Por otro lado, el Recall global mejora porque al disminuir el umbral, casi todo lo que aparece es capaz de clasificarlo.
- **Al aumentar el valor del filtro sobre el confidence score** se observa una restricción de la clasificación. Esto es debido a que al ser más restrictivo, las clases que suelen predominar por haber aprendido mejor sus patrones mejoran su precisión, sin embargo, el resto de las clases que necesitan valores de filtro inferiores no son capaces de clasificar correctamente y disminuye su Recall. A diferencia de antes, ahora aparecen **67 imágenes sin identificación**.

Con el fin de mejorar el sistema, se fijan filtros individuales por clase, como se muestra en la Tabla 2 y en la figura 14, obteniendo los siguientes resultados contra los datos de validación:

$$precisión = \frac{210}{210+11} * 100 = 95{,}02\%$$

$$recall = \frac{211}{211+37} * 100 = 85{,}08\%$$

	ID CLASES	Filtros	Verdaderos Positivos (TP)	Falsos Positivos (FP)	Falsos Negativos (FN)	Precisión	Recall
1	AVOVGEN2 FRAME	0,65	12	0	0	100,00%	100,00%
2	CITYSOULGEN2 CANOPY	0,8	8	0	0	100,00%	100,00%
3	CITYSOULGEN2 FRAME	0,8	15	0	0	100,00%	100,00%
4	CLASSICSTREET COVER	0,2	8	0	8	100,00%	50,00%
5	CLASSICSTREET FRAME	0,8	10	0	0	100,00%	100,00%
6	CLEARFLOOD COVER	0,8	12	2	2	85,71%	85,71%
7	CLEARFLOOD FRAME	0,8	12	0	4	100,00%	75,00%
8	CORELINELARGE COVER	0,8	12	0	0	100,00%	100,00%
9	CORELINEMEDIUM COVER	0,6	7	0	5	100,00%	58,33%
10	DIGISTREETCATENARY HOUSING	0,95	11	2	0	84,62%	100,00%
11	IJG COVER	0,8	9	1	0	90,00%	100,00%
12	LUMA COVER	0,98	20	3	0	86,96%	100,00%
13	LUMA FRAME	0,98	10	0	2	100,00%	83,33%
14	MINILUMA COVER	0,8	6	0	1	100,00%	85,71%
15	MINILUMA FRAME	0,72	7	0	3	100,00%	70,00%
16	STELALONG COVER	0,8	25	0	1	100,00%	96,15%
17	STELASQUARE COVER	0,8	17	0	1	100,00%	94,44%
18	TUBEPOINT HOUSING	0,35	5	0	3	100,00%	62,50%
19	UNISTREET HOUSING	0,05	5	3	7	62,50%	41,67%

Fig. 14: Resultados de validación con filtros específicos

Con el fin de asegurar que estos filtros son válidos para imágenes no empleadas hasta el momento, se emplean las **imágenes de test.** Tras realizar la ejecución, se observa que se han detectado 221 de las 248 imágenes, un 89,11% de las imágenes, de las cuales se han detectado correctamente 215, siendo 27 las imágenes sin identificación cuyos porcentajes no han superado ningún umbral.

$$precisión = \frac{215}{215+6} * 100 = \ 97{,}29\%$$

$$recall = \frac{215}{215+33} * 100 = \ 86{,}69\%$$

D. Implementación en producción con dispositivos IoT:

Finalizado el proceso de modelado de la red neuronal, el último paso realizado fue su implementación en el entorno de producción real para verificar su comportamiento. Para ello, se diseñó un entorno inteligente interconectado como el mostrado en la figura 15.

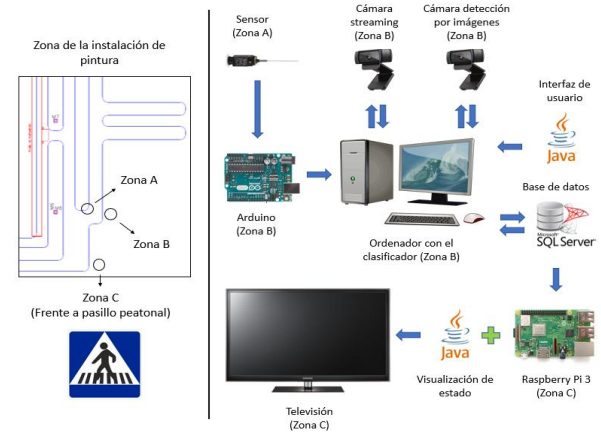

Fig. 15: Esquema de la instalación en producción

La instalación consta de 3 zonas:

- **Zona A**: ubicación del sensor de perchas, situado en la propia cadena de pintura.
- **Zona B:** ubicación del software y hardware encargado de recibir las señales del sensor, realizar el análisis de las imágenes a través del interfaz de usuario y almacenar los datos en la base de datos de la fábrica.
- **Zona C:** ubicación del monitor de visualización público con el proceso de detección (esta ubicación puede variar en función de los requerimientos de la factoría).

Con la instalación terminada, se analizó el comportamiento durante los primeros días, observando resultados muy cercanos a los obtenidos durante el modelado con los datos de test, con clases obteniendo valores de clasificación entre 85-99%. En la siguiente figura se muestran algunas imágenes captadas por el sistema en tiempo real:

Fig. 16: Imágenes clasificadas en tiempo real en el entorno de producción

V. Discusión o Análisis de Resultados

Con los casos de estudio realizados durante el trabajo y analizado el funcionamiento del sistema en el entorno de producción real, se sacan las siguientes conclusiones:

- La metodología mostrada para la generación de datos aumentados necesarios para el entrenamiento no solo muestra claras ventajas en cuanto al esfuerzo de recolección y etiquetado de los objetos, sino que en el entorno de producción real presenta resultados muy válidos, siendo capaz de clasificar en un porcentaje similar al obtenido con imágenes generadas artificialmente.
- La inclusión de filtros de score individuales para cada clase permite obtener mejores resultados que los obtenidos sin filtros o con filtrados genéricos, individualizando para cada clase en función de la capacidad clasificatoria de la red neuronal.
- El aumento o disminución de los filtros individuales permite ajustar el clasificador de acuerdo con las necesidades de la factoría, en función de si se desea más precisión o más Recall. Dicho en otras palabras, si se desea que el sistema sea capaz de clasificar cuando está más seguro de que la pieza es correcta a riesgo de dejar imágenes sin clasificar, o clasificando todas las imágenes de entrada del sistema a riesgo de reducir la precisión de este.

VI. Conclusiones

Analizando las contribuciones del proyecto, se pueden observar desde el punto de vista de las contribuciones aportadas a la empresa, y las aportadas por la metodología de generación de datos aumentados para facilitar la creación de datos de entrada suficientes para el entrenamiento de redes neuronales.

El objetivo de la compañía era disponer de un sistema de visión artificial capaz de detectar las piezas de la instalación sin necesidad de un gran desembolso económico, y sin necesitar recolectar gran cantidad de imágenes. El sistema IoT diseñado en el proyecto empleando elementos de bajo coste, así como la metodología diseñada para la generación de las entradas basada en aumentación de datos cumple con los requerimientos de la empresa.

Desde el punto de vista general sobre sistemas de aumentación de datos, este proyecto presenta una metodología de obtención de estos suficientemente potente como para ser empleada en entornos industriales, siendo un modelo tan complicado de realizar para la gran cantidad de clases presentes con formas similares, sin el uso de gran cantidad de imágenes reales, sino que solo ha sido requisito inicial disponer de unas 20-25 muestras de cada tipo de luminaria.

Por otro lado, es importante destacar la alta labor de ingeniería desplegada ante la necesidad que ha surgido durante el desarrollo del trabajo de unificar diferentes técnicas:

- Para el análisis previo de los datos obteniéndolos directamente de la base de datos de la fábrica con programación tradicional basados en lenguaje Java y SQL, así como su posterior análisis a través de herramientas estadísticas como Minitab.
- Para la generación de los datos aumentados con herramientas basadas en Python y diferentes librerías como OpenCV.
- Para su implementación final empleando sistemas operativos diferentes como Windows 10 y Linux, dispositivos IoT como Arduino y Raspberry Pi, y software específico basado en lenguaje Java para permitir al usuario final usar el sistema de una forma fácil.

En cuanto a las líneas futuras que aparecen a posteriori de este proyecto, se encuentran las siguientes:

- Mejora del sistema de captación: una de las primeras etapas en la mejora de la versión piloto es la de ampliar el modelo de detección de objetos para incluir todos los objetos que pueden circular por la instalación. De todas estas posibilidades, la versión piloto es capaz de detectar las 22 más importantes (90% de la producción). La inclusión del resto de luminarias es claramente el siguiente punto de trabajo.
- Empleo de los datos obtenidos para mejorar el proceso: Otra de las posibles líneas de trabajo que aparecen tras la realización de este proyecto es el uso de los datos para mejorar la instalación de pintura, diseñando un sistema de identificación de perchas a lo largo de la instalación empleando dispositivos IoT con sensores como TAG's u otro tipo de sensores, que sean capaces de informar a los pintores e incluso programar las diferentes máquinas de la instalación con los programas necesarios.
- Mejora del sistema IoT: Una mejora clara al sistema de dispositivos IoT diseñado es la eliminación de cables, permitiendo al sistema más flexibilidad a la hora de desear mover la instalación en el futuro. Para ello, se podrían emplear módulos Zigbee de Arduino [8] o de Raspberry Pi para comunicar ambos dispositivos con el ordenador, si necesidad de emplear cables.

Referencias

[1] Abadi, M. B. (2016, November). *Tensorflow: a system for large-scale machine learning.* . In OSDI (Vol. 16, pp. 265-283).

[2] Brownlee, J. (2017, December). *A Gentle Introduction to Transfer Learning for Deep Learning*. Obtenido de https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[3] Bradski, G. &. (2008). Learning OpenCV: Computer vision with the OpenCV library. "O'Reilly Media, Inc.".

[4] Brownlee, J. (2017, December). A Gentle Introduction to Transfer Learning for Deep Learning. Obtenido de https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[5] Brownlee, J. (2017, December). Train Faster, Reduce Overfitting, and Make Better Predictions. Better Deep Learning.

[6] Ciresan, D. C. (2011, July). Flexible, high performance convolutional neural networks for image classification. In IJCAI Proceedings-International Joint Conference on Artificial Intelligence (Vol. 22, No. 1, p. 1237).

[7] Davies, E. R. (2004). Machine vision: theory, algorithms, practicalities. Elsevier.

[8] Faludi, R. (2010). Building wireless sensor networks: with ZigBee, XBee, arduino, and processing. " O'Reilly Media, Inc.".

[9] Fawzi, A. S. (2016). Adaptive data augmentation for image classification. In 2016 Ieee International Conference On Image Processing (Icip) (No. EPFL-CONF-218496, pp. 3688-3692). Ieee.

[10] Fukushima, K. &. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. Springer, Berlin, Heidelberg: In Competition and cooperation in neural nets (pp. 267-285).

[11] Galipienso, M. I. (2003). Inteligencia artificial: modelos, técnicas y áreas de aplicación. Editorial Paraninfo.

[12] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[13] GitHub: Tensorflow Models. (21/02/2019). Obtenido de https://github.com/tensorflow/models

[14] GitHub: Tensorflow Object Detection API. (21/02/2019). Obtenido de https://github.com/tensorflow/models/tree/master/research/object_detection

[15] Goodfellow, I. B. (2016). Deep learning (adaptive computation and machine learning series). Adaptive Computation and Machine Learning series, 800.

[16] Gubbi, J. B. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), 1645-1660.

[17] He, W. Y. (2014). Developing vehicular data cloud services in the IoT environment. IEEE Transactions on Industrial Informatics, 10(2), 1587-1595.

[18] Huang, J. R. (2017). Tensorflow object detection api. Obtenido de Code: github. com/tensorflow/models/tree/master/object detection.

[19] Hui, J. (2018). Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). Obtenido de https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

[20] Intelligent. (2018). Deep learning & Convolutional Neuronal Network: qué es y en qué consiste. Obtenido de https://itelligent.es/es/deep-learning-convolutional-neuronal-network-cnn-consiste/

[21] Karpathy, A. (2016). Cs231n convolutional neural networks for visual recognition. Neural networks, 1.

[22] Kelly, S. D. (2013). Towards the implementation of IoT for environmental condition monitoring in homes. IEEE Sensors Journal, 13(10), 3846-3853.

[23] Krizhevsky, A. S. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[24] LeCun, Y. B. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[25] Lee, I. &. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. . Business Horizons, 58(4), 431-440.

[26] Liu, W. A. (2016, October). Ssd: Single shot multibox detector. In

European conference on computer vision (pp. 21-37). Springer, Cham.
[27] McLaughlin, N. D. (2015, August). Data-augmentation for reducing dataset bias in person re-identification. In Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on (pp. 1-6). IEEE.
[28] Medina Quero, J. C. (2018). Straightforward Recognition of Daily Objects in Smart Environments from Wearable Vision Sensor.
[29] Olivas, E. S. (2009). Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques. IGI Global.
[30] Raj, B. (2018). A Simple Guide to the Versions of the Inception Network. Obtenido de https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202
[31] Raj, B. (2018). Data Augmentation | How to use Deep Learning when you have Limited Data—Part 2. Obtenido de https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced
[32] Redmon, J. D. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
[33] Ren, S. H. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).
[34] Rother, C. K. (2004). Grabcut: Interactive foreground extraction using iterated graph cuts. In ACM transactions on graphics (TOG) (Vol. 23, No. 3, pp. 309-314). ACM. Obtenido de OpenCV.
[35] Sanchez, L. M. (2014). SmartSantander: IoT experimentation over a smart city testbed. Computer Networks, 61, 217-238.
[36] Schalkoff, R. J. (1997). Artificial neural networks (Vol. 1). . New York: McGraw-Hill.
[37] Szegedy, C. I. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In AAAI (Vol. 4, p. 12).
[38] Techopedia. (21/02/2019). *Aumento de datos*. Obtenido de https://www.techopedia.com/definition/28033/data-augmentation
[39] Zeiler, M. D. (2014, September). Visualizing and understanding convolutional networks. Springer, Cham.: In European conference on computer vision (pp. 818-833).
[40] Zhu, X. L. (2017). Data augmentation in emotion classification using generative adversarial networks. . arXiv preprint arXiv:1711.00648.

TABLA I

CLASES A IDENTIFICAR

ID	NOMBRE DE LA CLASE
1	AVOVGEN2_FRAME
2	CITYSOUL GEN2_CANOPY
3	CITYSOUL GEN2_FRAME
4	CLASSICSTREET_COVER
5	CLASSICSTREET_FRAME
6	CLEARFLOOD_COVER
7	CLEARFLOOD_FRAME
8	CORELINE LARGE_COVER
9	CORELINE MEDIUM_COVER
10	DIGISTREET CATENARY_HOUSING
11	IJG_COVER
12	LUMA _COVER
13	LUMA_FRAME
14	MINILUMA_COVER
15	MINILUMA_FRAME
16	STELALONG_COVER
17	STELASQUARE_COVER
18	TUBEPOINT_HOUSING
19	UNISTREET_HOUSING

TABLA II

FILTROS FIJADOS POR CLASE

ID	NOMBRE DE LA CLASE	FILTROS
1	AVOVGEN2_FRAME	0.65
2	CITYSOUL GEN2_CANOPY	0.8
3	CITYSOUL GEN2_FRAME	0.8
4	CLASSICSTREET_COVER	0.2
5	CLASSICSTREET_FRAME	0.8
6	CLEARFLOOD_COVER	0.8
7	CLEARFLOOD_FRAME	0.8
8	CORELINE LARGE_COVER	0.8
9	CORELINE MEDIUM_COVER	0.6
10	DIGISTREET CATENARY_HOUSING	0.95
11	IJG_COVER	0.8
12	LUMA _COVER	0.98
13	LUMA_FRAME	0.98
14	MINILUMA_COVER	0.8
15	MINILUMA_FRAME	0.72
16	STELALONG_COVER	0.8
17	STELASQUARE_COVER	0.8
18	TUBEPOINT_HOUSING	0.35
19	UNISTREET_HOUSING	0.05

Development and IoT implementation of an industrial image recognition system

David Sanz Cabreros

Universidad Internacional de la Rioja, Logroño (España)

Date: 17/02/2019

Abstract

The objective of this project is to design an object recognizer based on the "Object Detection" API of TensorFlow with the purpose of applying it to an industrial level in a luminaire manufacturing plant to improve one of its manufacturing processes. The improvement proposed with respect to the current models for detecting market objects at the industrial level is the generation of said recognizer with data augmentation techniques, where from a few input images of the different pieces to be detected and of a video of the fund where they will be located. This will allow creating a sufficiently powerful model to detect the desired objects, suppressing the generation part of enormous amounts of input data and facilitating its development and updating. After its physical implementation, the transmission of information for its visualization will be done through IoT devices.

Keywords

Object Detection

TensorFlow

DataAugmentation

Industrial Artificial Vision

IoT Devices

I. Introduction

WITHIN the artificial vision, the image object detection and their identification is one of the most attractive and developed areas in recent years.

In the industrial sector there are many companies that choose this type of systems to improve their production processes, quality and add value to their products.

It is in this area of the industry in which this final master's degree project is focused, which aims to develop a system of artificial vision, based on the recognition of images, with the objective of being implemented in the production process of painting of the Signify Manufacturing Spain SL factory, located in Valladolid, in order to improve this process, generating new information for the plant on the material produced, and providing necessary data for the calculation of utilization, efficiency and quality of the process in which it will be implemented.

The innovative part that this project intends is to design an image recognition system implemented directly in an industrial environment, without large economic expenses of commercial tools, and in an agile and affordable way being able to design a sufficiently powerful system for the recognition of images from a small amount of input images. In addition, it proposes the creation of an intelligent environment within the factory through IoT devices, obtaining the information generated by the classifier and visualizing it in the desired part of the installation via the Internet without the need for wiring.

II. State of the Art

Within artificial intelligence, there are different areas or cases of use such as machine learning, natural language processing, robotics or artificial vision among others.

A. Artificial Vision:

Among all these disciplines located within artificial intelligence [11], the objective application of this work is framed within the area of artificial vision [7]. For that reason, it will be the technique that will be explained more in detail in this section. Among the tools used in this project, are:

- **Object Detection API:** For the development of typical applications such as the recognition of objects in images, speech recognition ... TensorFlow [1] provides a series of APIs [13] that allow the use of existing models.
  "Object Detection" [18] is one of these APIs that as mentioned in the official website [14] allows "locate and identify multiple objects in a single image".
- **OpenCV**: this library developed by Intel is undoubtedly one of the most important in vision worldwide [3], it includes many artificial vision functions, both for the recognition of images and for their treatment.
- **Data Augmentation**: one of the most used techniques in recent years in the world of Deep Learning, especially for the applications of artificial vision is the data augmentation.
  To solve the problem of the amount of data needed, the technique called in English Data Augmentation emerged, which consists of "adding value to the basic data by adding information derived from internal and external sources" [39]. By adding small modifications to the images such as translations, rotations, "flips", scaling ..., many images can be generated. In articles such as the one presented by Bharath Raj [31] or in books such as McLaughlin's, N, [27], Fawzi's A [9] or Zhu's, [40] explain the operation of this technique.
  The objective is to contribute to the neural network, different images of the same object with small modifications, which oblige the network to be able to generalize concepts instead of memorizing them:

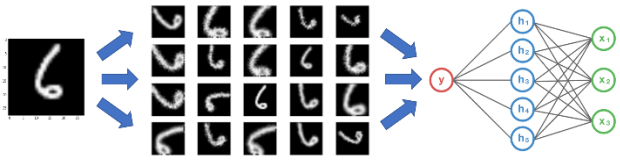

Fig. 1: Neural network with Data Augmentation

- **Transfer Learning:** this technique is an automatic learning method in which pre-trained models are used for a task, reusing it for new tasks, with the purpose of reducing the computation time required to develop models of neural networks [4] [5] [15].

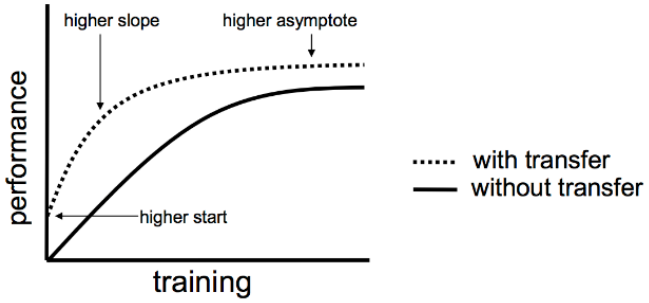

Fig. 2: Comparison of training with and without Transfer Learning

- **IoT devices**: IoT devices (internet of things) are devices that are based on the IoT concept, which refers to a digital interconnection of everyday objects with the internet. The use of this type of device and its connection through the Internet is one of the most booming technologies at present. As defined in the book "Internet of Things (IoT): A vision, architectural elements, and future directions." [16]. Creating intelligent environments where devices communicate with each other is one of the most powerful advances of recent years and one of the most important challenges that companies will have to face in the coming years to stay at the forefront of the industrial sector [25]. There are many projects based on these devices developed in recent years [17] [22] [35].

B. Object detection with Deep Learning:

Neural networks [36] are a type of computational model currently used, especially in artificial vision, where with the rise of deep neural networks, known as Deep Learning, have shown very good results in tasks of classification of objects in images.

Among the different types of existing networks, the most common are those shown below:

- Convolutional networks (CNN): it is one of the technologies that revolutionized the world of Deep Learning, and more specifically of artificial vision and the area of object recognition [6] [10] [24]. The main characteristic of these networks is their different layer structure compared to traditional feedforward networks [20] [21] [23] [39], among which are:
  - Convolutional layers.
  - Max Pooling layers.
  - Fully Connected Layers.
  -
- R-CNN and Faster R-CNN: As improvements to the models of convolutional networks, emerged models such as "Faster R-CNN" whose objective was to reduce the execution time of networks of neurons used for object detection [12] [33].
- "You Only Look Once" (YOLO): this is one of the last contributions contributed in which "they present a new approach for the detection of objects. Unlike previous models, its model frames the detection of objects as a regression problem with spatially separated delimiting frames and associated class probabilities "[32].
- "Single Shot MultiBox Detector" (SSD): As an improvement to the speed of the Faster R-CNN presented previously, in 2016 the SSD (Single Sol Multibox Detector) appeared, whose objective is to perform an image recognition as close as possible to the real time [26].

Comparing these 4 types of networks, as shown in Figure 3, one can observe the improvements that they present to each other.

Fig. 3: YOLO vs SSD vs Faster R-CNN: Accuracy vs Speed

- **Inception network**: one of the technologies that will be used in the development of work, is the one designed for the Inception network (GoogleNet) [30] [37]. The first version of this technology was intended to solve problems such as variation in the location of information between images of similar objects, problems of overfitting presented by very deep networks, which are capable of memorizing objects, or the high computational cost of very deep neural networks. To do this, he proposed using filters with multiple sizes operating at the same level, in order to obtain "wider" instead of "deeper". The different versions after this (v2 and v3) continued to improve problems such as reducing the bottlenecks produced by reducing the dimensions of the input or checking whether the auxiliary classifiers contributed enough to the final process as previously thought, or if they acted as regularizes.
  Inception version 4, as well as the Inception-ResNet version, aimed to make the modules more uniform, for which they modified the "stem", introducing in addition the "Reduction Blocks" (Reduction Blocks) used to change the width and height of the squares. To understand a little better the advantages of the different extractors of presented characteristics, an image of the article made by Jonathan Hui will be used [19].

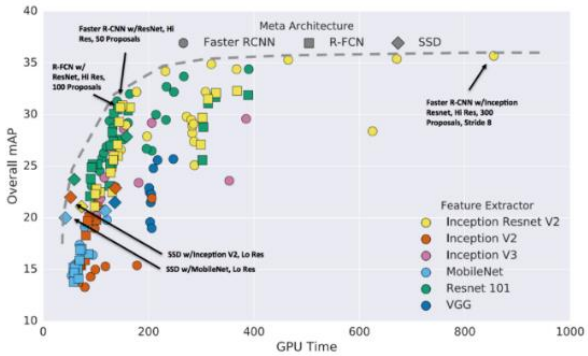

Fig. 4: Inception: Speed vs Accuracy

III. CONTRIBUTION

The objective of this final master's project is to develop a methodology for recognizing plates that will be integrated into a pilot test of a real-time detection and classification system through artificial vision, integrating the software development of the object detection algorithm in IoT devices with visual sensors, lowering the price of the current systems available in the market. The work could be divided into 5 sections:

1. Deploy existing Object Recognition tools and adapt them to the recognition of luminaire plates.
2. Use Deep Learning techniques with data augmentation and learning transfer.
3. Include IoT devices that access information to create an intelligent visual environment within the factory.
4. Design a solution using low cost systems to reduce expenses.
5. Evaluate the plate recognition system through the images collected in the real industry environment.

The steps and ideas developed during the development of the TFM are based on the ideas proposed in the research work "Straightforward Recognition of Daily Objects in Smart Environments from Wearable Vision Sensor" [28] developed by Medina Quero, Javier (tutor of the TFM), in which "they show a novel method that facilitates the visual recognition of everyday objects in intelligent environments from wearables vision sensors". This methodology has been integrated into the recognition of this project, which covers a scope of application and greater evaluation with IoT devices creating in the factory an intelligent visual space for the recognition of industrial parts. Below is an image of this work in which you can see the key ideas shown:

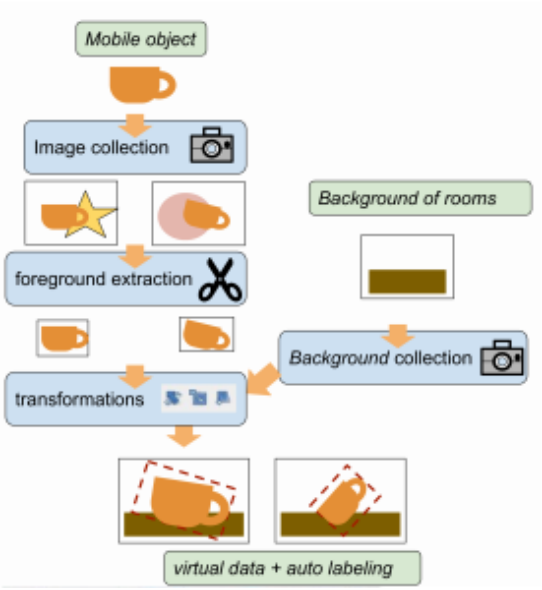

Fig. 5: Generation of input images from individual images

A. *Identification of the environment and data to be classified:*

At first, it is necessary to analyze the environment of the system where the application is intended to be implemented, as well as to understand the kinds of luminaires to be classified by the model.

Regarding the location, the system will be in an area of the factory's paint chain in which the pieces circulating along said chain cannot be accumulated one after the other. In this area, a white fabric was installed that will serve as a background to improve the accuracy of the classification model.

In the part of data analysis, after identifying all the possible elements that can circulate through the installation (154 possible combinations), a total of 19 main classes were obtained based on the production of the plant, shown in Table 1.

B. *Development of the detection system:*

With the data to be classified, the next step is to design the detection system based on the document mentioned in the state of the art [28].

1. Capture of images to classify:

For each of the identified classes, between 20-25 frontal captures are taken and from different angles like those in Figure 6:

Fig. 6: Capture of images to classify

2. Elimination of the background of the images:

With all the images captured, stored and named correctly, the next objective is to eliminate the background of these, leaving images of each of the pieces with their physical form, on a transparent background or a correctly set pixel colour.

To do this, a Python code was designed to pass all the desired images one by one by calling the Grabcut tool [34]. Figure 7 shows the process followed with an image to eliminate the background of the desired piece:

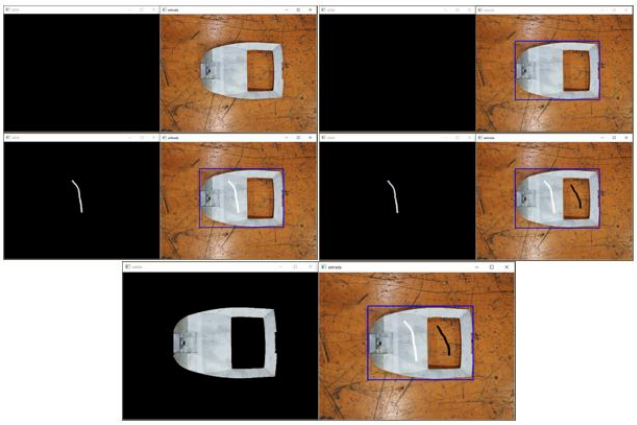

Fig. 7: Removing the image background

3. Capture of the environment:

Captured and segmented all the images of the different classes, the next step is to obtain images of the background. For this, a video of 20 seconds taken with a normal mobile camera was made, from which 15 frames of the selected area are obtained as shown in Figure 8.

Fig. 8: Frames of the environment obtained from the recorded video

4. Generation of training and test images:

For the generation of the images, a Python code was designed to superimpose each of the images of the pieces on all the images of a background, eliminating the black numbers generated in section 2 when segmenting the images.

In this part of the data augmentation, it is important to generate the most images. For this, the following transformations are applied:

- Translation: movements of the image of the piece ± 45 pixels with respect to the center of the background image.
- Rotation: turns of each image of the pieces ± 45º with respect to the center of the image of the piece, at greater than the rotation of 45º, 90º and 180º applied on all the images.
- "Flip": mirror effect on the image of the piece.
- Scaling: increase and decrease in a range of ± 50% in order to do modify the images size.

Each image is classified by the system between the image of the training or the test in a ratio of 2 images of the exam for every 8 of training.

Figure 9 shows some images generated at the system exit.

Fig. 9: Output images of the augmentation program

5. Compression of TFRecord files for training:

Once you have all the images superimposed and classified in training and test, the next step is to generate from the .csv files the corresponding files in ".record" format that will be used in the "Object Detection" API of TensorFlow as inputs and outputs.

6. Selection, adjustment and deployment of the training model:

With all available input images, the TensorFlow Object Detection API installed, and the .record files created, the next step is to define the necessary model parameters to perform the training.

For this, it is necessary to have the following files:

- Object-detection.pbtxt: This file is responsible for defining the relationship between the different classes and the corresponding ID of each of them.
- - Obj_inception.config: This file has the purpose of defining all the parameters of the model with which the training of the network is going to be carried out, among which are included:
    - Class number: 19.
    - Predefined model used in "Transfer Learning": faster_rcnn_inception_v2.
    - Location of the file.record of training and test.

With all these steps taken to prepare the training, you are already able to train the desired neural networks and evaluate the results. These steps will be explained in section IV.

IV. Results or Evaluation

With the methodology defined and its functioning checked, this section explains the two tests or evaluations carried out to verify the functioning of the trained neural networks with the augmented data generation system proposed during the project.

To do this, two evaluations are carried out, one with fewer classes to obtain more quickly a model with which to test, and in case of obtaining good results, expand it to the 19 classes indicated in the analysis part.

A. First study case of the neural network (6 classes):

In the first place, the training of a simple neural network was carried out, with only 6 physically differentiable classes with the purpose of detecting the possible problems to be considered later, as well as to validate the methodology used with data augmentation, and the software installed to carry out the Transfer Learning.

Of all the classes present in Table 1, classes 1, 2, 8, 10, 13 and 19 were selected.

The data of this training from the generation of increased data to the obtaining of the final model were the following:

- 118 original images of the bottomless pieces.

- 19 frames taken from a 25-second video.

- 21,922 training images and 385 validation images generated in 1 hour and 56 minutes.

- Training: 23675 iterations and a total of 24 hours of training in a computer with Intel Core i5 processor with CPU 2.50, S.O. Ubuntu (Linux) and 8GB RAM.

The results obtained with the test images for each of the classes and the corresponding confusion matrix were those shown in Figure 9 and 10.

ID	REAL CLASS	NUMBER OF IMAGES	TRUE POSITIVES		FALSE NEGATIVES	
1	AVOVGEN2_FRAME	36	36	100%	0	0%
2	CITYSOULGEN2_CANOPY	69	69	100%	0	0%
3	CORELINELARGE_COVER	78	72	92.3%	6	7.7%
4	DIGISTREETCATENARY_HOUSING	68	59	86.76%	9	13.24%
5	LUMA_FRAME	73	45	61.64%	28	38.36%
6	UNISTREET_HOUSING	61	61	100%	0	0%

Fig. 10: Validation data: first study case

ID CLASES		1	2	3	4	5	6
1	AVOVGEN2_FRAME	36					
2	CITYSOULGEN2_CANOPY		69				
3	CORELINELARGE_COVER	3		72			3
4	DIGISTREETCATENARY_HOUSING				59		9
5	LUMA_FRAME	6	1			45	21
6	UNISTREET_HOUSING						61

Fig. 11: Confusion matrix: first study case

From this study, apart from validating that the system for generating increased data for training was valid, a series of conclusions were considered:

- Check the visible side of each luminaire in the desired area according to the process sheet of the painting area.
- Analyse the different classifications one by one and set their confidence score filter values to add them in the final program.

B. *Second and last study case (19 classes):*

For the second case study, all the classes indicated in Table 1 were used, previously modifying the original images that showed parts of the luminaries that would not be seen, thus reducing the possible noise generated in the neural network.

The data of this case study are the following:

- 199 original images of the bottomless pieces of the 19 classes.
- 15 frames taken from a 20-second video.
- 19181 training images and 496 validation generated in 3.15 hours.
- Training: 123233 iterations and a total of 4 days and 20 hours of training with the same hardware.

The results obtained in this case of study are those shown in figure 12, in which the real data are shown:

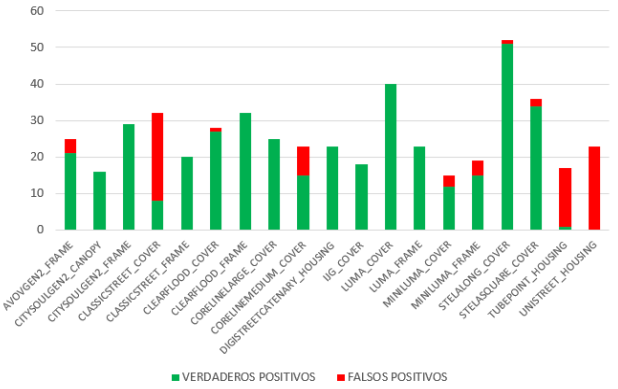

Fig. 12: Validation data: Second study case

From these data and from the confusion matrix, we obtained an average percentage per class of the positive classifications (TP) and the false negatives obtained in the erroneous classification (FN), which are shown in Figure 13.

ID	REAL CLASS	NUMBOR OF IMAGES	TRUE POSITIVES		FALSE NEGATIVES	
1	AVOVGEN2_FRAME	25	21	84%	4	16%
2	CITYSOULGEN2_CANOPY	16	16	100%	0	0%
3	CITYSOULGEN2_FRAME	29	29	100%	0	0%
4	CLASSICSTREET_COVER	32	8	25%	24	75%
5	CLASSICSTREET_FRAME	20	20	100%	0	0%
6	CLEARFLOOD_COVER	28	27	96%	1	4%
7	CLEARFLOOD_FRAME	32	32	100%	0	0%
8	CORELINELARGE_COVER	25	25	100%	0	0%
9	CORELINEMEDIUM_COVER	23	15	65%	8	35%
10	DIGISTREETCATENARY_HOUSING	23	23	100%	0	0%
11	IJG_COVER	18	18	100%	0	0%
12	LUMA_COVER	40	40	100%	0	0%
13	LUMA_FRAME	23	23	100%	0	0%
14	MINILUMA_COVER	15	12	80%	3	20%
15	MINILUMA_FRAME	19	15	79%	4	21%
16	STELALONG_COVER	52	51	98%	1	2%
17	STELASQUARE_COVER	36	34	94%	2	6%
18	TUBEPOINT_HOUSING	17	1	6%	16	94%
19	UNISTREET_HOUSING	23	0	0%	23	100%

Fig. 13: Average values TP and FN in validation: second study case

With these data the following conclusions were obtained:

- Classes that have not been classified poorly never have very high average values in correct classification.
- There are classes that, when they have been classified incorrectly, have very high average values but have rarely failed.
- Class 4 and 9 present a special case, since when it fails it does not present high values, but it does have high values when hit.
- The last case is that of classes 18 and 19, which present very low values when failing.
- It must be borne in mind that the classification of 19 such similar categories is complex and represents a challenge for existing techniques.

C. *Adjustment of the filters with validation and test data:*

With the data obtained in the previous section, an adjustment of the filters was made before putting the model into production.

With the validation images, the filters will be adjusted for each of the classes, obtaining the confusion matrix of the classification and adjusting said filters according to the most convenient metric.

- Filters at 0.5:

$$precisión = \frac{204}{204+39} * 100 = 83{,}95\%$$

$$recall = \frac{204}{204+44} * 100 = 82{,}26\%$$

- Filters at 0.1:

$$precisión\ 0{,}1 = \frac{207}{207+41} * 100 = 83{,}47\%$$

$$recall\ 0{,}1 = \frac{207}{207+41} * 100 = 83{,}47\%$$

- Filters at 0.9:

$$precisión\ 0{,}9 = \frac{166}{166+15} * 100 = 91{,}71\%$$

$$recall\ 0{,}9 = \frac{166}{166+82} * 100 = 66{,}94\%$$

With the previous tests it is observed:

- By decreasing the value of the confidence score filter, the individual precision does not improve in any case, since to classify a value as good it is necessary for the system to obtain lower precisions. In this case, no images appear that have not been able to classify, and therefore, appear more true positives. On the other hand, global Recall improves because by lowering the threshold, almost everything that appears can classify it.
- By increasing the value of the filter on the confidence score a restriction of the classification is observed. This is because being more restrictive, the classes that tend to predominate by having better learned their patterns improve their accuracy, however, the rest of the classes that need lower filter values are not able to classify correctly and decreases their Recall. Unlike before, now 67 images without identification appear.

In order to improve the system, individual filters are set per class, as shown in Table 2 and Figure 14, obtaining the following results against the validation data:

$$precisión = \frac{210}{210+11} * 100 = 95{,}02\%$$

$$recall = \frac{211}{211+37} * 100 = 85{,}08\%$$

	ID CLASSES	Filters	True Positives (TP)	False Positives (FP)	False Negatives (FN)	Precision	Recall
1	AVOVGEN2 FRAME	0,65	12	0	0	100,00%	100,00%
2	CITYSOULGEN2 CANOPY	0,8	8	0	0	100,00%	100,00%
3	CITYSOULGEN2 FRAME	0,8	15	0	0	100,00%	100,00%
4	CLASSICSTREET COVER	0,2	8	0	8	100,00%	50,00%
5	CLASSICSTREET FRAME	0,8	10	0	0	100,00%	100,00%
6	CLEARFLOOD COVER	0,8	12	2	2	85,71%	85,71%
7	CLEARFLOOD FRAME	0,8	12	0	4	100,00%	75,00%
8	CORELINELARGE COVER	0,8	12	0	0	100,00%	100,00%
9	CORELINEMEDIUM COVER	0,6	7	0	5	100,00%	58,33%
10	DIGISTREETCATENARY HOUSING	0,95	11	2	0	84,62%	100,00%
11	IJG COVER	0,8	9	1	0	90,00%	100,00%
12	LUMA COVER	0,98	20	3	0	86,96%	100,00%
13	LUMA FRAME	0,98	10	0	2	100,00%	83,33%
14	MINILUMA COVER	0,8	6	0	1	100,00%	85,71%
15	MINILUMA FRAME	0,72	7	0	3	100,00%	70,00%
16	STELALONG COVER	0,8	25	0	1	100,00%	96,15%
17	STELASQUARE COVER	0,8	17	0	1	100,00%	94,44%
18	TUBEPOINT HOUSING	0,35	5	0	3	100,00%	62,50%
19	UNISTREET HOUSING	0,05	5	3	7	62,50%	41,67%

Fig. 14: Validation results with specific filters

In order to ensure that these filters are valid for images not used so far, the test images are used. After performing the execution, it is observed that 221 of the 248 images have been detected, 89.11% of the images, of which 215 have been correctly detected, 27 being the images without identification whose percentages have not exceeded any threshold.

$$precisión = \frac{215}{215+6} * 100 = \ 97{,}29\%$$

$$recall = \frac{215}{215+33} * 100 = \ 86{,}69\%$$

D. Implementation in production with IoT devices:

Once the process of modelling the neural network was completed, the last step was its implementation in the real production environment to verify its behaviour. For this, an interconnected intelligent environment was designed as shown in figure 15.

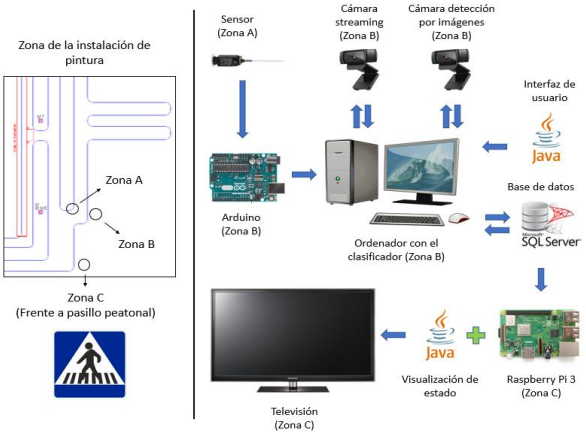

Fig. 15: Outline of the installation in production

The installation consists of 3 areas:

- **Area A:** location of the hanger sensor, located in the paint chain itself.
- **Area B:** location of the software and hardware responsible for receiving the signals from the sensor, performing the analysis of the images through the user interface and storing the data in the factory database.
- **Area C:** location of the public display monitor with the detection process (this location may vary depending on the requirements of the factory).

With the installation completed, the behaviour was analysed during the first days, observing results very close to those obtained during the modelling with the test data, with classes obtaining classification values between 85-99%. The following figure shows some images captured by the system in real time:

Fig. 16: Images classified in real time in the production environment

V. Discussion or Results Analysis

With the case studies made during the work and analysed the operation of the system in the real production environment, the following conclusions are drawn:

- The methodology shown for the generation of increased data necessary for training not only shows clear advantages in terms of the effort of collection and labelling of objects, but in the real production environment presents very valid results, being able to classify in a percentage like that obtained with artificially generated images.
- The inclusion of individual score filters for each class allows obtaining better results than those obtained without filters or with generic filters, individualizing for each class according to the classificatory capacity of the neural network.
- The increase or decrease of the individual filters allows to adjust the classifier according to the needs of the factory, depending on whether more precision or more Recall is desired. In other words, if you want the system to be able to classify when you are surer that the piece is correct at the risk of leaving unsorted images or classifying all the input images of the system at the risk of reducing the accuracy of this .

VI. Conclusion

Analysing the contributions of the project, they can be observed from the point of view of the contributions made to the company, and those contributed by the augmented data generation methodology to facilitate the creation of enough input data for the training of neural networks.

The objective of the company was to have an artificial vision system capable of detecting the parts of the installation without the need for a large financial outlay, and without needing to collect many images. The IoT system designed in the project using low-cost elements, as well as the methodology designed to generate the inputs based on data augmentation, meets the requirements of the company.

From the general point of view on systems of data augmentation, this project presents a methodology of obtaining these powerful enough to be used in industrial environments, being a model so complicated to perform for the large number of classes present with similar forms, without the use of a large number of real images, it has only been an initial requirement to have about 20-25 samples of each type of luminaire.

On the other hand, it is important to highlight the high level of engineering work deployed in the face of the need that has arisen during the development of the work to unify different techniques:

- For the preliminary analysis of the data obtained directly from the database of the factory with traditional programming based on Java and SQL language, as well as its subsequent analysis through statistical tools such as Minitab.
- For the generation of the augmented data with tools based on Python and different libraries like OpenCV.
- For its final implementation using different operating systems such as Windows 10 and Linux, IoT devices such as Arduino and Raspberry Pi, and specific software based on Java language to allow the end user to use the system in an easy way.

Regarding the future lines that appear after this project, the following are found:

- Improvement of the capture system: one of the first stages in the improvement of the pilot version is to extend the object detection model to include all the objects that can circulate through the installation. Of all these possibilities, the pilot version can detect the 22 most important (90% of production). The inclusion of the rest of the luminaires is clearly the next point of work.
- Use of the data obtained to improve the process: Another of the possible lines of work that appear after the completion of this project is the use of data to improve the installation of paint, designing a system of identification of hangers throughout the installation using IoT devices with sensors such as TAGs or other types of sensors, which are able to inform painters and even program the different machines of the installation with the necessary programs.
- Improvement of the IoT system: A clear improvement to the IoT device system designed is the elimination of cables, allowing the system more flexibility at the time of wishing to move the installation in the future. For this, you could use Arduino Zigbee [8] or Raspberry Pi modules to communicate both devices with the computer, if you need to use cables.

VII. References

[1] Abadi, M. B. (2016, November). Tensorflow: a system for large-scale machine learning. . In OSDI (Vol. 16, pp. 265-283).

[2] Brownlee, J. (2017, December). A Gentle Introduction to Transfer Learning for Deep Learning. Obtenido de https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[3] Bradski, G. &. (2008). Learning OpenCV: Computer vision with the OpenCV library. "O'Reilly Media, Inc.".

[4] Brownlee, J. (2017, December). A Gentle Introduction to Transfer Learning for Deep Learning. Obtenido de https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[5] Brownlee, J. (2017, December). Train Faster, Reduce Overfitting, and Make Better Predictions. Better Deep Learning.

[6] Ciresan, D. C. (2011, July). Flexible, high performance convolutional neural networks for image classification. In IJCAI Proceedings-International Joint Conference on Artificial Intelligence (Vol. 22, No. 1, p. 1237).

[7] Davies, E. R. (2004). Machine vision: theory, algorithms, practicalities. Elsevier.

[8] Faludi, R. (2010). Building wireless sensor networks: with ZigBee, XBee, arduino, and processing. " O'Reilly Media, Inc.".

[9] Fawzi, A. S. (2016). Adaptive data augmentation for image classification. In 2016 Ieee International Conference On Image Processing (Icip) (No. EPFL-CONF-218496, pp. 3688-3692). Ieee.

[10] Fukushima, K. &. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. Springer, Berlin, Heidelberg: In Competition and cooperation in neural nets (pp. 267-285).

[11] Galipienso, M. I. (2003). Inteligencia artificial: modelos, técnicas y áreas de aplicación. Editorial Paraninfo.

[12] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[13] GitHub: Tensorflow Models. (21/02/2019). Obtenido de https://github.com/tensorflow/models

[14] GitHub: Tensorflow Object Detection API. (21/02/2019). Obtenido de https://github.com/tensorflow/models/tree/master/research/object_detection

[15] Goodfellow, I. B. (2016). Deep learning (adaptive computation and machine learning series). Adaptive Computation and Machine Learning series, 800.

[16] Gubbi, J. B. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), 1645-1660.

[17] He, W. Y. (2014). Developing vehicular data cloud services in the IoT environment. IEEE Transactions on Industrial Informatics, 10(2), 1587-1595.

[18] Huang, J. R. (2017). Tensorflow object detection api. Obtenido de Code: github. com/tensorflow/models/tree/master/object detection.

[19] Hui, J. (2018). Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). Obtenido de https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

[20] Intelligent. (2018). Deep learning & Convolutional Neuronal Network: qué es y en qué consiste. Obtenido de https://itelligent.es/es/deep-learning-convolutional-neuronal-network-cnn-consiste/

[21] Karpathy, A. (2016). Cs231n convolutional neural networks for visual recognition. Neural networks, 1.

[22] Kelly, S. D. (2013). Towards the implementation of IoT for environmental condition monitoring in homes. IEEE Sensors Journal, 13(10), 3846-3853.

[23] Krizhevsky, A. S. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[24] LeCun, Y. B. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[25] Lee, I. &. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. . Business Horizons, 58(4), 431-440.

[26] Liu, W. A. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.
[27] McLaughlin, N. D. (2015, August). Data-augmentation for reducing dataset bias in person re-identification. In Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on (pp. 1-6). IEEE.
[28] Medina Quero, J. C. (2018). Straightforward Recognition of Daily Objects in Smart Environments from Wearable Vision Sensor.
[29] Olivas, E. S. (2009). Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques. IGI Global.
[30] Raj, B. (2018). A Simple Guide to the Versions of the Inception Network. Obtenido de https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202
[31] Raj, B. (2018). Data Augmentation | How to use Deep Learning when you have Limited Data—Part 2. Obtenido de https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced
[32] Redmon, J. D. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
[33] Ren, S. H. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).
[34] Rother, C. K. (2004). Grabcut: Interactive foreground extraction using iterated graph cuts. In ACM transactions on graphics (TOG) (Vol. 23, No. 3, pp. 309-314). ACM. Obtenido de OpenCV.
[35] Sanchez, L. M. (2014). SmartSantander: IoT experimentation over a smart city testbed. Computer Networks, 61, 217-238.
[36] Schalkoff, R. J. (1997). Artificial neural networks (Vol. 1). . New York: McGraw-Hill.
[37] Szegedy, C. I. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In AAAI (Vol. 4, p. 12).
[38] Techopedia. (21/02/2019). Aumento de datos. Obtenido de https://www.techopedia.com/definition/28033/data-augmentation
[39] Zeiler, M. D. (2014, September). Visualizing and understanding convolutional networks. Springer, Cham.: In European conference on computer vision (pp. 818-833).
[40] Zhu, X. L. (2017). Data augmentation in emotion classification using generative adversarial networks. . arXiv preprint arXiv:1711.00648.

TABLE I

CLASSES TO IDENTIFY

ID	NOMBRE DE LA CLASE
1	AVOVGEN2_FRAME
2	CITYSOUL GEN2_CANOPY
3	CITYSOUL GEN2_FRAME
4	CLASSICSTREET_COVER
5	CLASSICSTREET_FRAME
6	CLEARFLOOD_COVER
7	CLEARFLOOD_FRAME
8	CORELINE LARGE_COVER
9	CORELINE MEDIUM_COVER
10	DIGISTREET CATENARY_HOUSING
11	IJG_COVER
12	LUMA _COVER
13	LUMA_FRAME
14	MINILUMA_COVER
15	MINILUMA_FRAME
16	STELALONG_COVER
17	STELASQUARE_COVER
18	TUBEPOINT_HOUSING
19	UNISTREET_HOUSING

TABLE II

FILTERS FIXED BY CLASS

ID	NOMBRE DE LA CLASE	FILTROS
1	AVOVGEN2_FRAME	0.65
2	CITYSOUL GEN2_CANOPY	0.8
3	CITYSOUL GEN2_FRAME	0.8
4	CLASSICSTREET_COVER	0.2
5	CLASSICSTREET_FRAME	0.8
6	CLEARFLOOD_COVER	0.8
7	CLEARFLOOD_FRAME	0.8
8	CORELINE LARGE_COVER	0.8
9	CORELINE MEDIUM_COVER	0.6
10	DIGISTREET CATENARY_HOUSING	0.95
11	IJG_COVER	0.8
12	LUMA _COVER	0.98
13	LUMA_FRAME	0.98
14	MINILUMA_COVER	0.8
15	MINILUMA_FRAME	0.72
16	STELALONG_COVER	0.8
17	STELASQUARE_COVER	0.8
18	TUBEPOINT_HOUSING	0.35
19	UNISTREET_HOUSING	0.05