SOLUCIÓN IOT PARA PROMOCIÓN DEL TURISMO EN SMART CITIES MEDIANTE TÉCNICAS DE VISIÓN POR COMPUTADOR EN ENTORNOS CLOUD

IOT SOLUTION FOR TOURISM PROMOTION IN SMART CITIES USING COMPUTER VISION TECHNIQUES IN CLOUD ENVIRONMENTS

TRABAJO FIN DE MÁSTER
CURSO 2020-2021

AUTOR
FRANCISCO BURRUEZO ARANDA

DIRECTOR
GONZALO PAJARES MARTINSANZ

MÁSTER EN INTERNET DE LAS COSAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

SOLUCIÓN IOT PARA PROMOCIÓN DEL TURISMO EN SMART CITIES MEDIANTE TÉCNICAS DE VISIÓN POR COMPUTADOR EN ENTORNOS CLOUD

IOT SOLUTION FOR TOURISM PROMOTION IN SMART CITIES USING COMPUTER VISION TECHNIQUES IN CLOUD ENVIRONMENTS

TRABAJO DE FIN DE MÁSTER EN INTERNET DE LAS COSAS
DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL

AUTOR
FRANCISCO BURRUEZO ARANDA

DIRECTOR
GONZALO PAJARES MARTINSANZ

CONVOCATORIA: FEBRERO 2021
CALIFICACIÓN: 9.5 SOBRESALIENTE

MÁSTER EN INTERNET DE LAS COSAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

01 DE FEBRERO DE 2021

DEDICATORIA

Eternamente os agradeceré a vosotros, mi familia, haberme permitido llegar hasta aquí. Familia a la que tú perteneces.

AGRADECIMIENTOS

Quiero agradecer a mi director del Trabajo de Fin de Máster, Gonzalo Pajares, haberme concedido tiempo, material y consejo para ejecutar este trabajo.

A los autores de la especificación del modelo de detección de objetos, YOLOv3, Joseph Redmon y Ali Farhadi, por la relevancia que tiene su aplicación en este trabajo; así como a Anton Muehlemann y a Huynh Ngoc Anh por sus respectivas implementaciones de este modelo. Gracias.

En general, doy las gracias a la comunidad del software libre por sus distintas aportaciones a este trabajo.

RESUMEN

Solución IoT para promoción del turismo en Smart Cities mediante técnicas de Visión por Computador en entornos Cloud

En el contexto del Internet de las Cosas (IoT, *Internet of Things*), y en particular de las Ciudades Inteligentes (*Smart Cities*), se ha diseñado y desplegado en distintos entornos *Cloud* una solución que permite a cualquier comercio/institución publicar, a través de distintos canales, información que pudiera resultar de interés a fin de promocionar los servicios que ofrece independientemente de la actividad que desarrolle.

Toda esta información podrá ser consultada por los viandantes que se encuentren frente a ellos mediante una fotografía donde se señale con el dedo índice al panel informativo que los identifica. Se ha desarrollado un modelo de detección de objetos capaz de identificar a partir de estas imágenes, en cooperación con servicios de reconocimiento de caracteres, la fuente de información solicitada a fin de recuperarla. El módulo de detección se ha diseñado utilizando tecnologías de Aprendizaje Profundo, concretamente el modelo YOLOv3, como método eficiente.

Con el fin de explotar las capacidades de este modelo, y de recuperar la información publicada de antemano, se ha provisionado en entornos sin servidor un servicio diseñado para atender a las distintas solicitudes de los usuarios de la solución que este trabajo propone.

La solución conceptual desarrollada permite verificar su validez en el contexto propuesto y bajo el paradigma IoT, quedando así disponible para su adaptación y despliegue en entornos reales.

Palabras clave

Internet de las Cosas, Inteligencia Artificial, Visión por Computador, Detección de objetos en imágenes, Reconocimiento de caracteres en imágenes, Ciclo de vida de modelos basados en Aprendizaje Automático, Contenedores Software, Servicios Cloud, Infraestructura Cloud, Entornos sin servidor.

ABSTRACT

IoT solution for tourism promotion in Smart Cities using Computer Vision techniques in Cloud environments

In the context of the Internet of Things (IoT), and in particular of the Smart Cities, a solution has been designed and deployed in different Cloud environments that allows any company/institution to publish, through different channels, information that may be of interest to promote the services it offers, regardless of the activity it develops.

All this information will be able to be consulted by the pedestrians in front of them by a photograph where they point with their index finger at the information panel that identifies them. An object detection model has been developed capable of identifying from these images, in collaboration with the character recognition services, the source of information requested to recover it. The detection module has been designed using Deep Learning technologies, specifically the YOLOv3 model, as an efficient method.

In order to exploit the capabilities of this model, and to recover the information published in advance, a service designed to meet the requests of users of the solution proposed in this work has been provided through serverless environments.

The conceptual solution developed allows to verify its validity in the proposed context and under the IoT paradigm, thus remaining available for its adaptation and deployment in real environments.

Keywords

Internet of Things, Artificial Intelligence, Computer Vision, Object detection on images, Optical character recognition, Machine Learning lifecycle, Software containers, Cloud services, Cloud infrastructure, Serverless environments.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Escenario y propuesta

Es habitual que nos preguntemos a diario acerca de las condiciones meteorológicas de nuestra ciudad sobre la fluidez del tráfico o la disponibilidad de plazas de aparcamiento en nuestro lugar de trabajo. Por otra parte, mientras nos desplazamos a primeras horas de la mañana, podemos observar cómo el alumbrado público se apaga cuando brotan los primeros rayos de sol y los sistemas de regadío de parques y jardines comienzan a funcionar.

Este es sólo un ejemplo del papel que juega el paradigma conocido como Internet de las Cosas (IoT, *Internet of Things*) en nuestras ciudades en el día a día. Ha quedado demostrado que su explosión tecnológica y de aplicabilidad ha permitido a los organismos e instituciones públicas ser más eficaces en el uso de los recursos energéticos y logísticos al mismo tiempo que han proporcionado a sus ciudadanos información que resulta de gran utilidad en su vida cotidiana. El concepto Ciudad Inteligente (*Smart City*) proviene de la capacidad que bien puede disponer una ciudad, una villa o un pueblo para recoger información del medio y compartirla con la comunidad o los servicios de mantenimiento en general, a la vez que aplicar o realizar acciones en función de esta.

Las distintas áreas de investigación dentro de la Inteligencia Artificial (IA) constituyen un gran activo dentro del contexto IoT. El beneficio de la recopilación de información reside en muchos casos en el hecho de disponer de modelos basados en IA que permitan tomar decisiones acertadas a partir de estos datos. En ocasiones, la información resultante y que es empleada para distintos fines tiene como origen la predicción basada en los modelos establecidos al efecto.

Este trabajo realiza una propuesta en el ámbito de las Ciudades Inteligentes cuya finalidad es la de ampliar los servicios que estas pueden ofrecer poniendo el foco en el sector turístico, haciendo uso de modelos basados en IA, concretamente utilizando el modelo YOLOv3 (Redmon y Farhadi, 2018) como modelo detector de objetos en el ámbito del Aprendizaje Profundo. Se añade así una nueva propuesta a las mencionadas

previamente en el ámbito de IoT. Concretamente, este trabajo propone publicar en entornos *cloud* recursos que permitan a una Ciudad Inteligente publicar información relevante sobre los servicios que presta a través de protocolos IoT que se encuentran en el estado del arte, clientes de mensajería y redes sociales además de servicios web que permitan a los viandantes consumir esta información.

Un restaurante local podrá publicar su horario y su oferta además de su aforo y las condiciones ambientales de su interior, pudiendo ser recopilada esta información gracias al despliegue de una red IoT, mientras que una catedral podrá ofrecer a sus visitantes audioguías para que puedan ser escuchadas directamente en un teléfono móvil, por citar algunos ejemplos. El consumidor de estos servicios solo tendrá que realizar una fotografía señalando con su dedo índice al panel informativo que los identifica para que, con un modelo de detección de objetos en cooperación con servicios de reconocimiento de caracteres, pueda detectarse cuál es la fuente de información solicitada para proporcionarle todo aquello que pudiera haber sido publicado sobre ella. Justamente, es aquí donde se enfoca el desarrollo y la aplicación del presente trabajo, de forma que tras la captura de la imagen conteniendo el panel informativo de interés señalado, se procede a su procesamiento mediante técnicas basadas en Aprendizaje Profundo y utilizando servicios IoT apropiados.

1.2 Objetivos, motivación y alcance

El IoT ofrece la capacidad de realizar aportaciones significativas dentro de la industria del turismo, a favor ya no solo de las empresas del sector sino de la experiencia de los turistas. Una aeronave comercial puede estar equipada con sensores de viento y temperatura que permitan al piloto tomar la decisión de ahorrar combustible bajo determinadas circunstancias y un tren podría contar con muchos otros sensores que permitirían cuantificar el desgaste de los componentes de este a fin de realizar mantenimientos preventivos que puedan incurrir en un ahorro económico, por citar algunos ejemplos. En contraposición, también resultaría viable que los asistentes de vuelo suministrasen a sus pasajeros sensores capaces de medir el ritmo cardiaco y la sudoración a fin de llegar a conocer el nivel de ansiedad de estos para proporcionales toda la atención que pudieran necesitar, mientras que en un tren también resultaría

posible proporcionar interfaces que, de un modo centralizado, permitan a los pasajeros modificar las condiciones ambientales del vagón en el que se encuentran, también por citar algunos ejemplos.

Este trabajo se centra en la experiencia de turista y en particular de dotar a estos de la capacidad de poder solicitar información que es suministrada de antemano por diversas fuentes; que bien puede ser proporcionada por redes de sensores o por la publicación de la propia oferta de los distintos servicios de los lugares de destino.

Cuando tomamos la decisión de hacer turismo, por norma general, habituamos a reservar alojamiento, a contratar un medio de transporte y a informarnos sobre las ofertas de ocio de nuestro destino. Hoy en día son numerosos los portales públicos disponibles en internet como Tripadvisor (2020) o Yelp (2020), por citar algunos ejemplos, que disponen de información turística de destinos potenciales al mismo tiempo que promocionan y ofrecen sus servicios. Es frecuente encontrar que estas fuentes cuentan con reseñas y opiniones de usuarios que ponen de manifiesto la calidad de las experiencias ofertadas. La venta de servicios de viajes suele implicar que el comprador necesite recopilar mucha información sobre su destino, pero esa necesidad persiste aún después de realizar las contrataciones oportunas.

La actividad que lleva a cabo la industria del turismo puede definirse como la orquestación de diversos actores (gobierno anfitrión, atracciones, restauración, comercio local, artesanía, cultura, religión, etc.) en beneficio de la experiencia del turista. Todos estos actores acostumbran a publicar la información de los servicios que ofrecen en determinados portales en función del sector en que basan su actividad lo que provoca que la búsqueda de esta información puede no resultar un procedimiento ágil si nos encontramos ya en nuestro destino al encontrarse diversificada. Este trabajo propone que toda información que pueda resultar de interés quede centralizada, que los propios responsables de los servicios que se ofertan sean quienes deciden qué mostrar en cada momento y que la información publicada sea lo más accesible posible a los turistas.

En el momento que se le permite a un negocio poder publicar, modificar o eliminar información se le está concediendo el privilegio de ser dueño de esa información lo que le puede permitir explotarla a su antojo. Por otra parte, al encontrarse

esta centralizada puede resultar beneficioso desde el punto de vista del ahorro del tiempo empleado para obtenerla mejorando la experiencia del turista; siendo el tiempo un factor muy valioso en periodos vacacionales.

La naturaleza, sin ánimo de lucro, de las oficinas de turismo juegan un papel diferencial en la promoción del propio territorio al que pertenecen como conjunto pudiendo potenciar a cada uno de estos actores independientemente de la actividad que desarrollen. Además, es frecuente encontrar información procedente de grandes ciudades, lo que no sucede con otras más pequeñas lo que puede suponer una gran promoción de todos los servicios que puede ofrecer. Asumiendo su papel, la motivación de este trabajo es la de permitir que cualquier interesado pueda publicar información sobre los servicios que ofrece y que esta pueda ser compartida al viandante en tiempo real cuando se encuentre frente a cada uno de los establecimientos donde se ofrecen los servicios. Desde el punto de vista de IoT, esta propuesta permite la consulta de información publicada en tiempo real en relación con un elemento de interés, permitiendo al turista tomar decisiones in situ.

Por exponer un escenario de utilización, un turista puede consultar la ocupación de la barra de la taberna a la que tenía planificado visitar, comprobar que está llena sin necesidad de entrar, y al encontrarse la información de todos los servicios centralizada, puede decantarse por visitar el comercio local más próximo que oferta, a través de la solución que se propone en este trabajo, productos artesanales regionales a precio reducido mientras espera a que el nivel de ocupación de la taberna disminuya.

Presentada la propuesta y su elemento motivacional, se definen los siguientes objetivos generales delimitando así el alcance del proyecto:

- Ofrecer una metodología que permita a los interesados, dentro del sector turístico, promocionar su oferta con herramientas que les pueda suponer un bajo impacto económico y que no precise necesariamente de perfiles expertos en el ámbito de las tecnologías de la información.
- Permitir que un potencial consumidor de estos servicios envíe una imagen de la región o superficie sobre la que desea recibir información a un servidor central para que pueda ser procesada.

- A partir de una instantánea, detectar cuál es la información solicitada por el usuario y recuperarla para que pueda ser compartida. Dado que esta imagen podrá contener un número variable de los servicios registrados, en la fotografía deberá aparecer un dedo índice apuntado a un panel informativo permitiendo al sistema discernir entre posibles regiones ambiguas.

Se ilustra en la Figura 1.1 el diseño conceptual de la solución IoT propuesta.

Figura 1.1 Diseño conceptual de la solución IoT propuesta

Para cumplir estos objetivos se tiene en consideración que la actividad que desarrollan estas instituciones no pertenece al área de servicios de la información. Se asume, por tanto, que ninguna de ellas cuenta con sistemas propios capaces de ejecutar la solución que se plantea. Dado que algunos territorios centran su actividad turística de forma estacional, es conveniente tener la capacidad de escalar la capacidad de cómputo de la solución propuesta en función de la demanda de los usuarios según la estación del momento.

Ambas asunciones conducen a tomar la decisión de consumir servicios en la nube a fin de ahorrar costes desplegando los recursos necesarios en cada momento y delegando el mantenimiento y la seguridad de los sistemas sobre los proveedores.

1.3 Objetivos específicos y plan de trabajo

El plan de trabajo se rige por la descomposición de la solución en los distintos componentes que la forman y en la evaluación de los requerimientos de cada uno de ellos teniendo en consideración el alcance de este trabajo.

Para alcanzar los objetivos generales descritos se plantean los siguientes pasos, que constituyen a la vez los objetivos específicos de la propuesta:

- Módulo Inteligencia Artificial
    - Generación del *dataset* de imágenes donde aparezcan paneles informativos que permitan identificar el negocio al que hacen referencia. De forma adicional, este contendrá instantáneas de dedos índice donde quede de manifiesto que se encuentran señalando hacia el objeto de interés.
    - Utilización de herramientas que permitan etiquetar imágenes que contengan los elementos citados.
    - Entrenamiento de un modelo de red neuronal convolucional de detección de objetos que permita identificar la aparición de cada una de las etiquetas contempladas.
    - Despliegue de servicios que permitan hacer un seguimiento de la precisión de los modelos entrenados, reproducir el entorno de ejecución y sean capaces de provisionar aquellos modelos candidatos.
- Módulo Propietario de Servicios
    - Publicación de distintos canales que permita a los distintos usuarios promocionar su oferta de servicios.
    - Despliegue de bases de datos que contendrán información de estos usuarios y servicios.
- Módulo Consumidor de Servicios

- Desarrollo de servicio web que atienda a las peticiones de los potenciales consumidores.
- Publicación del servicio web en entornos serverless.
- Explotación del módulo de detección de objetos previamente entrenado.
- Consumo de modelo capaz de convertir los caracteres presentes en las imágenes a texto plano.
- Implementación de comunicaciones con el Módulo de IA y con el Módulo Propietario de servicios para realizar inferencia sobre los modelos entrenados y devolver a los usuarios la información demandada recuperada de las bases de datos.

La Figura 1.2, ilustra la composición e integración de los módulos especificados, con sus correspondientes conexiones que constituyen el esquema de la solución IoT propuesta.

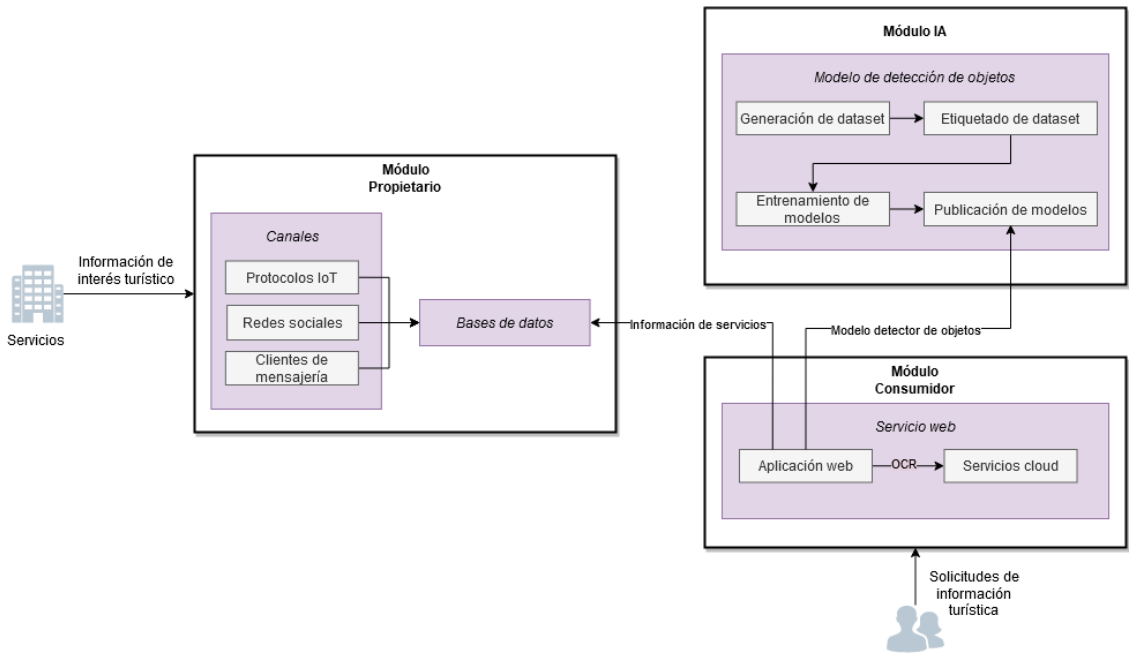


Figura 1.2 Esquema general de la solución IoT propuesta

Un usuario concreto, realiza una solicitud de información turística mediante la captura de una imagen que suministra al sistema a través del Módulo Consumidor. El

Módulo Propietario facilitará la publicación de información de interés turístico a través de distintos canales y se encargará de almacenarla en bases de datos. El Módulo Consumidor hace de interfaz, permitiendo acceder a esta información. Gracias a la publicación del modelo de detección de objetos, provisionado por el Módulo IA, será posible identificar, para luego recuperar, la fuente de información solicitada por los usuarios de la solución propuesta en este trabajo.

1.4 Organización de la memoria

La memoria se organiza como sigue:

Capítulo 2: Revisión de tecnologías *open source* que permiten satisfacer los requisitos técnicos de la solución y de aquellas cuyo marco teórico se encuentran dentro de las líneas de investigación en Inteligencia Artificial que hacen posible cumplir el propósito de este trabajo.

Capítulo 3: Diseño de las distintas partes que forman el proyecto. Este capítulo cubre la fase de diseño de los modelos basados en Inteligencia Artificial, así como el propio de la arquitectura y sus componentes software que hacen posible la interacción de los distintos usuarios con el sistema propuesto.

Capítulo 4: Evaluación de la solución. Tiene como objetivo ilustrar la eficacia de los modelos propuestos desde una perspectiva analítica hasta el rendimiento de la plataforma propuesta en su conjunto desplegada en distintos entornos *cloud*.

Capítulo 5: Exposición de conclusiones obtenidas y planteamiento de posible trabajo futuro.

Capítulo 2 - Marco teórico y tecnológico

Como paso previo a detallar el diseño de la solución propuesta, este capítulo enumera y describe todos los componentes de la arquitectura que se propone en términos de las tecnologías *open source* y *cloud* que la hacen posible y del marco teórico subyacente.

Queda dividido en 3 apartados bien diferenciados:

- En primer lugar, en el apartado 2.1, se realiza una revisión sobre la aportación de las distintas líneas de investigación en Inteligencia Artificial dentro de la especialización del Aprendizaje Profundo haciendo énfasis en la problemática de la detección de objetos y del reconocimiento de caracteres; aquellas que emplea este trabajo.
- A continuación, en el apartado 2.2, se presentan soluciones software que permiten definir entornos de entrenamiento, hacer seguimiento de estos y que ofrecen interfaces con las que poder publicar a modo de servicio los modelos basados en Inteligencia Artificial resultantes.
- En el apartado 2.3 se realiza un análisis sobre las tecnologías de contenedores existentes que hacen posible el empaquetado de los distintos componentes software requeridos en este trabajo también aquí descritos.
- Finalmente, identificados los requerimientos técnicos de la solución, en el apartado 2.4 se hace una evaluación sobre los servicios que ofrecen las principales plataformas *cloud* para satisfacerlos.

2.1 Aprendizaje Profundo y visión por computador

El Aprendizaje Profundo, conocido a nivel internacional como *Deep Learning*, es un área de la Inteligencia Artificial que hace posible la extracción automática de características que pueden encontrarse presentes en grandes volúmenes de datos.

Siendo esta su principal singularidad, las técnicas que quedan descritas bajo el concepto tienen distintos ámbitos de aplicación. Desde la detección de fraudes en movimientos bancarios, hasta la generación de resúmenes y clasificación de textos o la

integración de experiencias de usuario personalizadas en servicios web, entre muchos otros.

No obstante, a riesgo de que existe un diverso conjunto de formas de uso, su aplicación en la visión por computador es probablemente el más recurrente y donde ha experimentado un crecimiento exponencial en los últimos años, entrando por supuesto en el ámbito de IoT. El término visión por computador recoge procedimientos que permiten el procesamiento y la generación (*data augmentation*) de imágenes con distintos fines. En particular ha sido preciso hacer uso de ellos, para poder cumplir con los objetivos definidos en este trabajo, consistentes en detectar el servicio al que hace referencia un panel informativo en la vía pública, captado convenientemente mediante una cámara digital, para proporcionar al usuario información de interés turístico.

En el subapartado 2.1.1, se describe la problemática de la detección de objetos en imágenes y la aportación de algunas de las técnicas que ofrece a este trabajo mientras que en el subapartado 2.1.2 se presentan metodologías y servicios *cloud* que permiten el reconocimiento de caracteres impresos en imágenes. Será en el Capítulo 3 donde queden ilustrados los procedimientos que se llevan a cabo a fin de permitir explotar estos recursos mientras que en el Capítulo 4 se realizará una evaluación sobre las capacidades de estos.

2.1.1 Detección de objetos

Una de las aplicaciones de la visión por computador es la detección de objetos en imágenes mediante Redes Neuronales Convolucionales. En este trabajo se hace uso de este modelo de red con el fin de detectar:

1. El dedo índice de ambas manos de personas.
2. Paneles informativos.

En términos generales, el objetivo de los modelos de detección de objetos es delimitar regiones de una imagen susceptibles de contener objetos para después clasificar los elementos que pudiera haber en ellas.

La arquitectura que presentan las redes de detección de objetos consta, generalmente, de dos partes principales (Bochkovskiy y col., 2020). Una columna vertebral, *backbone*, definida por sucesivas capas de convolución que permiten extraer características de las imágenes de entrada y una cabeza que se encarga de determinar la probabilidad con la que se encuentra un objeto dentro de una determinada región, señalando esta. En función de si se procesa la imagen completa o si se realiza el mismo procedimiento sobre distintas regiones por separado, podemos afirmar que la cabeza pertenece a la categoría de lo que se conoce como un estado o de dos estados, respectivamente.

En cuanto a la columna vertebral, el estado del arte, Zhao y col. (2019), cuenta ya con varios detectores (VGG, ResNet, DenseNet o DarkNet por poner algunos ejemplos) mientras que, por otra parte, resulta frecuente rediseñar la cabeza de la red para generar un modelo que permita satisfacer los objetivos propuestos. El principal beneficio de usar una de estas columnas vertebrales es el poder disponer de modelos pre-entrenados, es decir, podemos tomar de partida una red que permite detectar y clasificar decenas de elementos que, en muchos casos, pueden encontrarse a diario en el devenir del día a día. El término *transfer learning* se emplea para referirse al proceso de entrenar una red previamente entrenada con nuevos datos a fin de ajustar los pesos de las capas convolucionales permitiendo clasificar objetos de nuevas categorías aprovechándose de características que ya han sido previamente detectadas.

Por citar algunos conjuntos de imágenes de dominio público que son utilizadas para entrenar estas arquitecturas, encontramos ImageNet (2012, 2014, 2020) que cuenta con 1000 clases de objetos, COCO (2020) con 80 elementos o Pascal VOC (Everingham y col., 2015) con 20 etiquetas distintas. Realizar *transfer learning* resulta beneficioso en el contexto de este trabajo dado que las imágenes que se utilizarán en la fase de entrenamiento podrán contener estos elementos y podrá favorecer el hecho de que la red minimice significativamente el error en la predicción después de un corto proceso de entrenamiento.

Estos conjuntos de datos se encuentran debidamente etiquetados para entrenar modelos de detección de objetos. Si deseamos formar uno nuevo, debemos especificar por cada una de las nuevas imágenes una etiqueta que enumere las clases de los

objetos que pueden encontrarse en ella y los rectángulos que los contienen definidos por sus coordenadas. Cada uno de estos rectángulos se le denomina *bounding box*.

Simplificando en primera instancia, el proceso de entrenamiento de un modelo de detección de objetos consiste en encontrar funciones $f_x(a), f_y(a), f_w(a), f_h(a)$, tales que dado un rectángulo $a = (a_x, a_y, a_w, a_h)$ con $a \in A$, pueda transformarse en otro rectángulo $b = (b_x, b_y, b_w, b_h)$ con $b \in B$; siendo x, y las coordenadas del centro del rectángulo mientras que $w\ y\ h$ representan el ancho y el alto, respectivamente. Cada uno de los rectángulos de A son ejemplos de entrenamiento y se denominan *anchor boxes* mientras que los rectángulos contenidos en B son los *bounding boxes* etiquetados de antemano; con frecuencia denominados estos últimos como *ground truth bounding boxes.* En la Figura 2.1 se muestran sendos rectángulos de esta naturaleza, ubicados sobre paneles informativos de tiendas a partir de los cuales se extrae la información relativa al establecimiento que lo identifican.

Figura 2.1 Ejemplos representativos de bounding boxes

Un método utilizado para obtener una gran cantidad de *anchor boxes* es generar por cada píxel de la imagen $p_i(x, y)$ con $p \in P$, siendo P un conjunto de píxeles de una imagen, un rectángulo con ancho w y alto h. A cada uno de estos rectángulos pueden aplicarse distintas razones de aspecto r y coeficientes de escala s a fin de generar nuevos rectángulos a partir de un mismo punto. Normalmente suele seleccionarse un subconjunto de valores que pueden tomar r y s a fin de reducir el número de *anchor boxes,* así como aplicar un factor de desplazamiento d a la hora de seleccionar cada píxel p que formará parte de P sobre el ancho y el alto de la imagen; de este modo

obtendremos un conjunto de ejemplos de entrenamiento con un volumen computacionalmente asumible. Por cada *anchor box* se recopilará la clase del objeto que se encuentra contenido en él además de la distancia o desplazamiento respecto al *ground truth bounding box*.

Para hacer una selección de *anchor boxes* que más se ajustan a estos últimos y poder así etiquetarlos se recurre al concepto *Intersection over Union (IoU)* para definir el grado de similitud entre los rectángulos de A con los de B, según se indica en la Figura 2.2 (Pajares y col., 2021).

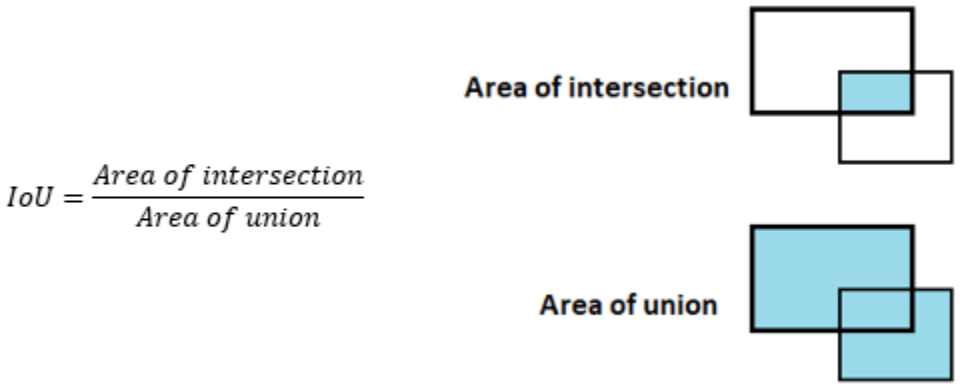


Figura 2.2 Ilustración del cómputo del índice IoU

Cuando $IoU = 1$ un anchor box representa exactamente a un *ground truth bounding box* mientras que si $IoU = 0$ ningún píxel contenido en un determinado anchor box pertenece al conjunto de píxeles contenidos en un *ground truth bounding box en particular;* en este último caso al *anchor box* se le asigna la categoría de fondo. Después de computar todos los valores de IoU entre cada rectángulo en A y cada rectángulo en B se termina por calcular el desplazamiento de cada *anchor box* a que maximiza el IoU respecto a un *ground truth bounding box* b y se le asigna la clase del objeto que contiene. Es frecuente determinar un umbral mínimo de IoU que debe ser superado a fin de realizar la mejor selección de *anchor boxes* posible. En la imagen de la Figura 2.3 pueden verse 2 *anchor boxes* propuestos que maximizan el *IoU* sobre un *bounding box* en particular.

Figura 2.3 Anchor boxes que maximizan el IoU

Una vez finalizada la fase de entrenamiento, dada una imagen de entrada se generarán *anchor boxes* sobre los que se predecirá la categoría del objeto que contiene y el desplazamiento sobre el *bounding box*. El concepto de supresión no máxima se conoce como el procedimiento por el cuál, se seleccionan únicamente *anchor boxes* que predicen *bounding boxes* que, una vez que se procesan esas regiones calculadas a partir del desplazamiento predicho, maximizan la probabilidad en la predicción de que se encuentre un objeto de una determinada clase dentro de ellos. En la imagen de la Figura 2.4 se muestran tres ejemplos de predicción con sus correspondientes valores de probabilidad de pertenencia a una determinada clase p, donde uno de ellos queda suprimido por el procedimiento de supresión no máxima.

Figura 2.4 Ejemplos de bounding boxes con sus valores de predicción

A fin de realizar la fase de evaluación sobre un modelo de detección de objetos entrenado hay que tener en consideración la relación de métricas que permitirán cuantificar su eficacia. El cálculo de cada una de ellas parte de haber realizado la contabilidad de los siguientes casos una vez sometido el modelo a distintas imágenes de *test*:

- Verdaderos positivos (TP): Predicciones del modelo donde la predicción de clase es acertada, además la probabilidad de pertenencia a una determinada clase supera un umbral mínimo, y el índice *IoU* sobre el *bounding box* propuesto y el *ground truth bounding box* supera también cierto límite.
- Falsos positivos (FP): No se cumple uno de los dos criterios presentados con anterioridad para considerar que un caso es verdadero positivo; predicción de clase incorrecta o índice de *IoU* sobre el *ground truth bounding box* que no supera el umbral.
- Verdaderos negativos (TN): Detecciones de objetos donde se presentan predicciones de clase que no cumplen el criterio de probabilidad mínima de pertenencia de clase en áreas donde no existe ningún *ground truth bounding box*.
- Falsos negativos (FN): Los *bounding boxes ground truth* que no son asociados a ninguna detección, ni verdaderos positivos ni falsos positivos, se consideran en este caso falsos negativos.

La suma entre los verdaderos positivos y los falsos negativos tiene como resultado el total de objetos presentes en una imagen que el modelo debería ser capaz de detectar y clasificar; esto es, el total de *ground truth bouding boxes (GT)*. En base a esta diferenciación sobre las predicciones de un modelo de detección de objetos pueden definirse las métricas *Recall* y *Precision* cuyo cálculo se realiza a partir de la relación entre distintos casos singulares que puede verse en las siguientes ecuaciones.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{GT} \tag{2.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

Teniendo esto en consideración, la métrica *Recall* determina el porcentaje de los objetos que el modelo es capaz de detectar y clasificar correctamente respecto al total de objetos presentes en la imagen. La métrica *Precision* relaciona, por otra parte, las detecciones y clasificaciones acertadas sobre la suma entre todas las detecciones realizadas, acertadas o no, sin tener en consideración el total de objetos presentes en una imagen. Ambas métricas se calculan de forma independiente para cada una de las clases contempladas por el modelo de detección de objetos.

Con el fin de relacionar ambas métricas es necesario recopilar, en orden descendente, el conjunto de probabilidades de clase obtenidas después de haber procesado todas las imágenes de test y luego, de forma iterativa, ir contabilizando los distintos casos presentados para calcular los diferentes valores de *Recall* y *Precision* donde el elemento i-ésimo de cada uno de estos conjuntos de valores es el cálculo de cada una de estas métricas después de haber contabilizado i detecciones; en el caso de los falsos negativos se asume que en la iteración i-ésima es igual a la diferencia entre el total de las detecciones posibles y los verdaderos positivos. Se define la curva PR como la relación entre estos dos conjuntos y el *Average Precision* (AP) como el área bajo esta curva después de situar a los distintos valores de *Recall* en el eje de abscisas y los propios de *Precision* en el de ordenadas. Dado que calcular la integral es computacionalmente costoso, ecuación (2.3), para calcular la precisión promedio se realiza una interpolación, como puede verse en la ecuación (2.4).

$$AP = \int_0^1 Precision(r)\, dr \tag{2.3}$$

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) * p_{int}(r_{i+1}) \tag{2.4}$$

En esta aproximación n representa el número de detecciones realizadas por el modelo, r_i hace referencia al valor de *Recall* cuando fueron contabilizadas i

detecciones y p_{int} es el máximo valor de *Precision* obtenido hasta haber sido contabilizada la detección i, incluyendo esta.

$$p_{int}(r) = \max Precision(r') \; con \; r' \geq r \tag{2.5}$$

Para relacionar las precisiones promedio (AP) de todas las clases con las que ha sido entrenado el modelo de detección de objetos se define la *mean Average Precision* (mAP), que se obtiene realizando el cálculo de la media sobre todas las precisiones promedio obtenidas de cada una de estas clases. Esta métrica es empleada con frecuencia para evaluar modelos de detecciones de objetos.

Sirviendo esto como introducción, se presentan a continuación las características particulares del modelo de detección de objetos YOLOv3 utilizado en este trabajo.

2.1.1.1 YOLOv3

Todos los conceptos presentados con anterioridad sirven de base para definir, a grandes rasgos, la arquitectura y el comportamiento general del sistema de detección de objetos a tiempo real YOLOv3 (Redmon y Farhadi, 2018).

La elección de este modelo para este trabajo fue en base a la aportación de su principal característica, su velocidad de procesamiento en fase de inferencia respecto a algunos modelos ampliamente utilizados en este ámbito, destacando Single Shot multibox Detector (SSD), y variantes, Fully Convolutional Network (FCN) basada en regiones o Feature Pyramid Network (FPN).

La Figura 2.5 muestra un análisis comparativo entre los modelos indicados, observándose cómo el tiempo de inferencia de YOLOv3 (320) son 22 ms, eso sí con algún menor desempeño en lo que respecta a la precisión promedio (mAP) frente a otros modelos, incluyendo los del tipo RetinaNet (He y col., 2016) en sus versiones 50 y 101.

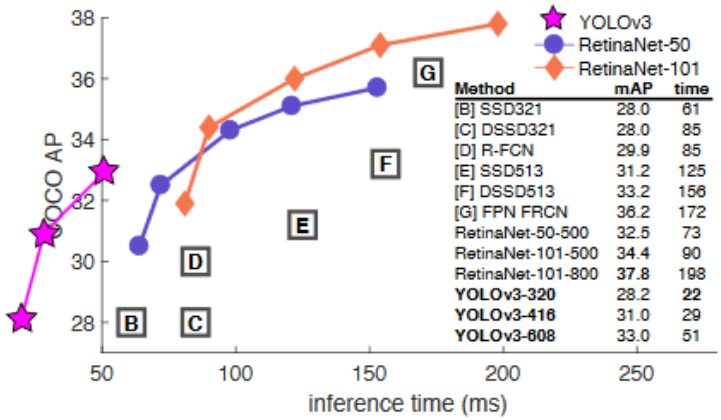


Figura 2.5 Análisis comparativo de métodos de detección de objetos (Redmon y Farhadi, 2018)

La arquitectura de la red está constituida por el *backbone* Darknet-53, con 53 núcleos convolucionales. Puede verse en la parte izquierda de la Figura 2.6 (Pajares y col., 2021) la tabla descriptiva del modelo donde queda presente la existencia de sucesivas capas de estos núcleos de tamaño 3x3 y 1x1.

	Tipo	Filtros	Dimensión/ *stride*	Salida
	Convolucional	32	3×3	256×256
	Convolucional	64	3×3/2	128×128
1×	Convolucional	32	1×1	
	Convolucional	64	3×3	
	Residual			128×128
	Convolucional	128	3×3/2	64×64
2×	Convolucional	64	1×1	
	Convolucional	128	3×3	
	Residual			64×64
	Convolucional	256	3×3/2	32×32
8×	Convolucional	128	1×1	
	Convolucional	256	3×3	
	Residual			32×32
	Convolucional	512	3×3/2	16×16
8×	Convolucional	256	1×1	
	Convolucional	512	3×3	
	Residual			16×16
	Convolucional	1024	3×3/2	8×8
4×	Convolucional	512	1×1	
	Convolucional	1024	3×3	
	Residual			8×8
	Average pool	Global		
	Conectada	1000		
	Softmax			

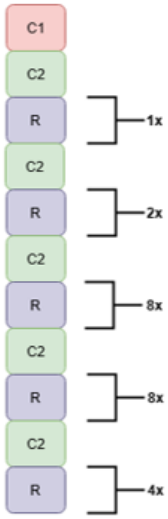


Figura 2.6 Modelo Darknet-53

Con el fin de ser más preciso en la definición de la arquitectura de la red, se distinguirán 3 conjuntos distintos de capas (C1, C2 y R1) presentes en la arquitectura. La composición completa de la arquitectura Darknet-53 donde se especifica la pertenencia de cada capa convolucional a cada uno de estos tres conjuntos definidos puede verse en la Figura 2.6 Modelo Darknet-5.

El primer conjunto (C1) está compuesto por sucesivos núcleos convolucionales (*Convolution2D*) seguido de una capa de normalización (*Batch Normalization*) y de una función de activación *ReLU* (*Leaky ReLU)*, operaciones comúnmente utilizadas en las redes neuronales convolucionales (Pajares y col., 2021).

Las capas convolucionales son la característica principal de las redes neuronales convolucionales, el tipo al que YOLOv3 pertenece. La operación de convolución permite extraer características de una imagen y consiste en combinar dos funciones para formar una tercera a partir de estas para obtenerla.

$$S(i,j) = (I * K)(i,j) \tag{2.6}$$

En este caso I representa a un vector o matriz de entrada y K al núcleo de convolución, o filtro, que contendrá la matriz de pesos de la red que irá reajustando sus valores durante la fase de entrenamiento. La salida de cada operación de convolución S se denomina mapa de características. La Figura 2.7 ilustra la operación de convolución en sí misma. Un filtro K, representado por el área sombreada dentro de la matriz de color verde, de tamaño 3x3 se va desplazando a lo ancho y largo (i,j) de una matriz I de tamaño 6x7, donde en cada iteración se realiza el producto entre ambas para obtener finalmente el mapa de características S, tal y como se representa en la parte inferior de la mencionada imagen.

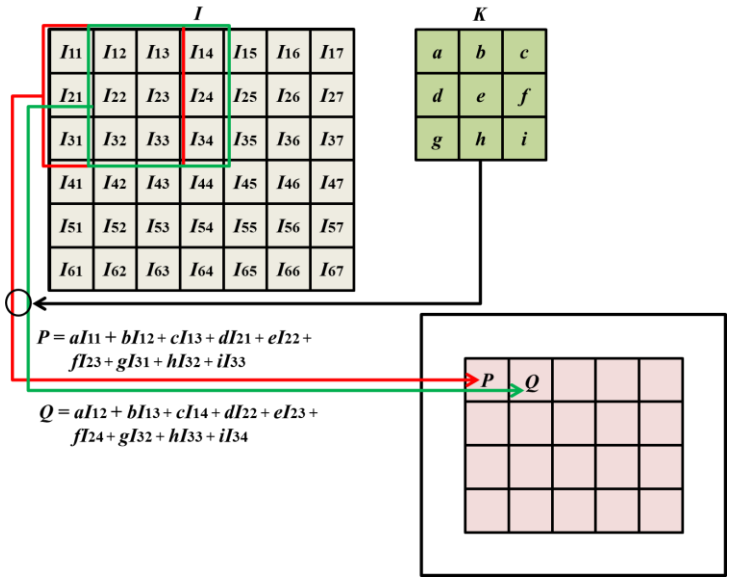


Figura 2.7 Operación de convolución con un núcleo de dimensión 3×3

En la tabla de la Figura 2.6, la columna *Filtros* indica el número de núcleos convolucionales de esa capa mientras que la columna *Dimensión* muestra el tamaño de cada uno de estos filtros. Por tanto, en la primera capa convolucional, sobre una imagen de entrada I de dimensiones 512×512 se le aplican 32 núcleos convolucionales K de tamaño 3×3.

Continuando con la misma figura y con la descripción del primer conjunto (C1), después de haber aplicado los sucesivos núcleos de convolución se procede a realizar la operación de normalización a fin de acelerar la convergencia durante el proceso de aprendizaje.

$$b_{x,y}^{i} = a_{x,y}^{i} \Big/ \left(k + \alpha \sum_{j=\max\left(0,i-\frac{n}{2}\right)}^{\min\left(N-1,i+\frac{n}{2}\right)} \left(a_{x,y}^{i}\right)^{2}\right)^{\beta} \tag{2.7}$$

En la ecuación (2.7) N indica el número de filtros de una capa dada y $a_{x,y}^{i}$ representa la actividad de la neurona mientras que k, n, $\propto$ y β son hiperparámetros fijados en Krizhevsky (2012) a $k = 2$, $n = 5$, $\propto = 10^{-4}$ y $\beta = 0.75$.

La última operación que se realiza en este conjunto C1 es la aplicación de la función de activación sobre el mapa de características normalizado. En particular, el modelo YOLOv3 utiliza la función *Leaky ReLU* representada en la Figura 2.8.

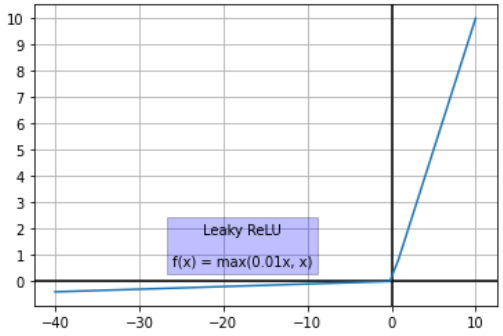


Figura 2.8 Operación Leaky ReLU

Al mapa de características B resultado de haber aplicado los sucesivos núcleos convolucionales, y posteriormente habiendo normalizado sus valores, se le aplica dicha función de activación. Puede verse en la imagen que los valores negativos de x con $x \in B$ quedarán divididos por un factor de 100, en contraposición con la función *ReLU* original que los iguala a 0, mientras que los positivos no serán alterados.

El segundo conjunto (C2), al que pertenece la segunda capa de convolución, añade una operación de *padding* ($p = 1$) sobre las matrices de salida resultado de la capa convolucional anterior donde se añade, a cada una de ellas, una fila y una columna de ceros en ambos extremos (*Zero Padding*). En ocasiones, se emplea el término *same* para indicar la existencia de esta operación previa a la aplicación de los sucesivos núcleos convolucionales, o se define un valor para p, y el término *valid* para informar de su ausencia siendo en este caso $p = 0$. Durante la convolución, se aplica un *stride* ($s = 2$) que consiste en desplazar el filtro a lo ancho y a lo alto de una matriz en s posiciones cada vez. En la tabla de la Figura 2.6 podemos ver la presencia de este desplazamiento en la columna *Size* con la nomenclatura 3x3/2 siendo 2 el valor que toma *stride*; por otra parte, en la imagen a la derecha de la tabla es empleada la nomenclatura $s = 2$ que pretende denotar el mismo significado.

Para ilustrar el comportamiento de una operación de convolución aplicando *padding* y *stride* se muestra el ejemplo de la Figura 2.9 con $p = 1$ y $s = 2$ siendo una matriz I de entrada de 7×7 y un filtro K de dimensión 3×3.

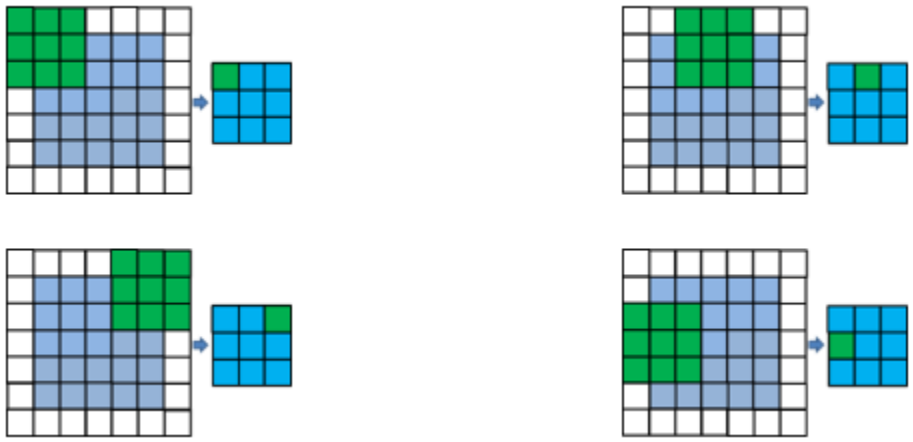


Figura 2.9 Zero-padding con p=1 y s = 2

Persiste en este conjunto C2 la capa de normalización y la función de activación *Leaky ReLU* presente del mismo modo que en el conjunto C1.

El tamaño del mapa de características de salida, reflejado en la imagen en la columna *Output*, resultado de la aplicación de sucesivos núcleos convolucionales viene dado por la aplicación, también sucesiva, de la siguiente ecuación donde i representa

la dimensión de una matriz de entrada I y k hace referencia a las dimensiones del filtro de convolución K mientras que p y s representan al valor de *pooling* y *stride* aplicado.

$$s = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1 \tag{2.8}$$

Como se ha mencionado previamente, en la última imagen donde $i = 5$, $k = 3$, $p = 1$ y $s = 2$ el resultado, como puede verse en esta, es una matriz de características S de tamaño sxs con $s = 3$. Este cálculo ha de realizarse de forma independiente para cada una de las dimensiones disponibles; dado que en este caso el *padding* es el mismo tanto en vertical como en horizontal y lo mismo sucede con el *stride* y con las dimensiones tanto de la matriz de entrada como del filtro, ha sido posible realizar la operación una única vez para obtener las dimensiones de S.

El tercer y último conjunto (R), al que pertenece la primera capa residual, está formado por la sucesión de dos conjuntos C1 donde la salida resultante de este segundo se concatena con la entrada del primer conjunto C1 de forma semejante a las unidades residuales presentes en el *backbone* ResNet, Figura 2.10. La característica principal de esta arquitectura a la que se hace breve mención, y lo que resultó ser un factor diferencial sobre lo que había en ese momento, son las *skip connections* que permiten propagar las características de una capa a otra que está por delante en la pila; esto permite que la red pueda comenzar a converger hacia la función objetivo con un menor número de iteraciones al propagar la señal de las capas inferiores en capas que no se han actualizado todavía.

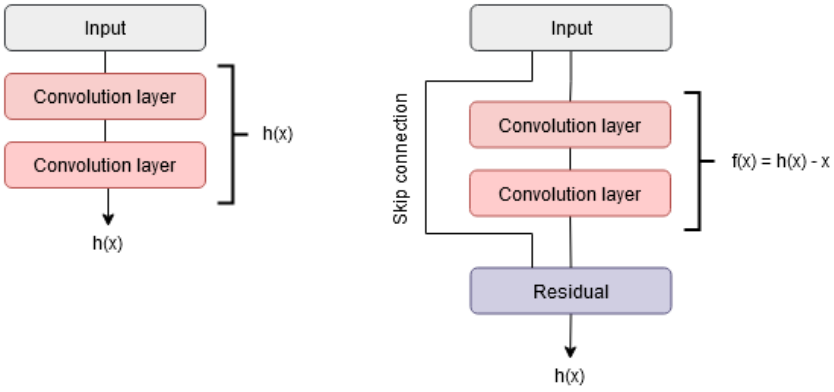


Figura 2.10 Residual units modelo ResNet

Se presenta en la Figura 2.11 la conexión entre las 2 primeras capas de núcleos convolucionales y la primera capa residual.

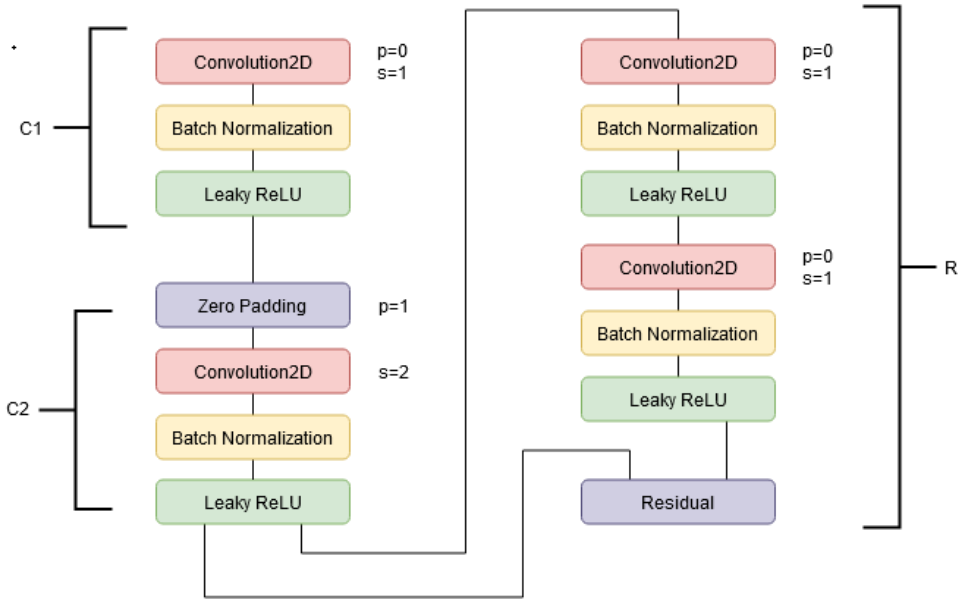


Figura 2.11 Primeras capas Darknet-53

El modelo YOLOv3 es capaz de predecir *bounding boxes*, la presencia de objetos contenidos en ellos y la probabilidad de clase dada la presencia de un objeto. El procedimiento que lleva a cabo este modelo a fin de proporcionar estas predicciones parte de dividir la imagen de entrada en una rejilla de dimensiones CxC, tal y como se representa en la imagen de la Figura 2.12.

Figura 2.12 Definición de rejilla para el modelo YOLOv3

Durante la fase de entrenamiento, si en una celda de la rejilla se sitúa el centro de un objeto esta es la responsable de su detección, proponiendo *anchor boxes* que traten de asemejarse al *ground truth*. En lugar de realizar una selección manual, en función también de variar la relación de aspecto de estos y/o aplicarles coeficientes de escala, YOLOv3 aplica un procedimiento basado en el agrupamiento tipo K-means. Se definen inicialmente K centroides y se propone que la distancia entre estos y los boxes se rija bajo el mejor valor del coeficiente *IoU* entre ambos.

A partir de los centroides obtenidos mediante la aplicación de este método se predice el ancho y el alto de los *anchor boxes,* en función del desplazamiento de estos sobre los *bounding boxes*. En la imagen de la Figura 2.13, c_x y c_y representan a las coordenadas de la esquina superior izquierda de la celda de la rejilla del *anchor box,* delimitado por b_x, b_y, b_w, b_h siendo p_w y p_h el ancho y el alto del *bounding box*.

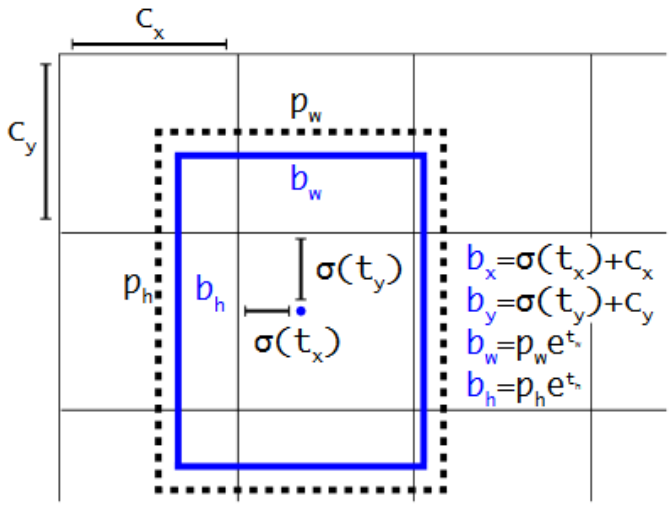


Figura 2.13 Definición del bounding box

En este caso, la función de coste de pérdida de localización se computa como la suma del error cuadrático medio entre el *ground truth* y los *bounding box* predichos por el modelo; siendo el gradiente $\hat{t}_* - t_*$ donde $\hat{t}_*$ representa las coordenadas del *ground truth* y t_* las predichas del *bounding box* que pueden ser despejadas estas de las siguientes ecuaciones.

$$b_x = \sigma(t_x) + c_x;\ b_y = \sigma(t_y) + c_y;\ b_w = p_w e^{t_w};\ b_y = p_y e^{t_y} \tag{2.9}$$

Con la capacidad de predecir *bounding boxes,* el modelo infiere la posibilidad de presencia, o inexistencia, de los objetos contenidos en ellos mediante regresión logística cuya variable relevante es el valor del *IoU* entre los *anchor box* iniciales frente a los *bounding boxes* predichos. El modelo aplica el método de supresión no máxima citado en el apartado anterior, estableciendo un valor de 0.5 como umbral mínimo para determinar si hay un objeto presente en esos *anchor boxes*.

En última instancia, en relación con las capacidades de predicción del modelo, YOLOv3 emplea clasificadores logísticos independientes lo que le permite realizar la clasificación de objetos mutuamente no exclusivos. Durante la fase de entrenamiento, se utiliza la entropía binaria cruzada L para cada etiqueta como función de coste para indicar la pérdida en la clasificación de un objeto donde, según la siguiente ecuación, y es la etiqueta de la clase binaria y p la probabilidad de cada clase.

$$L = -\sum y * log(p) \tag{2.10}$$

Haciendo inciso en la arquitectura mostrada en la Figura 2.6, la salida de la última capa residual de los 3 últimos conjuntos R está conectada además a sucesivas capas de convolución, formando estas en su conjunto la cabeza del modelo, permitiendo realizar las predicciones enumeradas a distintas escalas; esto proporciona alta robustez para detectar objetos de distintas dimensiones. La salida de estas capas es un tensor 3D de dimensiones N×N× [3*(4+1+C)] donde N×N es la dimensión de la escala (64×64, 32×32 o 16×16 coincidiendo con las dimensiones del mapa de características dado a la salida de sendas capas residuales) y 3 es el número de *bounding boxes* propuestos por celda. Por otro lado, el 4 hace referencia a las coordenadas de un *bounding box*, un campo adicional que almacena la posibilidad de existencia de un objeto en un determinado *bounding box* y C es el cardinal del conjunto de las clases de objetos contemplados por el modelo durante la fase de entrenamiento.

En el contexto de este trabajo $C = 2$, representando a la clase Dedo y a la clase Panel, los elementos a identificar. Se presentarán los detalles relacionados con la generación del *dataset* que ha permitido entrenar al modelo en el Capítulo 3 mientras que la descripción de los procedimientos llevados a cabo durante la fase de entrenamiento y la discusión de los resultados obtenidos tendrá lugar en el Capítulo 4.

2.1.2 Reconocimiento de caracteres

La técnica de reconocimiento de caracteres OCR (*Optical Character Recognition*) permite la conversión de textos, mecanografiados o manuscritos, a textos codificados que pueden ser interpretados por un computador.

Dado que el uso de esta técnica hace posible extraer texto contenido en imágenes, ha sido la opción elegida para tratar de identificar aquellos servicios a incorporar en el contexto de las Ciudades Inteligentes en base a sus paneles informativos asumiendo que todos ellos dispondrán de al menos uno en las proximidades de su establecimiento y que su identificación se realiza de forma inequívoca en lo que respecta a la asociación del panel con la imagen que se ofrece.

Desde la perspectiva de la visión por computador, el problema de reconocimiento de caracteres puede abordarse desde dos vertientes. En cualquiera de los casos, ambas parten de la digitalización de un documento o imagen a fin de obtener su representación binaria; a continuación, se procede a preprocesar esta información.

Esta fase de preprocesamiento se realiza con el propósito de maximizar el acierto durante la segmentación de los distintos caracteres que pueda contener una imagen digitalizada (Pajares y Cruz, 2007). Algunas Las operaciones que se aplican con carácter general son las siguientes, que se sintetizan en la Figura 2.14:

- Alineamiento: Puede darse el caso de que sea preciso rotar una imagen si contiene textos que no se encuentran debidamente alineados sobre los ejes horizontales y/o verticales.
- Conversión a escala de grises: Convierte una imagen en color a una representación de esta en escala de grises.
- Eliminación de ruido: Es bastante común la aplicación del filtrado tipo *despeckle* para eliminar el ruido presente en las imágenes, producido por manchas en las lentes de las cámaras por citar un ejemplo. El procedimiento consiste en calcular por cada píxel y sus vecinos la desviación estándar sobre los valores que contienen para determinar si el área es de alta o baja complejidad. Si la complejidad es menor que un umbral dado, entonces el área se suaviza aplicando un filtro.

Figura 2.14 Proceso de segmentación para identificación de caracteres (OCR)

Durante la fase de segmentación es preciso, en primera instancia, identificar las áreas de las imágenes con texto.

La metodología con un enfoque más clásico es aquella que emplea la técnica de la ventana deslizante, ilustrada en la Figura 2.15, que consiste en desplazar un rectángulo sobre toda la imagen a fin de detectar caracteres. Si no se encontrase ninguno con ese rectángulo es necesario modificar su relación de aspecto de forma iterativa hasta encontrar alguno. Podemos determinar, si aplicamos esta técnica, que una región contiene un carácter con un clasificador basado en redes neuronales o determinando la similitud entre los píxeles que componen el potencial carácter sobre los píxeles que forman cada uno de los caracteres y dígitos disponibles almacenados en alguna base de datos. Una vez reconocida una sucesión de caracteres próximos entre sí es posible simplificar el problema de la ventana deslizante pudiendo asumir que el resto tendrán un tamaño similar, que pertenecerán a la misma fuente y que se encontrarán debidamente alineados, tal y como se muestra en la figura 2.15.

Figura 2.15 Alineamiento de caracteres para su identificación

Desde el punto de vista del Aprendizaje Profundo, el problema de la detección de objetos puede ser aplicado en este contexto. Sirva de ejemplo la red RPnet (Wang y col., 2018), utilizada para la detección y segmentación de matrículas de vehículos de tracción mecánica. Su *backbone* está formado por sucesivas capas convolucionales a las que se denomina en su conjunto como módulo de detección. Este módulo de detección es el encargado de predecir *bounding boxes* que son susceptibles de contener, con una probabilidad alta, matrículas. Estas áreas son denominadas regiones de interés (ROI) y son transferidas al módulo de reconocimiento para la fase de clasificación.

Este trabajo hace uso del servicio *cloud* Rekognition (2020) de Amazon Web Services, capaz de predecir *bounding boxes* y clasificar los caracteres contenidos en ellos. En este caso, el proveedor no hace pública la información sobre las técnicas que emplea para enfrentarse a la problemática que se presenta en esta sección de reconocimiento de caracteres.

Para el desarrollo de este trabajo se asume que este servicio no tendrá una precisión del 100% en la clasificación. Por tanto, se utiliza la distancia de Levenshtein para definir el concepto de similitud entre dos cadenas de caracteres con el fin de maximizar la correspondencia de los textos contenidos en los paneles informativos procesados mediante el servicio Rekognition con la información proporcionada por los comercios/instituciones dados de alta en el sistema. La distancia de Levenshtein se define como el número de operaciones necesarias que ha de aplicarse a una cadena de caracteres para convertirse en otra; si la distancia es 0 estamos hablando de cadenas idénticas, por tanto, el objetivo es encontrar parejas de cadenas que minimicen esta métrica.

La ecuación (2.11) define esta operación, donde a y b son cadenas de caracteres e i y j representan a un carácter dentro de la cadena a y dentro de la cadena b, respectivamente.

$$lev_{a,b}(i,j)\begin{cases} max(i,j) & si\ \min(i,j)=0 \\ min\begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i\neq b_j)} \end{cases} & en\ otro\ caso \end{cases} \tag{2.11}$$

Para dos cadenas de caracteres dadas se computa una matriz $LD_{I,J}$ con $i \in I$ y $j \in J$ y el resultado de la distancia de Levenshtein queda en la posición $(I-1, J-1)$.

En el Capítulo 3 se detalla convenientemente la aportación del servicio Rekognition de Amazon Web Services a este trabajo además del empleo de la distancia de distancia de Levenshtein para determinar la similitud entre distintas cadenas de caracteres con el fin de identificar a los distintos comercios o instituciones.

2.2 Software de gestión de modelos basados en Inteligencia Artificial

La generación de modelos basados en Inteligencia Artificial queda definida como un proceso iterativo en el que distintas etapas forman parte de este. En cada una de ellas se toman decisiones que pueden resultar relevantes para poder generar modelos prometedores.

Estas decisiones van, desde llevar a cabo un determinado conjunto de procedimientos para escoger, o incluso alterar, los datos con los que se trabaja hasta seleccionar los hiperparámetros que se utilizarán para tratar de maximizar la precisión del modelo.

Es un procedimiento de ingeniería tener control sobre todo lo que transciende en estas etapas para saber identificar la trascendencia de cada decisión que se toma. Este trabajo propone la utilización de herramienta MLflow (Zaharia, M y col., 2018) para este fin. Puede consultarse en el Apéndice A la licencia asociada a este software.

MLflow se define como una plataforma *open source* de gestión del ciclo de vida de modelos basados en Inteligencia Artificial. En este contexto, el ciclo de vida queda definido como ese proceso iterativo, en el que se llevan a cabo ciertos procedimientos, del que se hacía mención previamente al comienzo de este apartado.

La contribución de MLflow a este trabajo resulta relevante dado que permite registrar los entrenamientos, es decir, será posible consultar las métricas resultado de los mismos y categorizar los modelos en función de los resultados obtenidos. Además, los procedimientos llevados a cabo para llegar a ellos, así como los entornos donde fueron generados, quedarán también registrados.

Los siguientes subapartados describen las funcionalidades principales de MLflow. En el Capítulo 3 y en el Capítulo 4 se ilustrará la aportación de este paquete software desde un punto de vista práctico y su interacción con los distintos subsistemas de la solución que este trabajo propone.

2.2.1 MLflow

La primera versión de MLflow data de junio de 2018 y desde entonces se actualiza de forma frecuente y continuada dando soporte cada vez a más funcionalidades.

Además de las actualizaciones frecuentes, es importante tener en consideración que esta herramienta cuenta con una extensa y detallada documentación y una comunidad muy activa que da soporte en los principales foros de consulta de la red. Característica muy importante de tener en cuenta cuando nos decantamos por hacer uso de herramientas *open source*.

Quedan distinguidas 3 funcionalidades principales en este *framework* en la que, cada una de ellas, tiene cabida en distintas etapas de la gestión del ciclo de vida de los modelos:

- MLflow Tracking. Hace posible el registro de todo lo que puede resultar relevante cada vez que se entrena un modelo, incluyendo la copia del modelo entrenado.
- MLflow Projects. Es la herramienta de configuración de entornos de ejecución para los entrenamientos.
- MLflow Registry. Permite el acceso a la información registrada de cada entrenamiento.

A continuación, se describirá el alcance de cada una de dichas funcionalidades.

2.2.1.1 MLflow Tracking

Tener en el control sobre los procedimientos que se llevan a cabo parte de registrar todo aquello que pueda ser transcendente. Esta herramienta ofrece al usuario la posibilidad de que pueda almacenar:

- Información acerca de los datos con los que se trabaja.

    - Distribución de los datos, procedencia, etc.
- Procedimientos y parámetros introducidos.
    - Preprocesamiento de los datos efectuados, número de iteraciones, tamaño de los lotes, etc.
- Entorno en el que se trabaja.
    - Versión del código que se ejecuta.
    - Dependencias del código fuente.
- Métricas relevantes resultantes de la fase de entrenamiento.
- Modelos obtenidos.
    - Versionado de modelos.
- Fecha de inicio y fin de un entrenamiento.

MLflow Tracking está escrito en Python y se distribuye como una librería a través de los repositorios oficiales de Python. La librería da soporte a poder utilizar un extenso número de *frameworks* de aprendizaje automático distintos. Cabe mencionar que, para algunos de ellos como Tensorflow, Keras o Spark ofrece extensiones adicionales para facilitar su integración. Gracias a ella podemos registrar cada uno de los artefactos citados con anterioridad.

El término experimento es utilizado por MLflow para agrupar un conjunto o secuencias de entrenamientos. La herramienta ofrece distintas alternativas para poner en ejecución un entrenamiento con la finalidad de permitir al usuario decidir el momento y en qué circunstancias estos se llevarán a cabo:

- Desde la propia librería de Python.
- Sobre una interfaz de línea de comandos.
- A través de una API REST.
- Mediante una API de Java.

2.2.1.2 MLflow Projects

MLflow Projects permite al usuario poder empaquetar sus desarrollos de forma que sean reproductibles en otros sistemas.

El requisito de MLflow para llevar a cabo una ejecución es el de definir el entorno donde el código va a ser ejecutado. Ha de proporcionarse la siguiente información:

- Entorno de ejecución.
    - Conda o Docker. Dependiendo del caso es preciso proporcionar información adicional que permita poder ejecutar el código fuente.
- Parámetros de entrada admitidos a través de la interfaz.
    - Resulta útil si las ejecuciones se llevan a cabo mediante interfaces programables de forma automatizada.
- Comando que permite ejecutar el código fuente.

En la sección MLflow Tracking se hace mención acerca de que esta información también queda registrada para su posible consulta.

2.2.1.3 MLflow Registry

La herramienta pone a disposición una interfaz web que permite visualizar toda la información previamente registrada de cada una de las ejecuciones llevadas a cabo. Además de visualizar los resultados del entrenamiento de un modelo en tablas, admite la generación de gráficos con la finalidad de comparar distintas ejecuciones de un mismo experimento, entre otras opciones.

Esta interfaz permite al usuario clasificar los modelos según criterio experto de tal modo que puedan ser diferenciados en el momento que vayan a ser consumidos.

El registro de la información es persistente. Para hacerlo posible es necesario habilitar una base de datos relacional, cualquiera compatible con la librería SQLAlchemy (2020) de Python (recomendando PostgreSQL (2020)) y espacio de almacenamiento local o remoto soportando desde escrituras en disco local, disco remoto a través de varios protocolos (FTP, NFS) hasta espacio en servicios de contenedores de plataformas *cloud*.

En la base de datos relacional se almacenará todo lo relacionado con MLflow y con las ejecuciones que se llevan a cabo salvo los modelos que quedarán almacenados en el espacio de almacenamiento seleccionado.

2.2.2 Alternativas a MLflow

A la hora de seleccionar esta herramienta resultó interesante el hallazgo de dos alternativas con características singulares por lo que cabe mencionarlas:

- neptune.ai: Permite la gestión de usuarios que acceden al *framework* y comparar código fuente.
- Kubeflow: Ofrece de manera nativa los servicios de Jupyter además de permitir realizar el registro de los entrenamientos en entornos *cloud* sobre Kubernetes.

Estas opciones han de tenerse en cuenta si el contexto en el que se desarrolla un proyecto ha de tener en consideración estas características, de las que MLflow carece hoy en día.

2.3 Componentes software requeridos por la aplicación

Después de hacer inciso en la tecnología que permite gestionar el ciclo de vida del modelo, así como de los fundamentos teóricos que permiten generarlo, se hace lo propio en esta sección con las tecnologías que hacen posible explotarlo en un entorno fácilmente reproducible.

Paso previo a la elección de componentes que formarán parte del *backend* de la solución, del Módulo Propietario y del Módulo Consumidor, ha sido necesario analizar los posibles escenarios en los que la información de los servicios pueda ser proporcionada a la solución que propone este trabajo y en los cuales el modelo puede ser consumido a fin de recuperarla.

El *framework* Node-RED (2020), clasificado dentro de las plataformas que ofrecen *'low-code programming',* permite la creación de aplicaciones web basadas en la recepción de eventos. En esta plataforma pueden encontrarse *assets* que permiten publicar *bots* que habilitan a los usuarios de redes sociales y de clientes de mensajería

canales de entrada al Módulo Propietario. Del mismo modo, existen también implementaciones que permiten habilitar la comunicación con este módulo mediante protocolos de comunicaciones frecuentemente utilizados en IoT, en este caso HTTP y MQTT, este último soportado por Eclipse Mosquitto (2020). Adicionalmente, una base de datos no relacional MongoDB (2020) es provisionada en este Módulo Propietario a fin de almacenar toda la información proporcionada.

En lo relativo al Módulo Consumidor, se ha tenido en consideración habilitar una interfaz web basada en API REST que permita el acceso a MLflow Registry con el fin de poder recuperar la versión del modelo candidata. Dado que MLflow se distribuye como una librería de Python y que el *framework* contemplado para hacer inferencia sobre el modelo también se integra del mismo modo, Python ha sido el lenguaje seleccionado para llevar a cabo el desarrollo del proceso principal de este módulo. Este lenguaje también asegura poder consumir el servicio de Amazon Web Services, Rekognition vía API de programación y la recuperación de los documentos almacenados en MongoDB a través de un cliente. Sanic (2020) es el *web framework* de Python escogido para ofrecer el servicio web que atenderá a las solicitudes de información turística de los usuarios, previamente almacenada. Este es un *framework* extendido, soportado por la comunidad y con una documentación extensa y precisa.

Se ha tenido también en cuenta que la demanda de solicitudes a este servicio web puede ser variable por lo que conviene tener la posibilidad de poder ejecutar más procesos que permitan consumirlo. Las tecnologías basadas en contenedores se encuentran en auge desde hace más de un lustro; para cumplir el requisito de poder escalar la capacidad de cómputo con el fin de maximizar la disponibilidad del servicio y de automatizar el despliegue de todos los procesos, tanto los que corresponden al Módulo Propietario como a aquellos del Módulo Consumidor, es una tecnología idónea.

Pueden ser consultadas en el Apéndice A todas las licencias del software requerido para el desarrollo de este trabajo. El siguiente apartado versa sobre la tecnología de contenedores empleada para empaquetar todos los procesos de cada módulo, Docker. Se enumerarán sus características principales y su aportación en este trabajo. La comunicación entre todos los procesos, así como funcionamiento interno de cada uno de ellos, se encontrará presente en el Capítulo 3.

2.3.1 Tecnología de contenedores Docker

Desde el primer instante se planteó el uso de Docker (2020) con la finalidad de que todos los componentes, y la conectividad requerida que debe haber entre ellos, sea reproducible en distintos sistemas con bajo esfuerzo.

Si bien es cierto que es posible ejecutar distintos procesos en un mismo contenedor, pero no es lo recomendable. De tal forma, cada proceso se ejecutará en un contenedor independiente del resto. Con el fin de poner en funcionamiento en un entorno de pruebas a los distintos contenedores es necesario definir:

- El entorno de ejecución de cada proceso, la definición de imágenes Docker.
- Un entorno de red común donde se ejecutarán todos los procesos.
- El punto de montaje de sistemas de ficheros compartidos.

Un contenedor es una instancia en ejecución de una imagen Docker. En esencia, una imagen Docker es un fichero que almacena:

- Un sistema operativo.
- Los paquetes software instalados en él.
- Ficheros de configuración.
- Variables de entorno.
- Código fuente y/o ejecutables.

Docker Hub es el principal punto de distribución de imágenes Docker; certificadas por Docker o contribuidas por la comunidad. La principal premisa es que existe la posibilidad de generar nuevas imágenes a partir de las existentes donde ya se encuentra un sistema operativo y determinados paquetes software que podrá ejecutar.

La herramienta de línea de comandos de Docker es capaz de interpretar ficheros de configuración donde se especifica, mediante el uso de reglas, cuál es la imagen de base y las operaciones que se realizará sobre el sistema operativo existente para construir una nueva imagen. Las etiquetas más importantes son:

- FROM: Especifica cuál es la imagen de base de la nueva imagen.

- RUN: Permite ejecutar cualquier operación soportada por el sistema operativo disponible en la imagen.
- CP: Hace posible incluir en la imagen ficheros alojados en el sistema de archivos de la máquina anfitrión. Suele utilizarse principalmente para copiar el código fuente o los ejecutables requeridos.
- ENV: Variables de entorno.
- CMD: Comando que ejecutará el contenedor una vez inicializado.

Existen guías de buenas prácticas que es recomendable seguir con el fin de que las imágenes ocupen el menor espacio posible en disco.

Las imágenes están formadas por capas. Cada regla ejecutada genera una nueva capa en la imagen de tal modo que la generación de una imagen a partir de otra radica en añadir más capas a la imagen de base.

Es también la herramienta de línea de comandos la que nos permite ejecutar distintas instancias, en forma de contenedores, de las imágenes construidas. Con ella podremos definir además el entorno de red y el punto de montaje de sistemas de ficheros compartidos. No obstante, existe una herramienta de abstracción de la interfaz de línea de comandos, docker-compose, que facilita en gran medida todo lo anterior gracias también a la interpretación de ficheros de configuración que siguen determinadas reglas.

Atendiendo a estas directrices, ha sido posible poner en funcionamiento todos los procesos requeridos en el contexto de este trabajo al completo en entorno local con el fin de hacer pruebas y, posteriormente, en un entorno *cloud*.

2.4 Servicios cloud

Este trabajo pretende hacer uso de los distintos servicios proporcionados por las principales plataformas *cloud* para poner a disposición de cualquier usuario la posibilidad de que pueda consumir la solución que aquí se propone.

En primer lugar, ha sido necesario identificar los requerimientos de cada módulo, que se reflejan en la Tabla 2.1:

Componente/ Requisito	Datalab	Entorno de ejecución	Base de datos	Espacio de almacenamiento	Registro de contenedores	OCR
Módulo IA	X		X	X		
Módulo Propietario		X			X	
Módulo Consumidor		X			X	X

Tabla 2.1. Componentes requeridos por la solución propuesta

A continuación, se realiza un análisis de las posibilidades que ofrecen las principales plataformas *cloud* (Microsoft Azure, Google Cloud Platform y Amazon Web Services) de cara a satisfacer los objetivos impuestos en este trabajo haciendo una estimación del precio que supone hacer uso de ellos. Será en el Capítulo 3 donde se ilustrará la comunicación entre los distintos servicios aquí descritos.

2.4.1 Requerimientos. Módulo IA.

Se ha tenido en consideración que este trabajo precisa hacer uso de hardware que incluya al menos una GPU para acelerar todo aquel entrenamiento del modelo, además de espacio de almacenamiento de ficheros donde alojar los *dataset*, y conectividad a internet para comunicarse con MLflow Registry a través de MLflow Tracking con el fin de salvar los registros de entrenamiento y almacenar los modelos entrenados.

Google Colab Pro (Bisong, 2019) ofrece una suscripción mensual por usuario de $9,99. Esta te permite acceder a las GPUs de NVIDIA T4 y P100 que son totalmente aptas para el entrenamiento de modelos basados en *Deep Learning*.

Se valora positivamente poder acceder a través de este servicio a *Notebooks* con las dependencias software que tiene *TensorFlow*, el *framework* de *Deep Learning* utilizado en este trabajo, ya satisfechas (además de permitir la instalación de librerías adicionales) y el pago de una cuota fija independientemente del uso. A pesar de que no garantizan la disponibilidad 100% del hardware no es un impedimento su empleo en el contexto de este trabajo.

Las alternativas por parte de Amazon Web Services (2020) y de Microsoft Azure son Amazon SageMaker y Azure Machine Learning pero con un tipo de suscripción de

pago por uso. Si es cierto que garantizan la disponibilidad completa del hardware, pero las opciones disponibles ofrecen recursos computacionalmente muy superiores a los necesarios; para el propósito de este trabajo sería hacer uso de una plataforma sobredimensionada y, por tanto, con un coste injustificado. Lo mismo ocurre con Google y su servicio Google Datalab.

De este modo, Google Colab será el entorno donde los modelos serán diseñados y entrenados y se contratarán 100 gb de almacenamiento en Google Drive a coste de $1,99 mensuales para almacenar los *dataset*.

Por otra parte, MLflow Registry requiere de una base de datos PostgreSQL donde almacenar los registros de entrenamiento y de espacio de almacenamiento de objetos para guardar los modelos entrenados.

Es turno entonces de evaluar la oferta de las 3 plataformas *cloud*, donde todas ofrecen bases de datos PostgreSQL, en términos de bases de datos relacionales:

Plataforma	**Nombre BBDD**	**vCPU**	**RAM**	**Precio/hora**	**Precio/GB**
Microsoft Azure	BASIC Gen 5	1	2	$0,037	$0,11
Amazon Web Services	db.t3.small	1	2	$0,038	$0,133
Google Cloud Platform	db-g1-small	1	1.7	$0,0350	$0,17

Tabla 2.2. Coste de bases de datos relacionales

Estos precios representan un punto de partida asumible para poder satisfacer el requisito técnico que impone MLflow Registry. Si el número de usuarios de MLflow aumentase, se ha valorado que no sería necesario aumentar la capacidad de cómputo puesto que las operaciones sobre la base de datos no son complejas en ninguna de las dos direcciones (ni escrituras ni lecturas) y que el volumen de solicitudes sobre esta es bajo. Sí resulta relevante que, con el tiempo, sería necesario aumentar el espacio requerido de almacenamiento; aunque no demasiado ya que cada registro de entrenamiento de MLflow Tracking se encuentra dentro de la magnitud del byte.

Considerando que esta base de datos ha de estar disponible en todo momento para que el servicio web pueda consultar en todo momento cuál es la versión del

modelo productivo se realiza el siguiente cálculo ((30 días al mes * 24 horas/día)*(Precio hora)) + ((5gb almacenamiento)*(Precio gb)):

- Azure: $27,19
- AWS: $28,025
- GCP: $25,285

Con el fin de que MLflow pueda almacenar los modelos entrenados se hace una evaluación sobre el coste del almacenamiento de objetos en las distintas plataformas.

Plataforma	Nombre Almacenamiento	Precio GB
Microsoft Azure	File Storage	$0,022
Amazon Web Services	S3	$0,024
Google Cloud Platform	Cloud Storage	$0,026

Tabla 2.3. Coste de espacio de almacenamiento

Los precios no varían demasiado independientemente de la plataforma *cloud*. Será necesario provisionar al menos 20 gb para almacenar los modelos por lo que el coste estimado será de $0,5.

2.4.2 Requerimientos. Módulo propietario.

Para albergar los distintos servicios que componen este módulo (MLflow Registry, Node-RED y MongoDB) una máquina virtual Linux con conectividad a internet, con 2 vCPU (Procesador Virtual), 4 gb de RAM y 30 gb de espacio de almacenamiento en formato HDD es suficiente.

El precio del servicio que ofrece cada plataforma atendiendo a estas especificaciones se refleja en la tabla 4:

Plataforma	Nombre MV	vCPU	RAM	Precio/hora	Precio/GB
Microsoft Azure	B2S	2	4	$0,0351	$0,11
Amazon Web Services	t3a.medium	2	4	$0.0326	$0,133
Google Cloud Platform	e2-medium	2	4	$0,03351	$0,17

Tabla 2.4. Coste del servicio de máquinas virtuales

Del mismo modo que en el caso de la base de datos, este servicio debe estar disponible en todo momento por lo que aplica también utilizar la siguiente fórmula ((30 días al mes * 24 horas/día)*(Precio hora)) + ((30gb almacenamiento)*(Precio gb))

- Azure: $28,572
- AWS: $27,461
- GCP: $29,227

Por otra parte, es necesario contratar un servicio para almacenar las imágenes generadas de los contenedores de Docker ejecutarán los procesos de la solución.

Mientras que Microsoft Azure tiene una cuota diaria de $0,16 por 10 GB, GCP y AWS ofrecen servicio de registro de contenedores basado en el paso por uso. Cada GB tiene un coste de $0,1 y $0,02 respectivamente.

Se tiene en consideración almacenar alrededor de 5 GB de datos por lo que el coste no ascenderá el precio de $0,20 en ningún caso.

2.4.3 Requerimientos. Módulo consumidor.

Con el fin de proporcionar a los usuarios acceso al Módulo Consumidor se tomó la decisión de que el proceso principal, el *web server* basado en Sanic, se ejecutase en entornos *severless*. Esta arquitectura está basada en un modelo de ejecución en el que el proveedor *cloud* es responsable de ejecutar código fuente dentro de contenedores mediante la asignación dinámica de los recursos.

En este contexto, la facturación se realiza en base a los recursos utilizados para ejecutar estos contenedores. Se comenzó provisionando 4 contenedores que ejecutarán el *web server* donde se reservó 1 vCPU y 2 gb de RAM a cada uno.

Plataforma	**Nombre servicio**	**Precio vCPU/hora**	**Precio GB/hora**
Microsoft Azure	Container Instances	$0,04860	$0,00533
Amazon Web Services	Elastic Container Service	$0,01279354	$0,00140482

Tabla 2.5. Coste de recursos entornos serverless

En este caso, Google Cloud Platform únicamente ofrece este servicio a través de Google Kubernetes Engine. El uso de este servicio requiere consideraciones adicionales de carácter técnico no contempladas en el alcance de este trabajo por lo que su uso queda descartado.

La diferencia de coste del precio/hora de ambos componentes que reside entre la plataforma de Microsoft Azure y la de Amazon Web Services se debe a que esta segunda ofrece recursos, conocidos como recursos *on spot*, capaces de ejecutar contenedores susceptibles a ser interrumpidos. En la documentación de este servicio queda patente que existen implementados mecanismos que permiten restaurar de forma automática los contenedores que puedan verse afectados. Dado que en el contexto de este trabajo la pérdida temporal de un contenedor del servicio que proporciona el Módulo Consumidor no supone la caída del sistema, resulta relevante su elección dado su coste.

Atendiendo a esta fórmula, ((30 días al mes * 24 horas/día)*(Precio hora vCPU * 1vCPU)) + ((30 días al mes * 24 horas/día)*(Precio hora GB * 2 GB)), el precio estimado de este servicio en Amazon Web Services es de $11,24 mientras que en Microsoft Azure es de $42,68 por contenedor.

Este módulo precisa, de forma adicional, hacer uso de un servicio de OCR. En la siguiente tabla se exponen los precios que ofrecen los distintos proveedores.

Plataforma	**Nombre OCR**	**Cantidad**	**Precio**
Microsoft Azure	Computer Vision API	1000	$1,50
Amazon Web Services	Rekognition API	1000	$1,00
Google Cloud Platform	Cloud Vision API	1000	$1,50

Tabla 2.6. Coste de servicios OCR

Los tres proveedores *cloud* facturan por lotes de 1000 peticiones de OCR al precio indicado siendo Amazon Web Services la opción más económica.

Capítulo 3 - Diseño de la propuesta

Una vez realizada la revisión sobre marco teórico en el que se encuentra este trabajo y de las tecnologías que permiten alcanzar los objetivos propuestos, es turno de definir la aportación en particular de cada uno de los recursos previamente descritos. En este capítulo se sigue el diseño de la Figura 1.2 con sus tres módulos (IA, Propietario, Consumidor) correspondientes y sobre los que se incide a continuación.

El apartado 3.1 versa sobre los procedimientos de Inteligencia Artificial llevados a cabo que han permitido la generación de *datasets* con los cuales el modelo de detección de objetos propuesto ha sido entrenado.

A continuación, en el apartado 3.2, se detalla la estructura y el comportamiento de la aplicación web basada en el *framework* Sanic que consume el modelo de detección de objetos, así como el servicio de reconocimiento de caracteres Rekognition ofrecido por Amazon Web Services. De forma adicional, se hace lo propio para la aplicación basada en el desarrollo de flujos de Node-RED encargada de almacenar en la base de datos MongoDB la distinta información publicada en los canales disponibles que será consumida por esta primera.

Por último, el apartado 3.3 enumera los servicios *cloud* desplegados que permiten el correcto funcionamiento de la solución, así como la comunicación entre todos ellos.

3.1 Módulo IA. Generación y etiquetado de *dataset*

Para el desarrollo de este trabajo se ha generado un *dataset ad-hoc* con imágenes de dedos y de paneles informativos. A riesgo de encontrar un *dataset* público que contenía imágenes debidamente etiquetadas de manos (Alam y col., 2019) apto para el entrenamiento de un modelo de detección de objetos no pudo obtenerse otro de las mismas características con instantáneas de paneles informativos de comercios o instituciones.

Este *dataset* ha sido creado a partir de la extracción y el etiquetado de fotogramas de vídeos realizados a pie de calle con un teléfono móvil convencional. Durante la grabación de vídeos se tuvo en consideración que una imagen podría

contener uno o varios paneles del mismo modo que los dedos podían ser de la mano derecha o de la mano izquierda. Adicionalmente, se tuvo en cuenta que los paneles pueden estar encendidos o apagados y que esto podría suceder tanto de día como de noche.

Para realizar la extracción de fotogramas y el etiquetado se ha hecho uso de la herramienta *open source* de anotación de vídeos VoTT (Visual Object Tracking, 2020), cuya licencia puede consultarse en el Apéndice A. En la imagen de la Figura 3.1 puede verse una captura realizada por dicha herramienta donde se presenta todo aquello que resultó relevante del proceso de etiquetado llevado a cabo.

Figura 3.1 Ejemplo de panel informativo y de dedo índice señalándolo

Después de crear un proyecto, la plataforma invita a incluir vídeos al mismo. Durante la reproducción de un vídeo, en el panel inferior hay dispuestos botones multimedia para reproducir/parar o rebobinar/adelantar el vídeo seleccionado a fin de fijar un determinado fotograma. Una vez fijado, es posible establecer *bounding boxes* con el ratón y asignarles una determinada categoría. En particular, la imagen ilustra un determinado fotograma extraído de un vídeo con dos *bounding boxes*, cada uno con la clase del objeto al que representa.

Al completar la tarea de etiquetado, la herramienta exporta todos los fotogramas seleccionados a un directorio y recopila en otro fichero una tabla que hace referencia a cada uno de los ficheros junto con la posición de las 4 coordenadas del *bounding box*

y de la clase del objeto contenido; en caso de que un fotograma contenga varios *bounding boxes* escribirá una fila por cada uno de ellos haciendo referencia al mismo fichero. Sirva de ejemplo el resumen mostrado en la imagen de la Figura 3.2 para ilustrar la tabla resultante.

	image	xmin	ymin	xmax	ymax	label
0	VID_20201004_184531.mp4#t=2.jpg	1126.293996	532.753067	1404.554865	902.470859	Dedo
1	VID_20201004_184531.mp4#t=2.jpg	602.898551	388.311350	1020.289855	465.170245	Panel
2	VID_20201004_184531.mp4#t=2.jpg	91.428571	397.587423	462.443064	474.446319	Panel
3	VID_20201004_184531.mp4#t=0.4.jpg	94.078675	406.863497	449.192547	469.145706	Panel
4	VID_20201004_184531.mp4#t=0.4.jpg	592.298137	390.961656	1022.939959	477.096626	Panel
5	VID_20201004_184531.mp4#t=0.416667.jpg	95.403727	398.912577	447.867495	471.796012	Panel
6	VID_20201004_184531.mp4#t=0.416667.jpg	590.973085	397.587423	1025.590062	474.446319	Panel
7	VID_20201004_184531.mp4#t=1.983333.jpg	1138.219462	535.403374	1392.629400	873.317485	Dedo
8	VID_20201004_184531.mp4#t=1.983333.jpg	604.223602	392.286810	1021.614907	466.495399	Panel
9	VID_20201004_184531.mp4#t=1.983333.jpg	92.753623	393.611963	465.093168	469.145706	Panel

Figura 3.2 Coordenadas de los bounding boxes junto con su etiqueta identificativa

Una vez generado el *dataset*, cuyo enlace de descarga puede encontrarse en el Apéndice B, fue turno de entrenar el modelo YOLOv3 propuesto en este trabajo. En el Capítulo 4 se describe la metodología que ha permitido el entrenamiento del modelo de detección de objetos contando con el soporte de MLflow, así como un análisis de los resultados obtenidos en la fase de *test*.

3.2 Diseño y comportamiento de la solución propuesta

En este apartado se describen tanto los aspectos de diseño como los funcionales de cada uno de los componentes que intervienen en la solución. Se ha dividido en dos secciones a fin de ilustrar, por un lado, los componentes y los procedimientos necesarios para que los propietarios de los comercios/instituciones interesados puedan registrarse y publicar contenido mientras que se hace lo propio con la aplicación web que, mediante el uso de los modelos de inteligencia artificial y de procesamiento de imágenes, consumen esta misma información de cara a que los potenciales consumidores de los servicios ofertados puedan verse atraídos por las distintas promociones que facilitan los primeros.

3.2.1 Módulo Propietario

El propósito de este módulo es el de ofrecer a los distintos proveedores de los servicios que se ofertan en las *Smart Cities* un servicio que les permita publicar información que resulte de interés eximiéndoles en algunos casos de conocimientos profundos sobre las tecnologías de la información y de costes directos.

El desarrollo de este módulo se ha realizado sobre Node-RED. Esta herramienta ha ofrecido durante la fase de programación un alto nivel de abstracción a la hora de implementar las distintas capacidades que aquí se describen. En particular, se han habilitado cuatro flujos, uno para el cliente de mensajería Telegram, otro para la red social Twitter (ambos servicios gratuitos para el usuario y de distintas características) y otros dos que integran protocolos de comunicaciones que se encuentran presentes en el ámbito de IoT, concretamente HTTP y MQTT. Para el primero ha sido necesario crear un *bot* siguiendo las instrucciones que ofrece Telegram y para el segundo una cuenta de desarrollador con la que poder consumir los servicios que ofrece Twitter a fin de recuperar un determinado conjunto de *tweets;* estos requisitos no han incurrido ningún tipo de coste. Para los dos últimos, el único requisito ha sido publicar ambos servicios en un puerto TCP disponible; el primero se publica de forma nativa a través de Node-RED y el segundo a través de software Eclipse Mosquitto. Cada uno de los canales se ha habilitado con un propósito específico, salvo en el caso de Telegram y HTTP, y tiene una persistencia en base de datos distinta en función del tipo de información que es transmitida, definido por el canal de procedencia. En la siguiente tabla queda ilustrada esta categorización de información en función de los canales de entrada y su persistencia en base de datos.

Canal	Propósito	Persistencia en BBDD
Telegram	Información general	Infinita
HTTP	Información general	Infinita
Twitter	Opiniones de usuarios	7 días
MQTT	Información de sensores	La última muestra

Tabla 3.1. Publicación de canales del Módulo Propietario

El motivo de la coexistencia entre el canal de Telegram y el de HTTP es que el primero abre una vía para que cualquier servicio tenga capacidad de publicar información sin que este ejercicio pueda incurrir algún tipo de coste directo para él y el segundo se proporciona a fin de que la publicación de información pueda ser automatizada.

En la Figura 3.3 se muestra el flujo habilitado para Telegram así como las 9 operaciones que ofrece a los usuarios, debidamente etiquetadas.

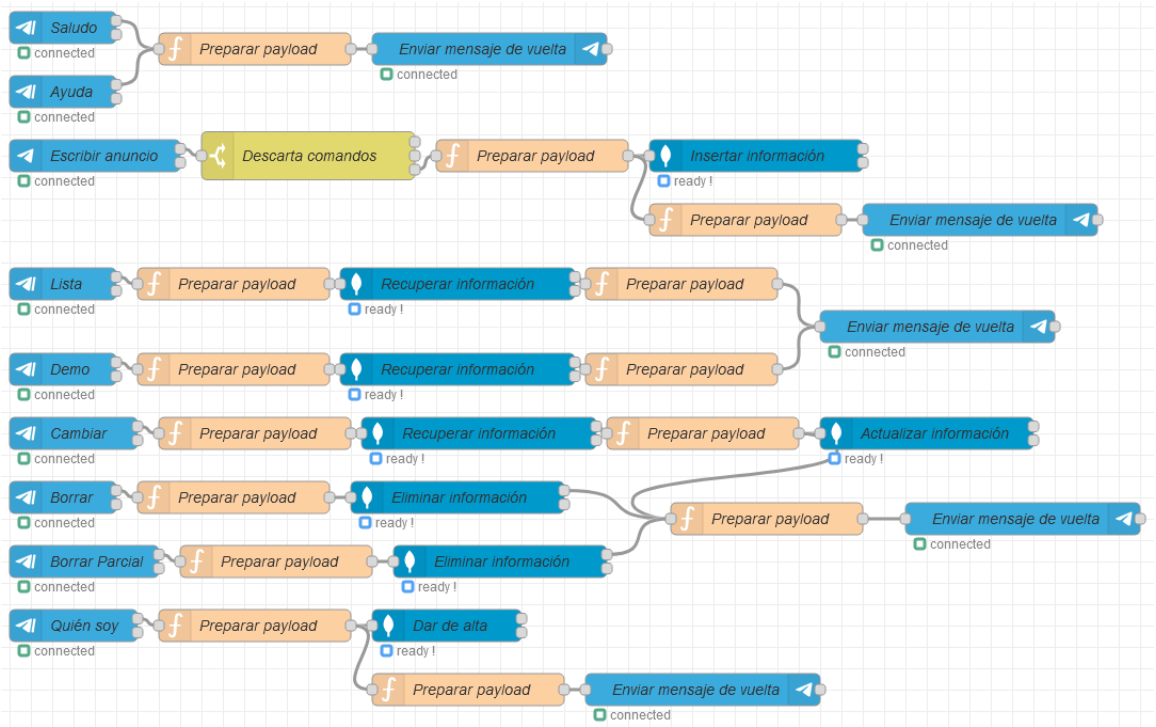


Figura 3.3 Flujo de Telegram

Iniciar una conversación con el *bot* de Telegram dispara el flujo de 'Saludo' y tiene el mismo efecto que el de solicitar ayuda con el comando 'Ayuda'. Resulta apropiado que cada vez que el usuario vaya a interactuar con el *bot* este reciba las indicaciones necesarias que le permitan hacer uso de las capacidades de servicio; además de permitirle acceder a ellas siempre que lo solicite.

Las demás operaciones hacen referencia a la posibilidad de escritura de anuncios, a listar estos, a deshabilitarlos y a la eliminación selectiva de los mismos. Por hacer inciso, es posible deshabilitar los anuncios publicados con la operación 'Cambiar' para que dejen de estar disponibles para los consumidores por lo que se han habilitado

dos modos de listado: 'Lista', que ofrece todos los servicios publicados, deshabilitados o no, y 'Demo' que solo mostrará las publicaciones habilitadas. Las operaciones 'Borrar' y 'Borrar Parcial' hacen lo propio con la operación de borrado; mientras la primera borra la totalidad de las publicaciones, la segunda solo elimina aquellas deshabilitadas.

Todas las operaciones que se han enumerado requieren un pre-procesado del *payload* que llega junto con el mensaje de los usuarios para transformarlo en un documento, con una determinada estructura, como paso previo a una operación sobre la base de datos MongoDB.

El comando que habilitaría a un proveedor poder estar presente para los consumidores sería 'Quién soy' que establece, y almacena en esta base de datos, la relación entre el identificador de su usuario de Telegram con el nombre que hace referencia a su servicio; presumiblemente el nombre presente en el panel informativo de su local a pie de calle. Es indispensable utilizar este comando a fin de obtener un identificador que permita hacer uso del resto de canales.

El flujo habilitado para el protocolo HTTP, mostrado en la Figura 3.4, es muy parecido al anterior.

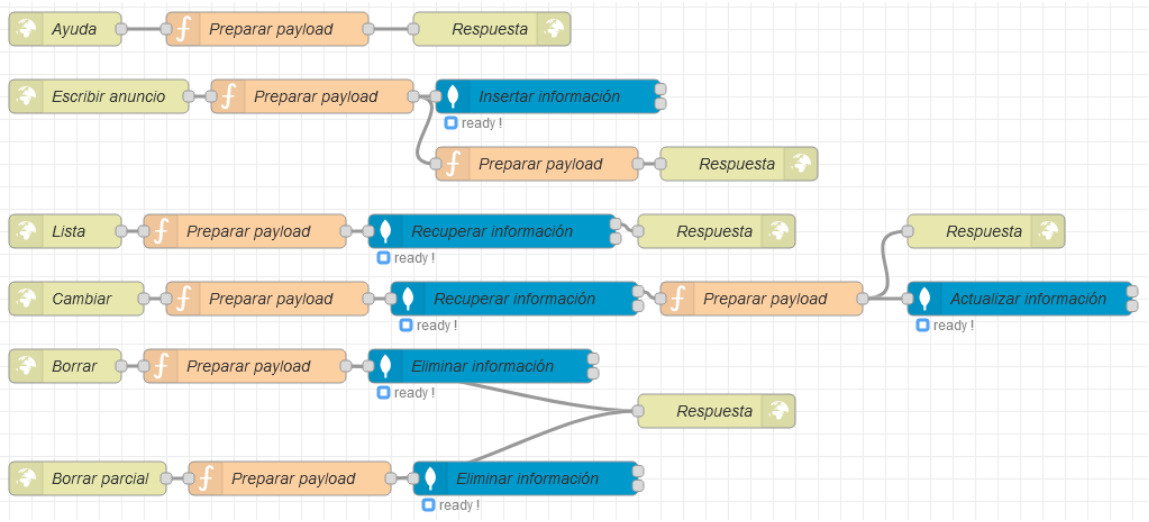


Figura 3.4 Flujo del protocolo HTTP

Se habilitan 6 de las 9 operaciones presentes en el canal Telegram (Ayuda, Escribir anuncio, Lista, Cambiar, Borrar y Borrar Pacial) comportándose del mismo modo que en caso anterior. El comando 'Saludo' se suprime dado que en el contexto HTTP no hay

necesidad de anunciar al servidor que se procede a establecer comunicación con él mientras que la supresión del comando 'Quién soy', presente en del flujo de Telegram, queda fuera de este flujo por motivos de diseño del propio módulo, a fin de unificar las solicitudes de alta de usuarios; cada petición HTTP deberá adjuntar el identificador de usuario del servicio.

El flujo de Twitter, presente en la Figura 3.5, resulta más simple que los anteriores dado que se delegan las operaciones de escritura, modificación y eliminación de *tweets* sobre el cliente de Twitter.

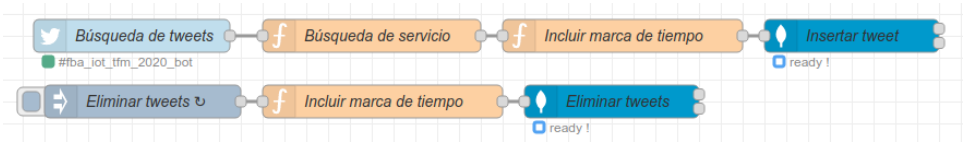


Figura 3.5 Flujo de Twitter

El nodo 'Búsqueda de tweets' se encarga de descargar *tweets* cada 15 minutos sobre la API de Twitter con las credenciales de desarrollador. El *payload* que recibe el flujo es tratado con el propósito de incluir una marca de tiempo y de identificar al servicio al que hace referencia; esta información se almacena en la base de datos MongoDB junto con el *tweet* publicado. El nombre del servicio debe coincidir con el que podrá leerse observando el panel informativo del comercio/institución. La marca de tiempo resulta útil ya que la persistencia en base de datos para estos documentos tiene una duración de 7 días dado que la naturaleza de este canal reside en ofrecer información de actualidad; el nodo 'Eliminación de tweets' es una tarea periódica que se ejecuta a las 06:00 AM en el huso GMT+1 los miércoles para hacer este borrado selectivo.

En última instancia, el flujo de MQTT, ilustrado en la Figura 3.6, habilita que el *broker* dispuesto con Eclipse Mosquitto quede subscrito a un único *topic* donde podrá recibir la publicación de datos de sensores a fin de almacenarlos en la base de datos MongoDB junto con una marca de tiempo.

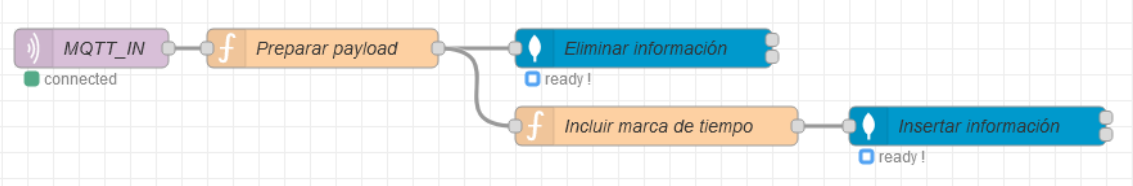


Figura 3.6 Flujo de MQTT

Como se ha mencionado previamente, la inserción de información procedente de un sensor provocará la sobreescritura del dato publicado con anterioridad en base a un identificador de sensor y al identificador del servicio que lo está publicando, proporcionado por el flujo de Telegram.

Los recursos que permiten el despliegue del Módulo Propietario pueden encontrarse en el Apéndice B mientras que la guía de usuario de los distintos canales puede verse en el Apéndice D. Por cualquiera de los cuatro medios es posible publicar la información que los usuarios del Módulo Consumidor podrán obtener.

3.2.2 Módulo Consumidor

Si bien el Módulo Propietario hace la función de productor de información, el Módulo Consumidor cumple el papel de consumidor de esta dentro de la solución.

El desarrollo de esta aplicación se ha realizado en Python utilizando el web framework Sanic además de hacer uso de las librerías necesarias para consumir el modelo de detección de objetos entrenado basado en YOLOv3, así como del cliente de Amazon Web Services que permite consumir, vía API de programación, sus servicios, entre otras. Los recursos habilitados para desplegar este Módulo se encuentran enumerados en el Apéndice B. La especificación de cómo consumir este servicio puede verse en el Apéndice D.

Durante la inicialización, el servicio web se encarga de establecer las comunicaciones con las distintas bases de datos, con PostgreSQL para obtener la información de los modelos disponibles, y realizar la descarga de la última versión si aplica, y con MongoDB para obtener la información de los usuarios dados de alta en el Módulo Productor.

En el diagrama de flujo de la Figura 3.7 puede verse el comportamiento general de la aplicación cuando se trata de consumir el servicio que esta ofrece.

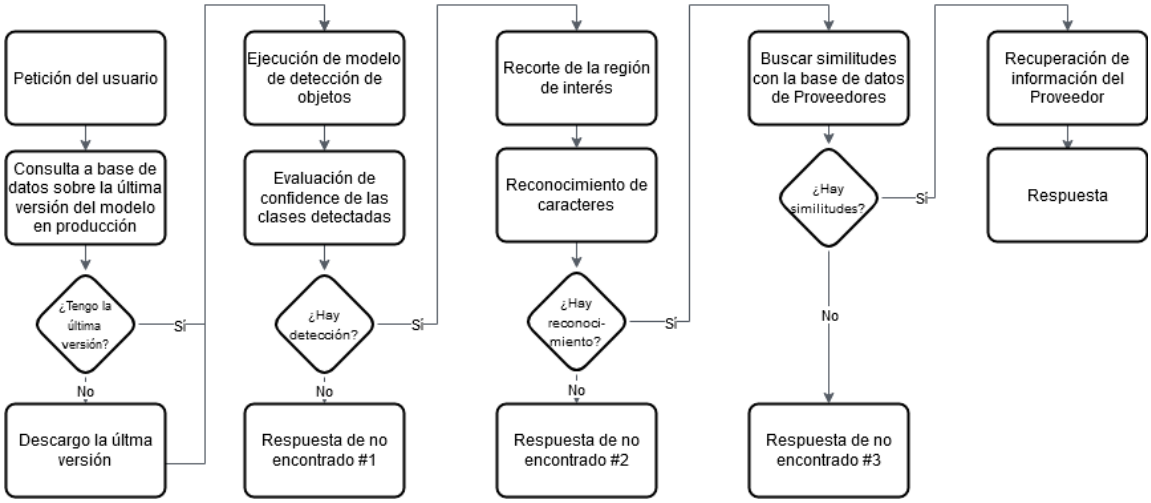


Figura 3.7 Diagrama de flujo general de la aplicación

La petición llega vía HTTP POST junto con la representación binaria de una imagen que, presumiblemente, contendrá al menos un panel informativo además de un dedo índice señalándolo.

Después de comprobar que se dispone de la última versión del modelo de detección de objetos, y de descargarlo si no es así, tiene lugar la fase de detección donde se procede a realizar la inferencia sobre este con la imagen proporcionada por el usuario. El modelo devuelve por cada elemento detectado la localización de los cuatro puntos que forman su *bounding box*, su probabilidad de clase y la clase a la que pertenece. Se ha tomado como criterio establecer un umbral con una probabilidad de 0,75 como mínimo para aceptar estas predicciones y únicamente considerar la predicción de clase 'Dedo' que maximiza este valor.

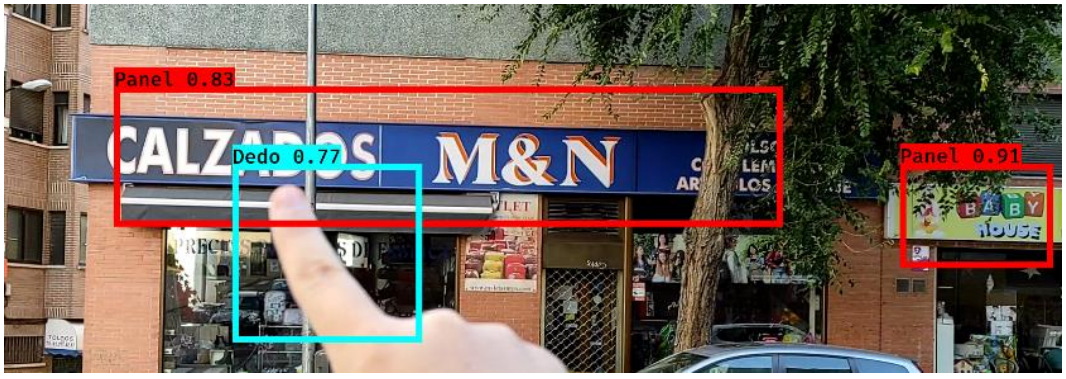


Figura 3.8 Ejemplo de predicción de clases con YOLOv3

El siguiente paso, que constituye la fase de reconocimiento, consiste en calcular el *IoU* entre cada par de predicciones 'Panel'-'Dedo' disponibles seleccionando aquel que lo maximiza. La región que contiene el panel es recortada y enviada al servicio de reconocimiento de caracteres Rekognition; este servicio devuelve por cada predicción las cuatro coordenadas del área donde ha encontrado texto, así como el texto contenido.

Figura 3.9 Ejemplos de recortes de los paneles

Con cada una de las cadenas de caracteres extraídas se aplica la distancia de Levenshtein sobre los nombres de los servicios de los usuarios dados de alta en la solución a través del Módulo Propietario; aquel servicio cuyo nombre minimice esta métrica con alguna de las cadenas de caracteres se tomará como candidato. Ha sido necesario establecer un umbral por el cuál un servicio puede ser candidato ya que, de lo contrario, puede suceder que el nombre que lo represente sea el que más se parezca a una cadena de caracteres dada, pero al mismo tiempo ser muy distinto. La distancia ente una cadena de caracteres ofrecida por el servicio de Rekognition y un nombre de un servicio debe ser estrictamente menor que la quinta parte del doble de la longitud de la cadena de caracteres dada por el servicio. La Figura 3.10 muestra algunos ejemplos.

	Input OCR	Comercio	Distancia
0	CALZAD DS M&N	CALZADOS M&N	2
1	LEC	LETICIA'S	6
2	LES	LETICIA'S	6
3	LET	LETICIA'S	6
4	CALZAD	CALZADOS M&N	6
5	LEC	LETICIA'S	6
6	LES	LETICIA'S	6
7	LET	LETICIA'S	6
8	AR LOS	LOS AMIGOS	7
9	LOS	LOS AMIGOS	7

Figura 3.10 Ejemplos de medidas de distancias de Levenshtein

Finalmente, identificado el nombre del servicio referenciado en la imagen, se procede a recuperar la información disponible almacenada en la base de datos MongoDB; denominado este proceso como fase de procesamiento. En la respuesta se facilita al usuario el resultado de cada uno de los procesos que tienen lugar al consumir el servicio o, en caso de que el resultado de un proceso no fuese satisfactorio para continuar con el algoritmo encargado del procesamiento, al usuario se le devolverá una respuesta parcial. La Figura 3.11 muestra un ejemplo de mensaje producido en formato JSON tras la finalización del proceso.


```
{'response': {'id': 'f368dd38-4e59-4dea-8e78-b672ffaa088c',
  'response_detection': {'elapsed': 2.246},
  'response_rekognition': {'elapsed': 2.362},
  'response_processing': {'payload': {'user': 'CALZADOS M&N',
    'matches': [[0, 'CALZADOS M&N', 'CALZADOS M&N'],
     [4, 'CALZADOS M&N', 'CALZADOS'],
     [7, "LETICIA'S", 'LSC'],
     [7, "LETICIA'S", 'LEE'],
     [7, 'LOS AMIGOS', 'AR LOS'],
     [7, "LETICIA'S", 'LSC'],
     [7, "LETICIA'S", 'LEE'],
     [7, 'LOS AMIGOS', 'LOS'],
     [8, "LETICIA'S", 'AR'],
     [9, 'CALZADOS M&N', 'M&N']],
    'messages': {'application': [],
     'publications': [{'content': '¡Rebajas de enero!\n\nAprovecha esta gran oportunidad llevándote zapatos de
piel de manufactura artesanal a mitad de precio.\n\nCORREEEEE 💃\u200d💰💃\u200d👠',
       'date': 1610065293},
      {'content': 'El tiempo estimado de espera en este establecimiento es de 10 minutos',
       'date': 1610072568}],
     'opinions': [{'content': '#fba_iot_tfm #CALZADOS_M&N Mientras esperaba a que comenzara el espectáculo de
la plaza entré a esta zapatería.',
       'date': 1609526600},
      {'content': '#fba_iot_tfm #CALZADOS_M&N Fantástica la oferta de enero. ¡¡¡No había visto nunca nada igua
l!!!',
       'date': 1609546980}],
     'sensors': [{'content': {'ambiance': 22, 'ocupation': 3},
       'date': 1608225423}]}},
   'elapsed': 0.008}},
 'success': True,
 'elapsed': 4.617,
 'init': '2021-01-08 06:29:47.887618',
 'end': '2021-01-08 06:29:52.504371'}
```


Figura 3.11 Ejemplo de información proporcionada tras la finalización del proceso

Tras la definición del comportamiento de los módulos indicados, así como de presentar los procedimientos llevados a cabo que han permitido realizar la fase de entrenamiento del modelo de detección de objetos, en el siguiente apartado se presentan los entornos de ejecución de los distintos componentes que integran los módulos, así como la conexión entre todos ellos.

3.3 Publicación en entornos cloud de la solución propuesta

Después de evaluar la oferta y el coste de los distintos recursos ofrecidos por los principales proveedores de servicios cloud se toma la decisión de dividir la arquitectura de la solución propuesta en dos componentes principales, a saber: Google Colab y Amazon Web Services (AWS).

La ingesta de los datos de entrenamiento, el desarrollo y el entrenamiento del modelo, así como la integración de MLflow Registry con el fin de poder registrar cada entrenamiento y salvar los modelos obtenidos, se realizará en Google Colab. El *dataset* de entrenamiento queda almacenado en Google Drive restringiendo el acceso a cualquier usuario ajeno a este trabajo. Todas las implementaciones llevadas a cabo en la plataforma de Google corresponden al Módulo IA.

En Amazon Web Services se ha dispuesto de los recursos necesarios para poder almacenar el registro de todos los entrenamientos y los modelos entrenados de los que hace uso el Módulo IA. De forma adicional, se despliegan los componentes necesarios para habilitar los Módulos Propietario y Consumidor.

La arquitectura propuesta se muestra en la Figura 3.12.

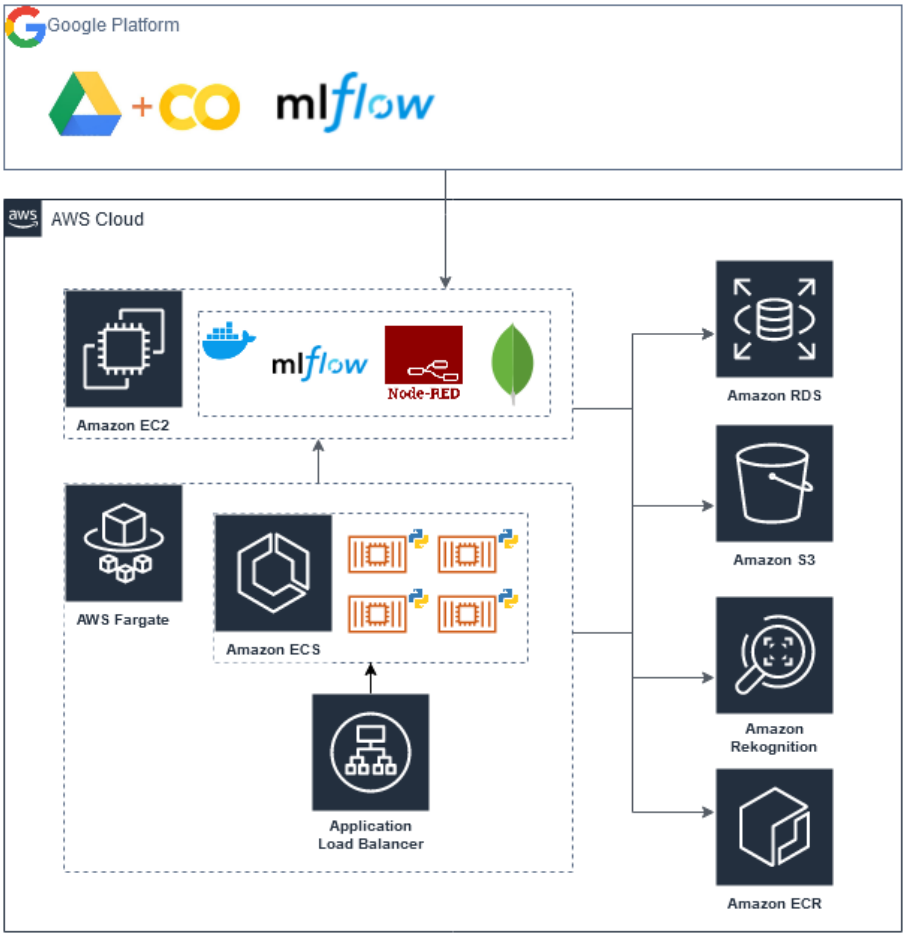


Figura 3.12 Servicios cloud y componentes

Ambos componentes de la arquitectura precisan comunicarse con el fin de que MLflow, el framework que hace posible realizar la gestión del ciclo de vida del modelo entrenado, pueda a través de MLflow Tracking registrar todos los entrenamientos llevados a cabo desde Google Colab. La base de datos Amazon RDS (2020) y el espacio de almacenamiento Amazon S3 (2020) juegan, por tanto, un papel principal; los registros y los modelos quedarán almacenados en estos servicios a la espera de ser consumidos. La base de datos Amazon RDS alojará la información que proporciona MLflow Tracking acerca del registro de cada uno de los entrenamientos que se lleven a cabo; este registro contempla desde hiperparámetros seleccionados hasta métricas resultantes tras el entrenamiento del modelo. Otro de los requisitos de MLflow Tracking es el de disponer de espacio de almacenamiento para guardar los objetos, en el contexto de MLflow denominados artefactos, y por ello es preciso crear un *bucket* en Amazon S3 donde poder almacenar estos artefactos.

Para albergar el servicio MLflow Registry, aquel que permite consultar el registro de los entrenamientos y seleccionar los modelos candidatos para ponerlos a disposición del servicio web, se ha requerido del servicio de máquinas virtuales Amazon EC2 (2020). Una máquina virtual ejecuta en contenedores Docker este proceso además del servicio Node-RED y la base de datos MongoDB que corresponden al Módulo Propietario. La imagen a la que hace referencia cada contenedor Docker se encuentra publicada en Amazon ECR (2020); además de estar aquí también publicada la imagen que contiene el servicio web que forma en sí mismo el Módulo Consumidor.

Para publicar este módulo se ha hecho uso de AWS Fargate (2020), el servicio de Amazon Web Services que ofrece cómputo sin servidor para contenedores Docker. AWS Fargate no es más que un asistente que facilita la creación de un clúster de contenedores de Amazon ECS (2020) y la creación de un balanceador de carga Application Load Balancer (2020) que centraliza y distribuye las peticiones de los usuarios sobre el clúster. Este servicio proporciona alta disponibilidad a la aplicación y capacidad automática de escalado en función de la demanda de cómputo. En el momento que Amazon ECS detecta la existencia de una nueva versión de la aplicación en Amazon ECR automáticamente descargará la imagen y la desplegará sobre todo el

clúster; de la misma forma que comprueba periódicamente el estado de cada uno de los contenedores para reiniciarlos si fuese necesario.

Las instrucciones que permiten provisionar cada uno de estos servicios se encuentran en el Apéndice C. Este despliegue hace posible cumplir los objetivos propuestos en este trabajo. En el siguiente capítulo se realiza la evaluación sobre el alcance del modelo de detección de objetos desarrollado, así como del rendimiento de la plataforma dispuesta en los distintos entornos *cloud*.

Capítulo 4 - Ejecución de procesos, evaluación de la propuesta y discusión de resultados

En este capítulo se definen, en primera instancia, los procedimientos que han permitido obtener el modelo de detección de objetos propuesto en este trabajo para luego evaluar su eficacia, así como el rendimiento que ofrece la arquitectura desplegada en entornos *cloud*, ilustrada en la Figura 3.12.

En el apartado 4.1, además de realizar observaciones sobre el conjunto de datos de partida generado, se detallan las configuraciones requeridas por MLflow Projects y MLflow Tracking para entrenar el modelo YOLOv3 que han permitido maximizar la precisión en la detección y la clasificación de dedos y de paneles informativos en imágenes.

A continuación, en el apartado 4.2, se realiza la evaluación sobre los resultados obtenidos tomando como recurso las utilidades de MLflow Registry para este fin; utilidades que hacen posible la visualización de aquello que resulta relevante durante la fase de entrenamiento y de las métricas resultado de haber entrenado el modelo.

Finalmente, en el apartado 4.3, se muestra el rendimiento ofrecido por la arquitectura propuesta después de haber realizado simulaciones de solicitudes de información turística al Módulo Consumidor con la herramienta JMeter.

4.1 Entrenamiento del modelo YOLOv3

En este apartado se citan aquellos procedimientos que han resultado relevantes durante la fase de entrenamiento del modelo de detección de objetos propuesto en este trabajo haciendo hincapié en las capacidades MLflow, para poder generar y registrar un conjunto de entrenamientos, para ello se toma como punto de partida una implementación del modelo YOLOv3 (Muehlemann, A., 2019) en TensorFlow (Abadi, M. y col., 2016). Puede consultarse la licencia de este software en el Apéndice A mientras que el código fuente que ha permitdo llevar a cabo la fase de entrenamiento del modelo está disponible en los recursos del Módulo IA enumerados en el Apéndice B.

Con el fin de llevar a cabo distintos entrenamientos del modelo YOLOv3 donde MLflow forme parte de la gestión de estos ha sido necesario configurar, en primera instancia, el entorno de MLflow Projects. Esto radica en enumerar y versionar las dependencias del entorno de Conda necesarias para reproducir el entrenamiento además de:

- Establecer el número de *epochs*. Este hiperparámetro define el número de iteraciones sobre las que se procesará el conjunto de imágenes de entrenamiento a fin de ajustar los pesos de la red.
- Fijar el valor del *batch size*. A riesgo de que en cada iteración se procesa el conjunto de imágenes de entrenamiento al completo, este hiperparámetro define cómo dividir el conjunto completo para procesarlo en varios pasos, o *steps,* de forma independiente. El número de *steps* es el resultado de dividir el cardinal del conjunto de imágenes de entrenamiento entre el *batch size*.
- Definir el porcentaje del conjunto de imágenes de validación. Después de cada iteración se utiliza este conjunto, que no ha sido utilizado a fin de ajustar los pesos de la red durante la misma, para cuantificar la precisión ofrecida por el modelo. Se asigna de forma adicional el valor del *batch size* para la etapa de validación y se almacena también el número de *steps* en consecuencia.
- Establecer el *learning rate*. Este hiperparámetro indica cómo de rápido debe ajustarse el modelo a los datos de entrenamiento. Además, se fijan hiperparámetros que permiten reducir este valor de forma automática durante la fase de entrenamiento bajo determinadas circunstancias.

Para cumplir con el objetivo de obtener un modelo capaz de detectar, para luego clasificar, panales informativos y dedos índices se ha tomado, según la forma descrita en el apartado 3.1, un total de 576 imágenes donde se han etiquetado 752 paneles y 252 dedos índice. La Figura 4.1 ilustra la distribución de etiquetas sobre el conjunto de imágenes de entrenamiento.

Figura 4.1. Distribución de etiquetas dataset train-validation

Con este conjunto de imágenes de partida se ha llevado a cabo la técnica de *data augmentation* permitiendo generar nuevas imágenes a partir de estas (Jung, A., 2020). Esta técnica consiste en manipular las imágenes con el fin de tener un conjunto de datos mayor con el que entrenar. Entre todas las operaciones que subyacen bajo el concepto se ha tenido en consideración:

- Invertir las imágenes sobre los ejes horizontal y vertical 0º o 180º. Esto resulta interesante dado que el dedo índice de la mano derecha no se encuentra en la misma posición, respecto al resto de dedos, que el de la mano izquierda y añade muestras de dedos señalando a paneles que se encuentran debajo de ellos.
- Rotar las imágenes entre -15º y 15º. Esto permite generalizar la posición en la que puede estar señalando un dedo y en otros casos ser tolerante a imágenes que tengan paneles no alineados con el eje horizontal.
- Realizar la operación de *zoom* sobre un área de la imagen entre un -25% y un 25% sobre las dimensiones originales. A riesgo de que se ha tratado de elaborar un *dataset* con paneles y dedos índice con distintos tamaños resultó interesante añadir muestras adicionales.
- Modificar el brillo de la imagen entre un -50% y un 50%. Resultó interesante incluir esta operación desde el punto de vista de contemplar distintas condiciones de iluminación de forma adicional a las ya recolectadas de antemano; iluminación de día e iluminación de noche.

Para cada imagen del primer conjunto se ha creado una nueva a la que le han sido aplicadas todas las operaciones citadas siendo arbitrario el valor utilizado en cada una de ellas. En la Figura 4.2 se ilustra un ejemplo de cada una de las operaciones sobre la misma muestra.

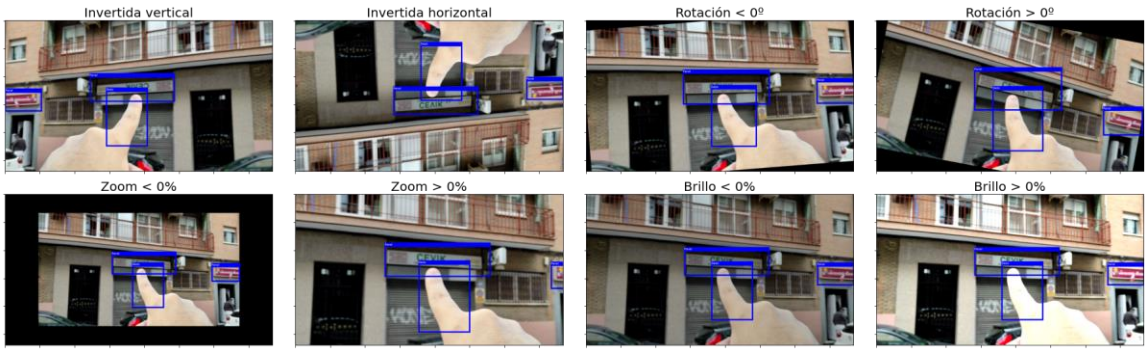


Figura 4.2 Ejemplo de operaciones de data augmentation sobre una imagen

De esta forma se ha podido duplicar el conjunto de imágenes de entrenamiento; del total se utilizará un 90% para la fase de entrenamiento, reservando un 20% de este porcentaje para la fase de validación, y un 10% para la fase de test. En el Apéndice B puede consultarse la dirección web que permite la descarga de este *dataset*.

Antes de utilizar el conjunto completo de imágenes ha sido necesario normalizarlas y estandarizarlas además de alterar su tamaño para que coincida con el tamaño de entrada del modelo YOLOv3. Mientras que el proceso de normalización consiste en dividir cada píxel entre 255, estandarizarlas parte de restarle a cada píxel la media del valor de todos los píxeles para luego dividirle la desviación estándar obtenida. Esto es particularmente importante cuando se trabaja con algoritmos de Aprendizaje Profundo ya que es bien sabido que si los mapas de características contienen valores muy grandes las operaciones de convolución generarán otros nuevos con valores aún más grandes lo que afectará al proceso de optimización.

En la fase de entrenamiento en sí misma, se utilizó el optimizador Adam (*Adaptive Moment Estimation*, Kingma y Ba, 2014) con el fin de que la función objetivo maximizara la precisión sobre el conjunto de validación. Durante cada entrenamiento ha sido posible disminuir el *learning rate* fijado de forma automática, en un factor de disminución de 10, cada vez que un *batch* es procesado si han sido procesados anteriormente ya un número dado, establecido a 3, de ellos sin que se haya conseguido

durante ese periodo reducir el error en la precisión sobre el conjunto de validación por encima de un umbral, fijado a 0, en ese intervalo en particular. Esto facilitará la búsqueda de aquella función que minimice el error sin tener como penalización sobreajuste a las imágenes de entrenamiento por mantener un *learning rate* demasiado alto.

Para evitar la ejecución de *epochs* cuando no se reduce ya de forma significativa el error sobre el conjunto de validación se han implementado mecanismos que interrumpen el entrenamiento ante tal circunstancia; este ejercicio suele denominarse con frecuencia como *early stopping*. Concretamente, si después de 5 iteraciones no se reduce esta métrica el entrenamiento queda interrumpido antes de alcanzar las 100 iteraciones. Estas 100 iteraciones están repartidas al 50% en dos fases; en la primera fase se congelan todas las capas del modelo YOLOv3 salvo las 3 últimas encargadas de realizar las predicciones a diferentes escalas, como se explica en el apartado 2.1.1.1, mientras que en la segunda se elimina esta restricción. Esta técnica se realiza en el contexto de detección de objetos a fin de reducir el tiempo necesario en alcanzar la función objetivo a través de estas iteraciones, aunque con la consecuencia de perder algo de precisión. Adicionalmente, dado que se utilizan los pesos de partida resultado de haber entrenado con anterioridad con el *dataset* de ImageNet (2020), y por tanto estar realizando lo que se conoce como *transfer learning*, esta práctica beneficia también a que estas últimas capas sean capaces de clasificar correctamente las nuevas clases.

MLflow Tracking se ha configurado para almacenar en MLflow Registry todo aquello que resulta relevante de cada uno de los procedimientos sobre los que se ha hecho mención. En particular, registra:

- Parámetros de entrada de MLflow Projects.
- Presencia de operaciones de *data augmentation*, normalización o estandarización.
- Optimizador utilizado.
- Métricas resultado del entrenamiento.
- Pesos de la red después de haber sido completada la fase de entrenamiento.

Se ilustra la Figura 4.3 un registro de entrenamiento, obtenido a partir de la interfaz web de MLflow Registry, donde puede verse toda la información registrada sobre la que se ha hecho mención en este apartado.

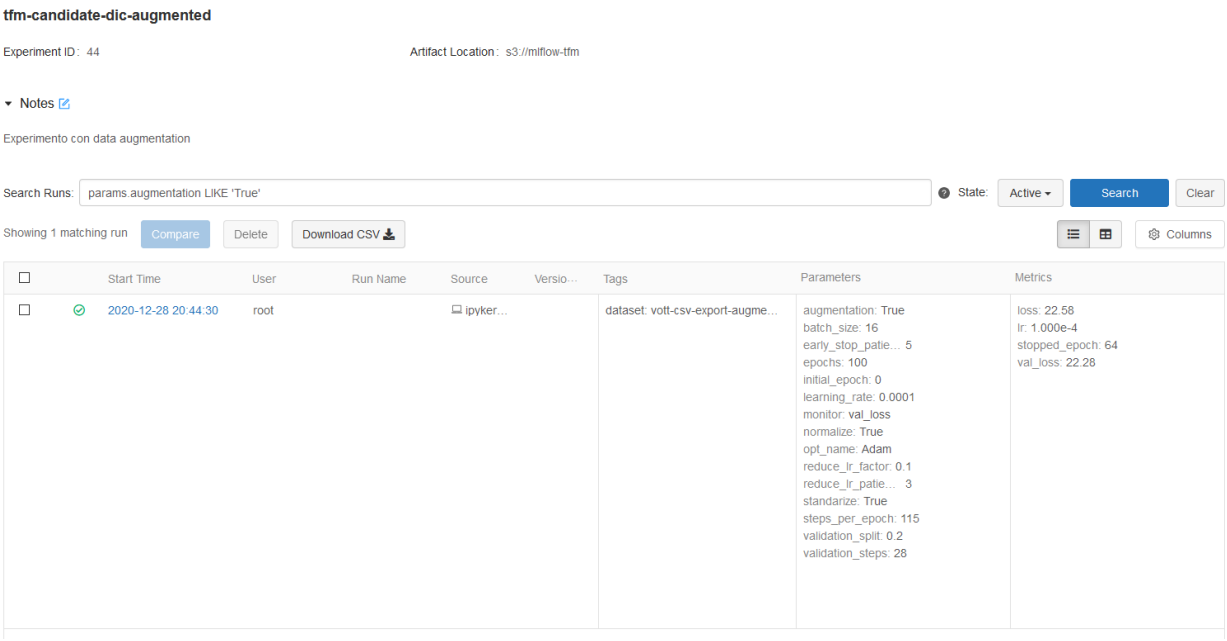


Figura 4.3 Ejemplo de registro de entrenamiento en MLflow

Este registro permitió realizar una comparación sobre los distintos entrenamientos que se llevaron a cabo, sobre los que se discute en el siguiente apartado, haciendo posible la selección de aquel modelo que, presumiblemente, ofreció mejores resultados para su explotación.

4.2 Evaluación y publicación del modelo YOLOv3

Como paso previo a realizar la evaluación sobre distintas versiones del modelo de detección de objetos, basado en YOLOv3, se realizaron varios entrenamientos a fin de establecer aquellos hiperparámetros sobre los que se ha informado sobre su valor constante en el apartado anterior. Estos hacen las veces de información de control en detrimento de otros que llegan a resultar relevantes sobre resultado del entrenamiento en sí mismo; sobre el modelo final generado con los pesos de la red ajustados.

Estos otros, el *batch size*, el *learning rate* o la presencia de imágenes generadas con *data augmentation* se ponen en este apartado a juicio para tratar de fijar la mejor

configuración posible. Teniendo esto en consideración se ha confeccionado una batería de pruebas que ha permitido obtener una serie de conclusiones.

Las distintas configuraciones del *learning rate* resultaron ser las más llamativas a simple vista; en la Figura 4.4 se ilustra la evolución del error sobre el conjunto de validación en 3 entrenamientos distintos habiendo variado en cada uno de ellos este hiperparámetro. Fijado a 1e-3 el entrenamiento finalizó de forma temprana, en la iteración número 27, por motivo de la política *early stopping* aplicada mientras que asignándole a este el valor 1e-5, la red necesitó 86 *epochs* para acercarse, que no alcanzar, al mínimo error sobre el conjunto de validación obtenido por el primer entrenamiento citado. Igualando este valor a 1e-4, la red comenzó a minimizar el error de forma más temprana que en este segundo caso alcanzando un mínimo en la iteración 69, siendo este el mínimo global entre los 3 entrenamientos analizados.

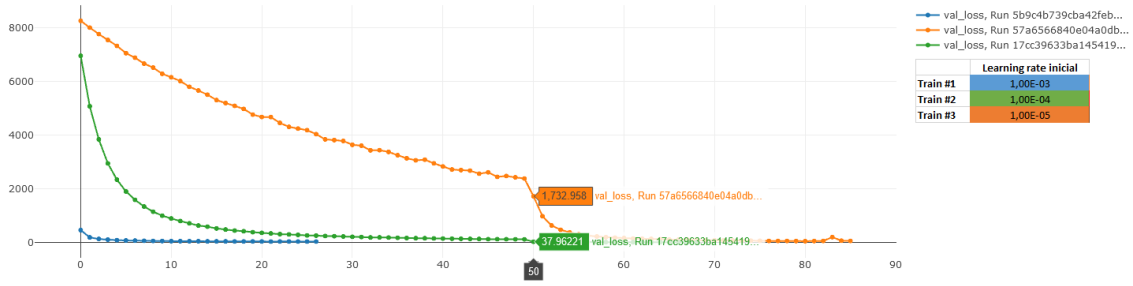


Figura 4.4. Incidencia de la configuración del learning rate sobre distintos entrenamientos

Habiendo realizado esta experimentación se fija en los sucesivos experimentos el valor de 1e-4 para el hiperparámetro *learning rate* a expensas de evaluar si el valor 1e-5 pueda causar *underfitting* u *overfitting* si se selecciona el valor 1e-3. Puede verse en la Figura 4.5 que el método para disminuir el *learning rate* de forma automática, a riesgo de observar que funciona correctamente, no termina siendo efectivo, ya que no causa incidencia en la minimización del error sobre el conjunto de validación y los 3 entrenamientos evaluados finalizan pasadas unas pocas iteraciones después de la activación del mecanismo de descenso automático del *learning rate*. En este gráfico, la línea naranja, la roja y la marrón representan el valor de este hiperparámetro a lo largo de las iteraciones que tienen lugar en cada uno de los entrenamientos evaluados; donde se establecieron los valores iniciales para este de 1e-3, 1e-4 y 1e-5, respectivamente. La curva azul, la morada y la verde representan el error sobre el

conjunto de validación; también durante el transcurso de cada uno de estos entrenamientos. Podemos ver esta ineficacia en el entrenamiento 2 si observamos la evolución de la curva morada a partir del momento en el que la línea marrón disminuye por primera vez su valor, no afectando sensiblemente a la curva en las siguientes iteraciones. El *learning rate* de este entrenamiento nuevamente disminuye y la ejecución del entrenamiento acaba finalizando después de unas pocas iteraciones de este suceso por la política de *early stopping* establecida; al no poder la red continuar minimizando el error sobre el conjunto de validación. El análisis sobre la incidencia de este mecanismo es similar para los entrenamientos 1 y 3, no siendo efectivo en ninguno de los casos como se mencionaba con anterioridad.

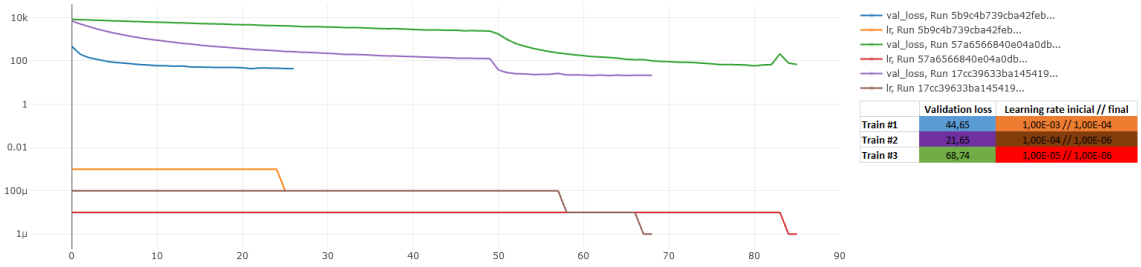


Figura 4.5. Incidencia de la reducción automática del learning rate sobre distintos entrenamientos

Con la que presumiblemente sería la mejor configuración del hiperparámetro *learning rate,* se quiso observar la evolución de la función de pérdida sobre el conjunto de validación y sobre el propio de entrenamiento a lo largo de distintas iteraciones con el *dataset* original y con el aumentado. Puede verse en la Figura 4.6 que después de descongelar las primeras capas el error sobre los distintos conjuntos va difiriendo en el entrenamiento sin *data augmentation*, representado por la curva roja (*loss*) y la verde (*validation loss*), mientras que en el propio donde sí se ha aplicado esta técnica, ilustrado mediante la curva naranja (*loss*) y la azul (*validation loss*), continua parejo hasta finalizar su ejecución.

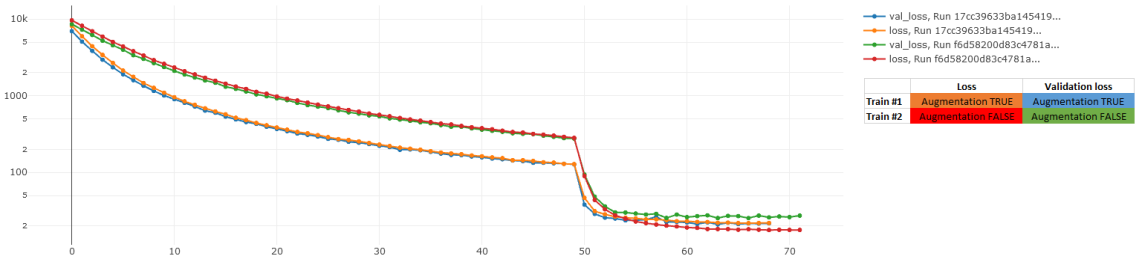


Figura 4.6. Incidencia data augmentation sobre distintos entrenamientos

Resulta positiva la aportación de las muestras generadas mediante la técnica de *data augmentation* al considerar que la red puede ser más tolerante a la posición de los paneles y de los dedos índices en las imágenes de entrada al minimizar el error de forma pareja entre ambos conjuntos. Se exploró finalmente la opción de variar el valor del *batch size* entre 4, 8 y 16 donde por limitaciones propias del hardware que ejecutaba el código de entrenamiento no ha sido posible aumentar el valor a la siguiente potencia de 2.

Puede verse en la Figura 4.7. Top 3 de configuraciones que minimizan el error sobre el conjunto de validación que los 3 entrenamientos que han reducido en mayor medida el error sobre el conjunto de validación, señalados en la parte inferior derecha con una línea vertical rosa, son el resultado de haber seleccionado, de izquierda a derecha, el *learning rate* a 1e-4 y haber aplicado la técnica de *data augmentation* cuyo análisis previo realizado indicaba que podrían ser versiones del modelo prometedoras respecto al resto de configuraciones. Con esta configuración de base se han aplicado los 3 tamaños distintos de *batch* mencionados siendo el entrenamiento donde se estableció el valor igual a 8 el que minimiza esta métrica. La incidencia del *batch size* tuvo mayor transcendencia durante la fase de *test*.

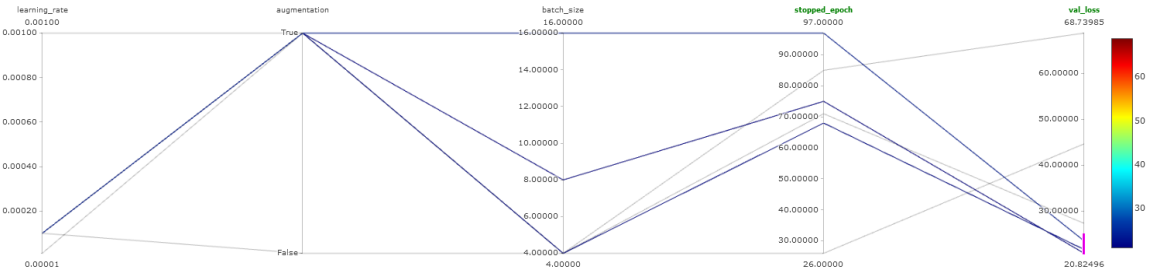


Figura 4.7. Top 3 de configuraciones que minimizan el error sobre el conjunto de validación

Con el fin de abordar la fase de *test,* con la cual evaluar la precisión del modelo, se ha reservado un 10% del *dataset* generado para este trabajo. Aunque en primera instancia se realizó la fase de evaluación con este subconjunto de muestras, esta proporción es muy similar, en términos de similitud entre las imágenes en sí mismas, a aquellas que se utilizaron para la fase de entrenamiento por lo que se tomó la decisión de generar un nuevo *dataset*, de uso exclusivo para esta fase, atendiendo al procedimiento descrito en el apartado 3.1. Este nuevo *dataset* contiene imágenes totalmente distintas al conjunto destinado de antemano para realizar la evaluación del modelo; siendo también diferentes a las que se utilizaron durante la fase de entrenamiento. La dirección de descarga de este *dataset* se encuentra disponible en el Apéndice B para su consulta. Realizar la fase de *test* con este nuevo conjunto resulta relevante al considerar que los resultados obtenidos serán los que cabría esperar una vez puesto el modelo a disposición de los usuarios a través de los servicios habilitados; en particular, a través del Módulo Consumidor.

Estas nuevas muestras tienen una etiqueta adicional que viene a indicar si las instantáneas fueron tomadas de día o de noche; para tratar también de evaluar la precisión del modelo distinguiendo tal circunstancia en el análisis que aquí se presenta. En este caso no se ha aplicado la técnica de *data augmentation* y el conjunto de test se compone de 403 imágenes con un volumen de etiquetado de 356 y 580 entre dedos índice y paneles, respectivamente.

En el contexto de detección de objetos, como se ha introducido en el apartado 2.1.1, la eficacia de un modelo puede medirse en términos de las métricas *Recall*, *Precision* y *Average Precision* (AP). Cada una de estas métricas se obtiene mediante la evaluación de cada una de las clases con las que el modelo está entrenado en base a la probabilidad obtenida de presencia un objeto de cada una de estas en los *bounding boxes* propuestos. La métrica mAP se define de forma complementaria para calcular la media de la precisión promedio entre todas las clases con la que poder evaluar al modelo en sí mismo. Para hacer el cálculo de estas métricas se toma como partida código fuente de terceros (Anh, 2017). Los recursos generados a partir de este código fuente, cuya licencia puede consultarse en el Apéndice A, pueden encontrarse listados en el Apéndice B.

Habiendo seleccionado un subconjunto de versiones del modelo YOLOv3 potencialmente prometedoras, aquellas obtenidas como resultado de haber realizado la batería de entrenamientos previamente analizada, se ha utilizado cada una de ellas para hacer inferencia sobre el total del conjunto de imágenes de *test*. En el apartado 2.1.1.1 se analiza el resultado de cada una de las predicciones previas del modelo YOLOv3; en cada una de ellas contamos con información sobre la posibilidad de existencia de un objeto, las probabilidades de que ese objeto pertenezca a cada una de las clases contempladas, dos en este caso, y las cuatro coordenadas del *bounding box* que potencialmente contiene el objeto.

Dado que el conjunto de *test* se encuentra debidamente etiquetado, que contiene información sobre los *bounding boxes* y los objetos contenidos en ellos, resulta posible comparar los *ground truth bounding boxes* que representan estos con los propuestos por el modelo. En este análisis se establece, en primera instancia, un umbral de 0.5 como la probabilidad mínima de pertenencia de clase y un umbral, también de 0.5, en el índice *IoU* mínimo entre 2 *boxes* para determinar cuándo una predicción es acertada.

A fin de seleccionar a grano grueso aquellas configuraciones que presentaron mejores resultados en términos de mAP se ha elaborado el gráfico mostrado en la Figura 4.8 donde se presentan en el eje de abscisas las distintas configuraciones (en orden descendente: versión del modelo*, batch size, learning rate*, y *data augmentation*) mientras que las barras representan el resultado de mAP obtenido para cada una de ellas. Este valor de mAP se desglosa por cada configuración en tres segmentos distintos; mientras que *A* indica que se utilizaron para esa evaluación el conjunto completo de imágenes de test, *D* y *N* indican el valor de mAP cuando sólo se utilizó un subconjunto concreto de estas imágenes. A saber, de las imágenes tomadas durante el día y de las propias tomadas por la noche, respectivamente.

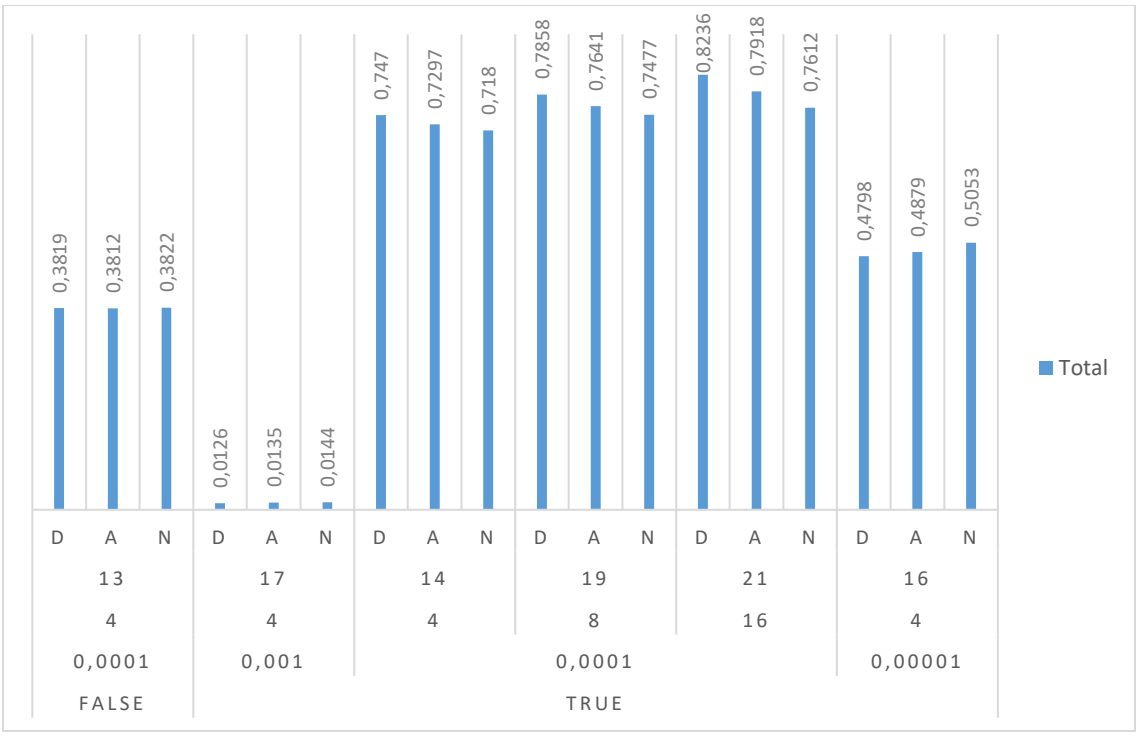


Figura 4.8. Evaluación de la métrica mAP sobre los modelos obtenidos resultado de la batería de configuraciones de hiperparámetros

Como hacía pensar la Figura 4.4, el resultado obtenido de la versión 16 parece indicar que la red no aprendió correctamente durante la fase de entrenamiento y el ofrecido por la versión 17 confirmó que el *learning rate* era demasiado alto, causando sobreajuste a las imágenes de entrenamiento. Además, puede confirmarse con este gráfico que la aportación de la técnica de *data augmentation* resultó positiva permitiendo descartar la versión 13 del modelo donde no se empleó la misma durante la fase de entrenamiento. Poniendo el foco en las versiones 14, 19 y 21 donde se fija el *learning rate* a 1e-4 y se emplea la técnica de *data augmentation* puede observarse que según crece el *batch size* la métrica mAP también lo hace. Puede añadirse que, en términos de esta métrica, la precisión resulta ser más alta cuando se procesan las instantáneas que se toman de día que cuando se capturan de noche.

Para analizar la media de la precisión promedio, haciendo hincapié en la precisión en la localización de los objetos en particular, se procedió a variar el índice *IoU* mínimo, establecido en primera instancia a 0.5, para considerar un acierto en la predicción de localización. En la Figura 4.9 se presentan en el eje de abscisas las distintas versiones seleccionadas junto con los distintos valores del índice *IoU,* en el rango [0.6,0.9],

contemplados en este análisis. De forma análoga al gráfico anterior, las barras representan el mAP obtenido salvo que en esta ocasión se ha empleado únicamente el *dataset* de *test* que incluía tanto las fotos tomadas de día como las capturadas de noche.

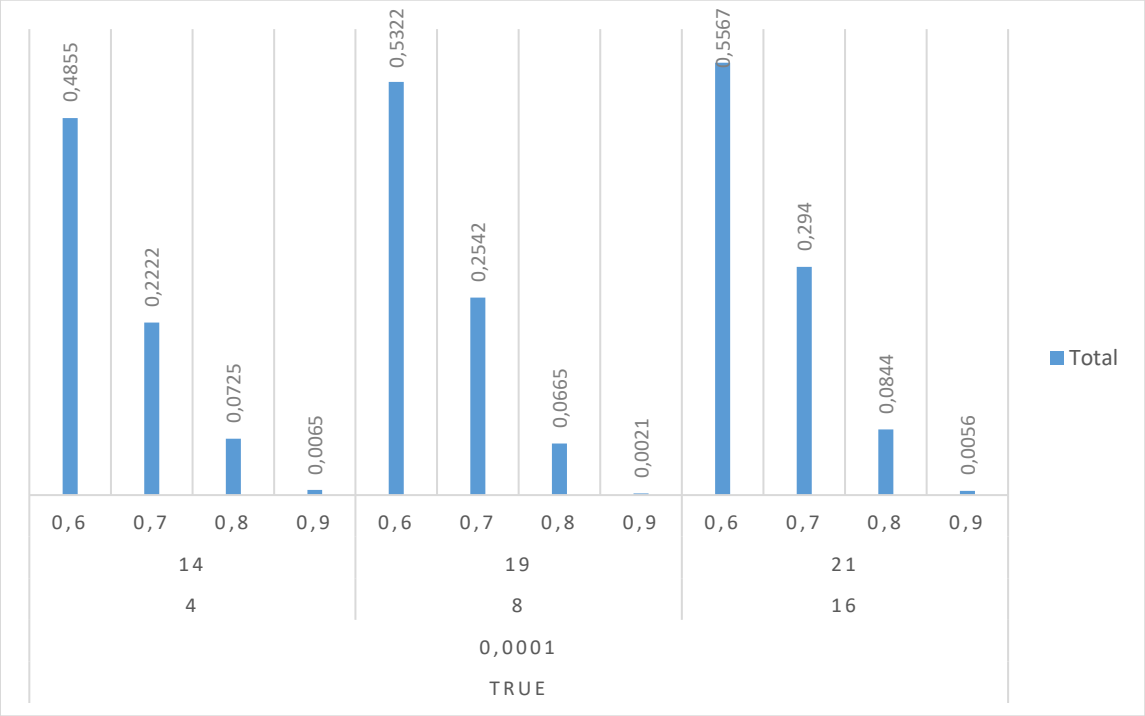


Figura 4.9. Evaluación de la métrica mAP sobre las versiones del modelo candidatas variando el índice IoU

Puede observarse en este gráfico que la métrica mAP desciende drásticamente en los 3 casos en una proporción equivalente según se aumenta el índice *IoU* en un factor de 0.1. Por tanto, se procedió a evaluar la variación de la precisión promedio (AP) sobre las distintas clases que contempla el modelo a fin de conocer la incidencia de la variación del índice *IoU* entre cada una de las clases. Los resultados de esta evaluación pueden verse en la figura Figura 4.10.

Versión 21: IoU = 0.5, mAP: 0.7918

	label	AP	recall	precision	support	TP	FP
0	Dedo	0.900609	0.935393	0.932773	356.0	333.0	24.0
1	Panel	0.683016	0.701724	0.869658	580.0	407.0	61.0

Versión 21: IoU = 0.6, mAP: 0.5567

	label	AP	recall	precision	support	TP	FP
0	Dedo	0.505320	0.688202	0.686275	356.0	245.0	112.0
1	Panel	0.608066	0.650000	0.805556	580.0	377.0	91.0

Versión 21: IoU = 0.7, mAP: 0.2940

	label	AP	recall	precision	support	TP	FP
0	Dedo	0.123049	0.331461	0.330532	356.0	118.0	239.0
1	Panel	0.465045	0.555172	0.688034	580.0	322.0	146.0

Versión 21: IoU = 0.8, mAP: 0.0844

	label	AP	recall	precision	support	TP	FP
0	Dedo	0.006899	0.070225	0.070028	356.0	25.0	332.0
1	Panel	0.162001	0.287931	0.356838	580.0	167.0	301.0

Versión 21: IoU = 0.9, mAP: 0.0056

	label	AP	recall	precision	support	TP	FP
0	Dedo	0.000048	0.005618	0.005602	356.0	2.0	355.0
1	Panel	0.011076	0.060345	0.074786	580.0	35.0	433.0

Figura 4.10. Evaluación de la métrica AP sobre el modelo final variando el índice IoU

Resulta interesante poner foco en las tablas donde el índice *IoU* se encuentra en el rango [0.5,0.7] donde hay un menor descenso de la métrica *precision* en la clase Panel en comparación con el descenso que se produce para la clase Dedo. Esta métrica está en relación directa con el descenso de los verdaderos positivos y el aumento de los falsos positivos como puede verse en el apartado 2.1.1. Al aumentar el umbral del índice *IoU* se produce un transvase directo entre ambos criterios, lo que hace descender esta métrica.

Esto puede deberse a las proporciones que pueden ocupar objetos ambas clases en una imagen. A fin de precisar con la detección de caracteres resulta prioritario trabajar con paneles donde los caracteres son grandes y bien distinguibles, por lo que han ocupado gran parte de las muestras de entrenamiento; motivo por el que la predicción de esta clase puede resultar ser más tolerante a la pérdida de localización. En contraposición las predicciones de la clase Dedo resultan menos tolerantes a esta pérdida, véase la variación de la precisión promedio (AP) entre el rango [0.5,0.6] del índice *IoU*, donde la caída es drástica.

Independientemente de este hecho, la fase de *test* resultó satisfactoria para el propósito de este trabajo debido a que el *recall* de la clase Dedo nos indica que se detectan y se clasifican correctamente la gran mayoría de los dedos índice si se desprecia la pérdida de localización, considerando el valor del índice *IoU* mínimo a 0.5 como asumible. El caso habitual será procesar instantáneas donde existe un único dedo en vertical con la intención de señalar un cartel de grandes proporciones respecto al dedo o que se encuentre muy distante de otros carteles que puedan aparecer en la imagen como ocurre en la Figura 4.11 por lo que puede asumirse esa pérdida.

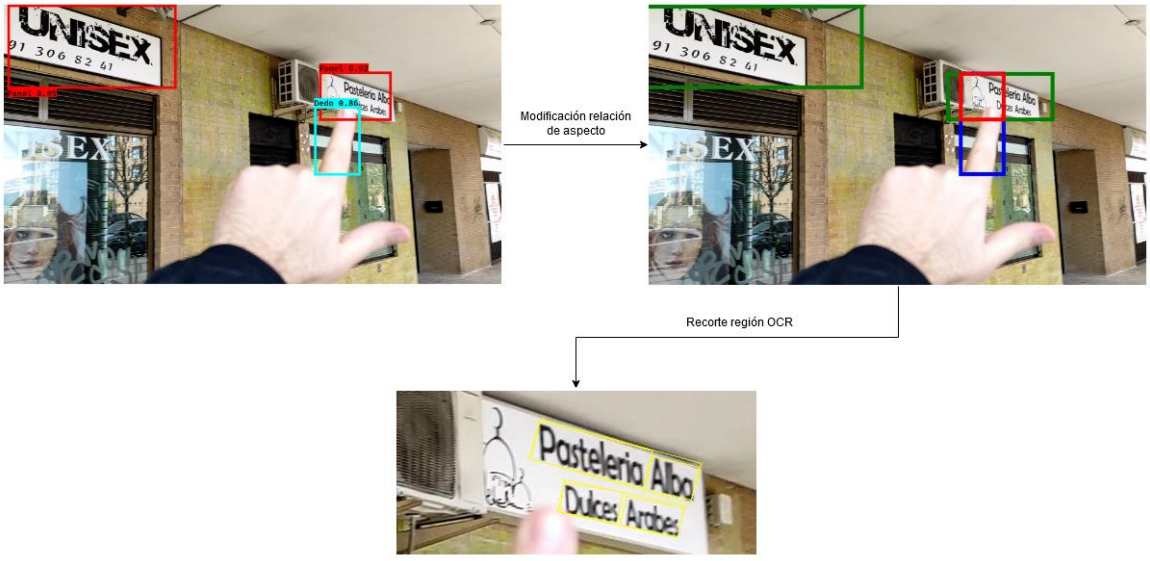


Figura 4.11. Ajuste fino modelo de detección de objetos

Observando minuciosamente las predicciones de la clase Panel, más tolerantes a la pérdida de localización, se ha podido comprobar que parte de esta pérdida tiene como hándicap recortar el área susceptible a enviar al servicio de OCR de AWS

Rekognition, como se indica en el apartado 2.4.3, por lo que se toma la decisión de modificar su relación de aspecto respecto a las dimensiones de los mismos en la aplicación como se indica en esta misma figura, 4.11.

Habiendo realizado este análisis se tomó la decisión de publicar la versión número 21 a través de la interfaz web de MLflow Registry como puede verse en la Figura 4.12.

Figura 4.12. Publicación de la versión final del modelo en la interfaz web de MLflow

De este modo se pone a disposición del Módulo Consumidor una versión del modelo YOLOv3 entrenado para ser distribuido, bajo demanda, por todo el clúster desplegado en el entorno *cloud;* clúster que constituye en sí mismo al Módulo Consumidor que contiene el servicio de información turística. En el siguiente apartado se discute el rendimiento y el alcance del Módulo Consumidor ante la petición simultánea de solicitudes.

4.3 Análisis de rendimiento del servicio de información turística

Una vez finalizada la fase de entrenamiento del modelo de detección de objetos propuesto en este trabajo, soportada por el *dataset* generado y etiquetado ad-hoc siguiendo los procedimientos citados en el apartado 3.1, y publicada una versión específica después de haber realizado la fase de test, puede afirmarse que el Módulo IA se encuentra ya disponible para ser explotado.

Con el fin de consumir información turística a través del Módulo Consumidor, y así poder evaluar el rendimiento del servicio de información turística, se ha introducido a través de los distintos canales disponibles en el Módulo Propietario, definidos en el apartado 3.2.1, información de estas características habiendo habilitado 4 comercios en total con el objetivo de que el primero pueda recuperarla con las técnicas basadas en Inteligencia Artificial implementadas tal y como se especifica en el apartado 3.2.2.

Para realizar la prueba de rendimiento del Módulo Consumidor, aquel que presumiblemente tendrá más solicitudes de usuarios simultáneas, se ha procedido, a través de AWS Fargate, al despliegue de un clúster con 4 contenedores independientes que ejecutarán el servicio de información turística, cada uno provisionado con 1 vCPU y 2 gb de RAM *on spot*, y de un balanceador de carga Application Load Balancer que centralizará las peticiones de los usuarios y sobre el que se delega la comunicación con cada uno de los contenedores dispuestos en el clúster.

Con el software JMeter (2020) es posible realizar simulaciones de solicitudes simultáneas de distintos usuarios. Dada la configuración que da soporte al Módulo Consumidor se pone a discusión el siguiente experimento:

- Se envían 12 solicitudes en el intervalo de 5 segundos.
- Transcurridos los 5 segundos, se envían otras 12 solicitudes.
- Se repite el proceso hasta en 4 ocasiones, generando un total de 48 solicitudes.

Cada una de las solicitudes lleva adjunta una imagen de la que se conoce que la aplicación será capaz de identificar el servicio turístico en cuestión y de recuperar la información turística asociada, almacenada con anterioridad. Por tanto, se pretende comprobar el rendimiento de esta aplicación en el caso ideal. Identificar el servicio turístico señalado en la imagen es la tarea que se presumía que más se demoraría en completarse. Por otra parte, antes de realizar esta prueba se conocía el tiempo medio de respuesta del servicio de OCR de Amazon Rekognition. La información procedente de distintos canales es recuperada de forma ágil, dentro del entorno de red local, atendiendo a relaciones entre los distintos modelos de datos contemplados.

El resultado esperado de esta prueba es la ejecución de todos los procedimientos necesarios en un tiempo asumible en el contexto de este trabajo recuperando toda la información almacenada de antemano. Se ilustra en la Figura 4.13 un ejemplo de respuesta en forma de texto plano cuando el servicio logra recopilar toda esta información.

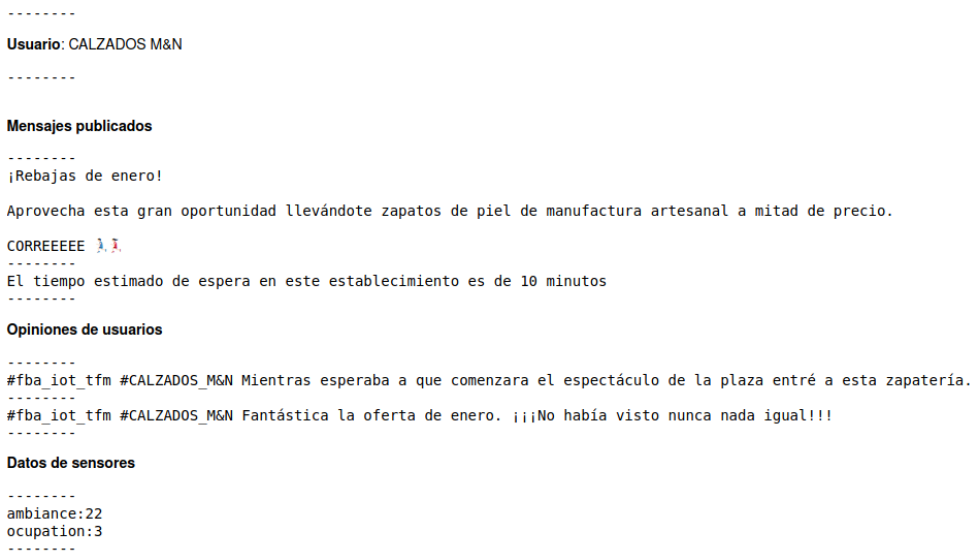


Figura 4.13. Información de interés proporcionada ante la detección de un servicio turístico

En esta Figura 4.13 pueden verse, a modo de ejemplo, las publicaciones asociadas a un servicio turístico identificado mediante la utilización del modelo de detección de objetos desarrollado. De arriba hacia abajo, podemos observar publicaciones procedentes de la interacción con el cliente de mensajería de Telegram y de otros sistemas más complejos mediante la publicación de mensajes a través del canal HTTP. Dentro del contexto de las *Smart Cities* se contempla que existan fuentes de datos abiertos, obtenidos en algunos casos mediante la aplicación de procedimientos complejos basados también en Inteligencia Artificial, que proporcionen, mediante subscripción, información que pueda resultar relevante. Las publicaciones se complementan con las opiniones de los usuarios de este servicio turístico identificado y con la información proporcionada por el despliegue de una red IoT, que se encuentra dentro del establecimiento, basada en la integración del protocolo de comunicación MQTT.

A continuación, la Tabla 4.1 viene a ilustrar el resultado del experimento en términos del tiempo medio de respuesta para recuperar esta información por parte del servidor, ~8 segundos, el mínimo y el máximo, ~3 segundos y ~20 segundos respectivamente, así como el rendimiento ofrecido, ~1.2 solicitudes por segundo. Puede verse también que el porcentaje de errores es nulo lo que certifica la robustez en la gestión de los recursos de los servicios *cloud* dispuestos.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput
HTTP Request	48	8088	3176	20139	3615,09	0	1,2/sec

Tabla 4.1. Resultados generales de simulación de solicitudes al Módulo Consumidor

Tener la capacidad de atender a ~1.2 solicitudes por segundo, dado que es posible procesar 4 imágenes simultáneamente y que el tiempo de procesamiento mínimo es de ~3 segundos, resultaría beneficioso en el contexto del procesamiento por lotes. Sin embargo, en el contexto de procesamiento en tiempo real esta medida de rendimiento no resulta relevante como sí lo es el tiempo de procesamiento mínimo; la medida de procesamiento máxima, así como el valor medio, son también despreciables al no referirse a tiempo de procesamiento si no de latencia. En la Figura 4.14 se muestra que la solución tarda entre 3 y 4 segundos en ejecutar los algoritmos necesarios para dar respuesta a los usuarios y que los tiempos máximos, y por consiguiente los medios, presentados con anterioridad se deben al bloqueo del balanceador de carga de esas solicitudes ante la inexistencia de recursos disponibles.

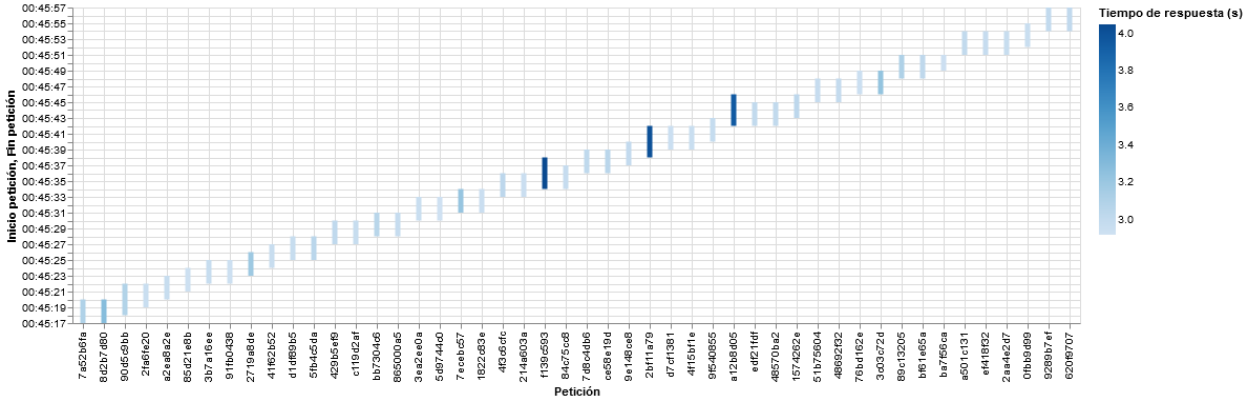


Figura 4.14 Comportamiento del clúster ante la simulación de solicitudes al Módulo Consumidor

En esta figura puede apreciarse también que solo 4 solicitudes, como máximo, se solapan en el tiempo, validando la gestión del balanceador de carga en el uso de los recursos disponibles. A la hora de identificar el tiempo requerido por cada una de las etapas, definidas en el apartado 3.2.2, pudo detectarse que deshabilitando la escritura de etiquetas y *bounding boxes* sobre las imágenes que se devuelven al usuario, de utilidad a la hora de ilustrar todos los procesos que se llevan a cabo en la aplicación, el tiempo de respuesta medio se redujo en aproximadamente 2 segundos. A la izquierda en Figura 4.15 puede verse el tiempo requerido por cada una de estas etapas habiendo

deshabilitado esta escritura mientras que a la derecha se muestra lo propio sin haber realizado esta acción. La fase de detección ejecuta código fuente propio por lo que pudo mejorarse el rendimiento significativamente; sin embargo, en la fase de reconocimiento los cambios que se han podido hacer son mínimos y el tiempo requerido por esta resulta ser prácticamente equivalente a la latencia con el servicio de Amazon Rekognition.

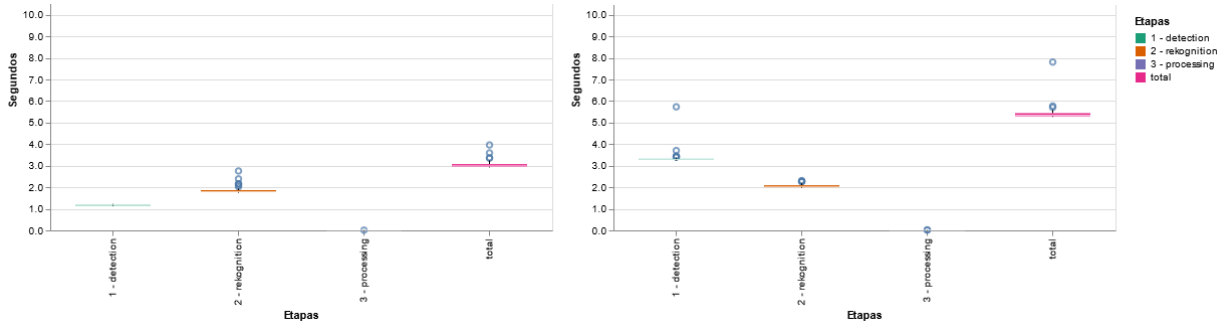


Figura 4.15 Tiempo de respuesta por etapas durante la simulación de solicitudes al Módulo Consumidor

En definitiva, para tener una latencia de ~3 segundos, y no un tiempo de respuesta de ~3 segundos, siendo un tiempo aceptable en el contexto de este trabajo, es necesario provisionar un contenedor por usuario. Para provisionar los contenedores en el clúster que fuesen necesarios en cada momento se han registrado políticas de *autoscaling* para aumentar o reducir la capacidad de cómputo en base al uso de CPU de los contenedores disponibles.

De forma independiente a la simulación expuesta con anterioridad se ha realizado una nueva con la cual poder validar esta capacidad de *autoscaling*. Para ello se han suprimido en primera instancia 3 de los 4 contenedores activos; persistiendo tan solo uno de ellos habiéndole provisionado los mismos recursos que durante la simulación anterior en términos de vCPU y RAM. La política registrada ejecutará a los 30 segundos los mecanismos convenientes para provisionar 2 contenedores adicionales cuando el uso de vCPU del que se encuentra disponible tenga una media por encima del 15% en el intervalo de un minuto; dado que cuando un contenedor no atiende peticiones, el uso de vCPU se encuentra prácticamente al 0%, una media del 15% durante un minuto resultó ser un umbral aceptable para este experimento. En este caso, la configuración de JMeter se simplificó para realizar una petición por segundo de forma indefinida.

La Figura 4.16 corresponde a un extracto de este experimento a partir de cuándo se activó el evento marcado por la política registrada para provisionar 2 contenedores adicionales hasta que estos estuvieron disponibles para el balanceador de carga. Tomando como asunciones que a partir de este evento existe un *delay* de 30 segundos para provisionar los nuevos contenedores y que un contenedor requiere entre 20-30 segundos adicionales para provisionarse y ejecutar los procesos necesarios para poder atender a las peticiones del balanceador de carga, puede observarse que ~50 segundos después del inicio de la primera petición presente en la imagen, la aplicación deja de atender a las distintas peticiones de una en una pudiendo procesar dos simultáneamente; hasta 3 cuando el tercer contenedor se encuentra disponible 10 segundos después.

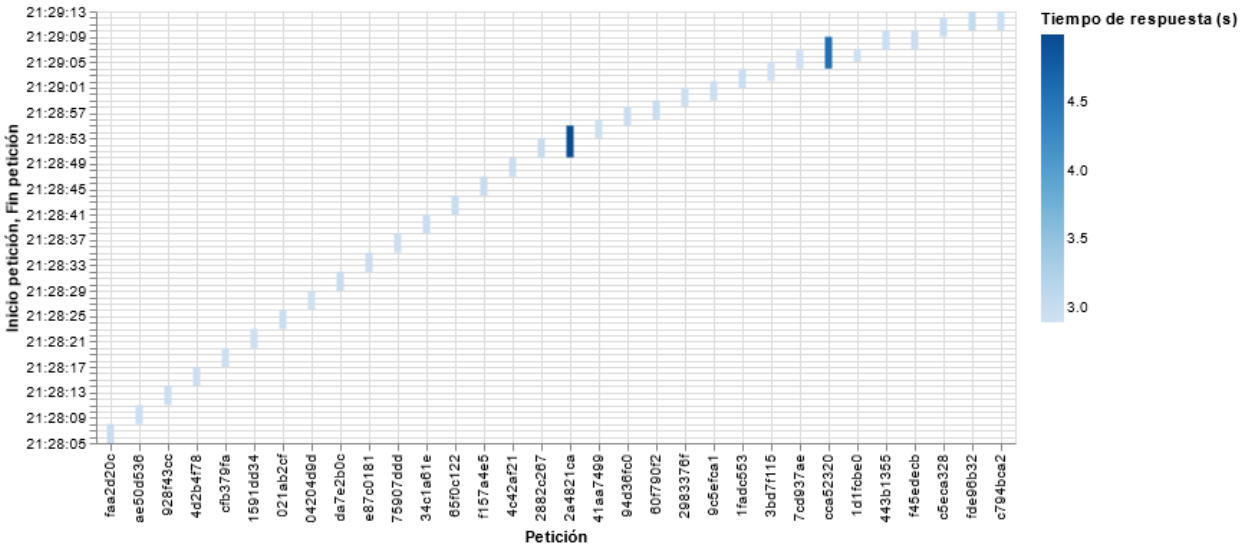


Figura 4.16 Comportamiento del clúster ante la activación de políticas de autoscaling

Esta validación resultó imprescindible teniendo en cuenta que la integración de la solución que se propone en este trabajo, en el contexto de las *Smart Cities,* solo sería viable si el servicio de información turística fuese capaz de atender a todas las peticiones que recibe independientemente del volumen de usuarios en el tiempo mínimo establecido alcanzado durante esta experimentación.

Además, estos experimentos han permitido validar el despliegue de los tres módulos en los distintos entornos *cloud*, así como delimitar su alcance.

Capítulo 5 - Conclusiones y trabajo futuro

Dentro del paradigma IoT, en este trabajo se ha tratado de habilitar canales de comunicaciones susceptibles a recibir información de interés turístico de distinta naturaleza (procedente de sensores, proporcionada por otros subsistemas o por las propias personas o recopilada de su actividad en redes sociales), para poder ser facilitada a potenciales consumidores de los servicios que una *Smart City* puede ofrecer en sus calles. Para poner a disposición estos canales se ha hecho uso de plataformas *low-code programming* que han permitido realizar desarrollos de forma ágil y eficaz acelerando así la consecución de este trabajo al conocer de antemano el funcionamiento de los distintos protocolos de comunicaciones en el ámbito de IoT que con frecuencia son utilizados. La utilización de estas plataformas, a las que se les ven muchas posibilidades, ha resultado satisfactoria.

Para acceder a esta información se proponía una aplicación que, haciendo uso de un modelo de detección de objetos, detectase cuál es la fuente de información turística solicitada a través de una imagen. En este caso, ha sido también positiva la eficacia del modelo de detección de objetos desarrollado utilizando YOLOv3 en los distintos casos de uso testados. El soporte de MLflow ha permitido un ajuste fino en la fase de entrenamiento y ha resultado relevante para distribuir el modelo sobre todo el clúster que ejecuta la aplicación. Por otra parte, las utilidades de MLflow que permiten la generación de gráficos, aquellas que han hecho posible realizar el análisis sobre los distintos entrenamientos llevados a cabo, aunque han resultado ser prácticas se ha echado en falta un mayor volumen de opciones de personalización y se espera que se introduzcan en futuras versiones. De gran importancia también ha sido la aportación del servicio de Amazon Rekognition, donde ha resultado ser mucho más efectivo para el propósito de este trabajo que las herramientas *open source* disponibles que se han evaluado.

La aplicación que consume tanto el modelo de detección de objetos desarrollado como el servicio de Amazon Rekognition es la encargada de ofrecer la información almacenada procedente de todos los canales dispuestos. Para ello, esta se ha dispuesto en forma de un servicio web tratando así de maximizar la compatibilidad

con otros subsistemas que precisen requerir de su uso como bien pueden ser las aplicaciones para smartphone; un contexto preferente.

Para realizar la integración de los distintos componentes que forman la solución que se proponía en este trabajo se han provisionado los recursos necesarios en entornos *cloud*. Se valora muy positivamente haber desplegado una solución con la capacidad de escalar en función de la demanda de usuarios y del volumen de datos a almacenar lo que puede ayudar en unos casos a dar servicio de forma indiscriminada y en otros a reducir costes en periodos de menor afluencia de actividad turística.

El trabajo futuro gira en torno al aumento de capacidades de la solución implementada y al consumo de esta, más concretamente se propone:

- Aumentar el volumen de canales disponibles de recepción de información turística a fin de adaptarse a distintos despliegues de redes de sensores IoT.
- Desarrollar, integrar y hacer seguimiento de la evolución de un modelo de Inteligencia Artificial basado en el procesamiento del lenguaje natural que permita cuantificar la satisfacción de los usuarios al haber hecho uso de los servicios consumidos.
    - Opcionalmente, puede realizarse un análisis del coste que podría suponer una integración de estas características con el servicio de Amazon Comprehend.
- Diseñar una aplicación de *smartphone* que mediante técnicas de realidad aumentada permita disponer en pantalla la información proporcionada por el servicio de información turística mediante la comunicación con el mismo.
- Realizar un estudio del coste de Amazon Translate a fin de integrarlo en la solución permitiendo así facilitar la información recopilada en distintos idiomas.

Chapter - Introduction

Scenario and proposal

We often ask us about the weather conditions in our city, about traffic flow or the availability of parking places at our workplace. On the other hand, when we move in the early hours of the morning, we can observe how the street lights are turned off when at sunrise and the irrigation systems in the parks and gardens starts to work.

This is just one example of the role played by the paradigm known as the Internet of Things (IoT) in our cities day by day. It has been demonstrated that its technological explosion and its applicability have allowed organizations and public institutions to be more efficient in the use of energy and logistic resources, while providing their citizens with very useful information for their lives. The concept of the Smart City comes from the capacity that a city, town or village may have to collect environmental information and share it with the community or maintenance services in general, while applying or carrying out actions depends on it.

The different research areas based on Artificial Intelligence (AI) are a great asset in the context of IoT. The benefit of information collection lies in many cases in the fact that there are AI-based models that allow good decisions to be made based on this data. Sometimes, the resulting information that is used for different purposes has as its origin from the prediction of the models established for this purpose.

In this work, a proposal is made in the field of Smart Cities whose objective is to extend the services that these can offer, focusing on the tourist sector, using AI-based models, specifically using the YOLOv3 model (Redmon and Farhadi, 2018) as an object detection model in the field of Deep Learning. This adds a new proposal to those previously mentioned in the field of IoT. Specifically, this work proposes to publish, in cloud environments, resources that allow Smart Cities to publish relevant information about the services it provides, through state-of-the-art IoT protocols, messaging clients and social networks, as well as web services that allow pedestrians to consume this information.

A local restaurant will be able to publish its schedule and offer, as well as its capacity and the environmental conditions of the place where it is located, this

information can be collected thanks to the deployment of an IoT network, while a cathedral will be able to offer its visitors audio guides so that they can be heard directly on a smartphone, to mention a few examples. The consumer of these services will only have to take a photograph pointing with his index finger at the information panel that identifies them so that, with an object detection model in cooperation with character recognition services, it can be detected identifying the source of information requested to provide everything that could have been published about it. It is precisely here where the development and application of this work is focused, so that after the capture of the image that contains the indicated panel of information of interest, it is processed through techniques based on Deep Learning and using the appropriate IoT services.

Objectives, motivation and scope

IoT offers the ability to make significant contributions within the tourism industry, not only to companies in the sector but also the experience of tourists. A commercial aircraft may be equipped with wind and temperature sensors that allow the pilot to make the decision to save fuel under certain circumstances, and a train could have many other sensors that would enable the pilot to quantify the wear of fuel components in order to perform preventive maintenance that can incur economic savings, to name a few examples. In contrast, it would also be feasible for flight attendants to provide passengers with sensors capable of measuring heart rate and sweating in order to get to know their anxiety level to proportionate all the attention they might need, while on a train it would also be possible to provide interfaces which, in a centralized manner, allow passengers to modify the environmental conditions of the wagon in which they are located, also to name a few examples.

This work focuses on tourist experience and in particular to provide them with the ability to request information that is provided in advance by various sources; which may either be provided by sensor networks or by the publication of the offer of the various services of turist places.

When we make the decision to do sightseeing, as a general rule, we used to booking accommodation, to hire a means of transport and to inform us about the leisure offers of our destination. Today there are many public portals available on the internet

such as Tripadvisor or Airbnb, to name a few examples, that have tourist information about potential destinations while promoting and offering their services. It is often found that these sources have user reviews that show the quality of the experiences offered. The sale of travel services often means that the buyer needs to collect a lot of information about their destination, but that need persists even after making appropriate contracts.

The activity carried out by the tourism industry can be defined as the orchestration of various actors (host government, attractions, hospitality, local trade, crafts, culture, religion, etc.) for the benefit of the tourist experience. All these actors tend to publish the information about the services they offer on certain portals depending on the sector in which they base their activity, which causes that the search for this information may not be an agile procedure if we are already in our destination as it is diversified. This paper proposes that all information that may be of interest should be centralized, that those responsible for the services offered are those who decide what to show at any time and that the information published is as accessible as possible to tourists.

If a business is allowed to publish, modify or delete its information, it is granted the privilege of owning that information which may allow it to exploit it at will. On the other hand, being centralized, it can be beneficial from the point of view of saving the time spent to obtain it improving the tourist experience; time being a very valuable factor in vacation periods.

The non-profit nature of the tourist offices plays a differential role in the promotion of the territory to which they belong as a set services, being able to empower each of these actors regardless of the activity they carry out. In addition, it is common to find information from large cities, which is not the case with smaller ones which can be a great promotion of all the services it can offer. Assuming its role, the motivation of this work is to allow any interested to publish information about the services it offers and that this can be shared to the pedestrian in real time when they are in front of each of the establishments where the services are offered. From the point of view of IoT, this proposal allows the query of published information in real time in relation to an element of interest, allowing tourists to make decisions in place.

By exposing a scenario of use, a tourist can consult the occupancy of the bar of the tavern to which he had planned to visit, check that it is full without having to enter,

and being posible to find the information of all services centralized, he can choose to visit the nearest local trade that offer, through the solution proposed in this work, regional handicrafts at reduced prices while waiting for the tavern level to decrease.

Presented the proposal and its motivational element, the following general objectives are defined thus delimiting the scope of the project:

• Offer a methodology that allows stakeholders, within the tourism sector, to promote their offer with tools that can have a low economic impact and that does not necessarily require expert profiles in the field of information technology.

• Allow a potential consumer of these services to send an image of the region or surface on which he wants to receive information to a central server where it can be processed.

• From a picture, detect what the information is requested by the user and retrieve it. This image may contain a variable number of registered services and an index finger pointed to an information panel should appear in the photograph allowing the system to discern between possible ambiguous regions.

In order to achieve these objectives, it is taken into account that the activity carried out by these institutions does not belong to the area of information services. It is assumed, therefore, that none of them have their own systems capable of executing the proposed solution. Since some territories focus their tourist activity on a seasonal basis, it is desirable to have the ability to scale the computing capacity of the proposed solution based on user demand according to the current season.

Both assumptions lead to the decision to consume cloud services in order to save costs by deploying the necessary resources at any time and delegating system maintenance and security to vendors.

Specific objectives and work plan

The work plan is governed by the decomposition of the solution into the different components that form it and in the assessment of the requirements of each of them taking into account the scope of this work.

In order to achieve the general objectives described above, the following steps are proposed, which constitute the specific objectives of the proposal:

- Artificial Intelligence Module
    - Generation of the image dataset where information panels appear to identify the business they refer to. In addition, it will contain pictures of index fingers where it is clear that they are pointing towards the region of interest.
    - Use of tools that allows to tag images that contain the mentioned elements.
    - Training of a convolutional neural network model for object detection to identify the appearance of each of the listed labels.
    - Deploy services that allows to track the accuracy of trained models, reproduce the execution environment and provision those candidate models.
- Owner Module
    - Publication of different channels that allows different users to promote their offer of services.
    - Deployment of databases that will store information about these users and services.
- Consumer Module
    - Development of a web service that attends to the requests of potential consumers.
    - Publish the web service in serverless environments.
    - Exploitation of the trained object detection model.
    - Request web services capable of converting characters present in images to plain text.

- Implementation of communications with the AI Module and the Owner Module to inference the trained models and return to users the demanded information retrieved from the databases.

A specific user needs to make a request for tourist information by capturing an image and it send to the system through the Consumer Module. The Owner Module will facilitate the publication of information of tourist interest through different channels and will be responsible for storing it in databases. The Consumer Module allows access to this information. Thanks to the publication of the object detection model, provisioned by the IA Module, it will be possible to identify, and then retrieve, the source of information requested by users of the solution proposed in this work.

Work organization

The memory is organized as follows:

Chapter 2: Review of open source technologies that allow to meet the technical requirements of the purposed solution and those whose are within the lines of research in Artificial Intelligence that make it possible to fulfill the purpose of this work.

Chapter 3: Design of the different parts that make up the project. This chapter covers the design phase of Artificial Intelligence based models, as well as the architecture and software components that make possible the interaction of different users with the proposed system.

Chapter 4: Assessment of the solution. It aims to illustrate the effectiveness of the proposed model from an analytical perspective and the performance of the proposed platform deployed across different cloud environments.

Chapter 5: Statement of conclusions and approach to possible future work.

Chapter - Conclusions and future work

Under the IoT paradigm, this work has enabled communication channels susceptible to receive information of tourist interest of different nature (from sensors, by other subsystems or by people own or collected from their activity on social networks), in order to be able to be facilitated to potential consumers of services that a Smart City can offer in its streets. To make these channels available, I have made use of low-code programming platforms that have allowed to carry out developments in an agile and efficient way accelerating the achievement of this work by knowing in advance the operation of the different communications protocols in the field of IoT that are often used child. The use of these platforms, which are seen many possibilities, has proved satisfactory.

To access this information, an application was proposed that, using an object detection model, could detect the source of tourist information requested through an image. In this case, the effectiveness of the object detection model developed using YOLOv3 has also been positive in the various tested use cases. MLflow support has allowed fine-tuning in the training phase and has been relevant to distributing the model over the entire cluster running the application. On the other hand, the MLflow utilities that allow the generation of graphs, which have made it possible to perform the analysis on the various workouts carried out, although they have turned out to be practical, a greater volume of customization options has been missing and are expected to be introduced in future versions. The contribution of the Amazon Rekognition service has also been important, where it has proved to be much more effective for the purpose of this work than the available open source tools that have been evaluated.

The application that consumes both the developed object detection model and the Amazon Rekognition service is to provide stored information from all available channels. To do this, this has been arranged in the form of a web service thus trying to maximize compatibility with other subsystems that require their use such as smartphone applications; a preferential context.

To integrate several components that make up the solution proposed in this work, the necessary resources have been provisioned in cloud environments. It is highly valued

to have deployed a solution with the ability to scale according to the demand of users and the volume of data to be stored which can help in some cases to serve indiscriminately and in others to reduce costs in shorter periods of influence of tourist activity.

Future work turns around increasing the capacities of the implemented solution and the consumption of the implemented solution, more specifically it is proposed:

- Increase the volume of available channels for receiving tourist information to adapt to different deployments of IoT sensor networks.
- Develop, integrate and track the evolution of an Artificial Intelligence model based on natural language processing that allows quantifying users' satisfaction when using the services consumed.
    - Optionally, perform an analysis of the cost of integrating these features with the Amazon Comprenhend service.
- Design a smartphone application that through augmented reality techniques allows to have on screen the information provided by the tourist information service establishing communication with it.
- Perform an Amazon Translate cost study to integrate it easily into the solution allowing information collected in different languages.

BIBLIOGRAFÍA

1. Alam, M. M., & Rahman, S. M. (2019). Detection and Tracking of Fingertips for Geometric Transformation of Objects in Virtual Environment. In 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA) (pp. 1-8). IEEE.

2. Anh, H. N. (2017). YOLO3 (Detection, Training, and Evaluation). Disponible on-line: https://github.com/experiencor/keras-yolo3 (Accedido Diciembre 2020).

3. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265-283).

4. Amazon EC2 (2020). Secure and resizable compute capacity to support virtually any workload. Disponible on-line: https://aws.amazon.com/ec2/?nc2=type_a (Accedido Diciembre 2020).

5. Amazon ECR (2020). Share and deploy container software, publicly or privately. Disponible on-line: https://aws.amazon.com/ecr/?nc2=type_a (Accedido Diciembre 2020).

6. Amazon ECS (2020). Highly secure, reliable, and scalable way to run container. Disponible on-line: https://aws.amazon.com/ecs/?nc2=type_a (Accedido Diciembre 2020).

7. Amazon Elastic Load Balancing (2020). Distribute network traffic to improve the scalability of your applications. Disponible on-line: https://aws.amazon.com/elasticloadbalancing/?nc2=type_a (Accedido Diciembre 2020).

8. Amazon Fargate (2020). Serverless compute for containers. Disponible on-line: https://aws.amazon.com/fargate/?nc2=type_a (Accedido Diciembre 2020).

9. Amazon Rekognition (2020). Automate your image and video analysis with machine learning. Disponible on-line: https://aws.amazon.com/rekognition/?nc2=type_a (Accedido Diciembre 2020).

10. Amazon RDS (2020). Set up, operate, and scale a relational database in the cloud with just a few clicks. Disponible on-line: https://aws.amazon.com/rds/?nc2=type_a (Accedido Diciembre 2020).

11. Amazon S3 (2020). Object storage built to store and retrieve any amount of data from anywhere. Disponible on-line: https://aws.amazon.com/s3/?nc2=type_a (Accedido Diciembre 2020).

12. Amazon Web Services (2020). Cloud Computing Services. Disponible on-line: https://aws.amazon.com/es/ (Accedido Diciembre 2020).

13. Apache JMeter (2020). Load testing tool for analyzing and measuring the performance of web applications. Disponible on-line: https://jmeter.apache.org/ (Accedido Diciembre 2020).

14. Bisong, E. (2019). Google Colaboratory. In Building Machine Learning and Deep Learning Models on Google Cloud Platform (pp. 59-64). Apress, Berkeley, CA.

15. Bochkovskiy, A., Wang, C.Y., Mark-Liao, H.Y (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934v1 [cs.CV].

16. COCO (2020). Microsoft COCO: Common Objects in Context. Disponible on-line: http://cocodataset.org/#home (Accedido Diciembre 2020).

17. Docker (2020). The fastest way to securely build, share and run modern applications on a single platform. Disponible on-line: https://www.docker.com/ (Accedido Diciembre 2020).

18. Eclipse Mosquitto (2020). An open source MQTT broker. Disponible on-line: https://mosquitto.org/ (Accedido Diciembre 2020).

19. En, S., Lechervy, A., & Jurie, F. (2018). Rpnet: An end-to-end network for relative camera pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 738-745).

20. Everingham, M., Ali-Eslami, S.M., van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A. (2015). The PASCAL Visual Object Classes Challenge: A Retrospective. Int. J. Computer Vision, 111(1), 98–136.

21. He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. In Proc. of the IEEE conference on computer vision and pattern recognition, pp. 770-778.

22. Jung, A. (2020). Imgaug: a library for image augmentation in machine learning experiments. Disponible on-line: https://buildmedia.readthedocs.org/media/pdf/imgaug/stable/imgaug.pdf (Accedido Diciembre 2020).

23. Kingma, D.P., Ba, J.L. (2015). Adam: A method for stochastic optimization. arXiv:1412.6980v9. In Proc. 3rd International Conference on Learning Representations (ICLR 2015), pp. 1-15.

24. ImageNet (2020). Disponible on-line: http://www.image-net.org (Accedido Diciembre 2020).

25. ImageNet LSVRC (2012). Large Scale Visual Recognition Challenge 2012 (ILSVRC2012). Disponible on-line: http://www.image-net.org/challenges/LSVRC/2012/ (Accedido Diciembre 2020).

26. ImageNet LSVRC (2014). Large Scale Visual Recognition Challenge 2014 (ILSVRC2014). Disponible on-line: http://image-net.org/challenges/LSVRC/2014/eccv2014 (Accedido Diciembre 2020).

27. Krizhevsky, A., Sutskever, I., Hinton, G.E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. In Proc. 25th Int. Conf. on Neural Information Processing Systems (NIPS'12), vol. 1, pp. 1097-1105.

28. MongoDB (2020). Cross-platform document-oriented database program. Disponible on-line: https://www.mongodb.com/ (Accedido Diciembre 2020).

29. Muehlemann, A. (2019). TrainYourOwnYOLO: Building a Custom Object Detector from Scratch. Disponible on-line: https://github.com/AntonMu/TrainYourOwnYOLO (Accedido Diciembre 2020)

30. Node-RED (2020). Low-code programming for event-driven applications. Disponible on-line: https://nodered.org/ (Accedido Diciembre 2020).

31. Pajares, G., de la Cruz, J.M. (2007). Visión por Computador: imágenes digitales y aplicaciones. RA-MA, Madrid.

32. Pajares, G., Herrera, P.J., Besada, E. (2021). Aprendizaje Profundo. RC-Libros, Madrid.

33. PostgreSQL (2020). PostgreSQL: The World's Most Advanced Open Source Relational Database. Disponible on-line: https://www.postgresql.org/ (Accedido Diciembre 2020).

34. Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv:1804.02767v1.

35. Sanic (2020). Web framework that's written to go fast. Disponible on-line: https://sanic.readthedocs.io/en/latest/ (Accedido Diciembre 2020).

36. SQLAlchemy (2020). The Python SQL Toolkit and Object Relational Mapper. Disponible on-line: https://www.sqlalchemy.org/ (Accedido Diciembre 2020).

37. Tripadvisor (2020). World's largest travel platform. Browse hundreds of millions of traveler reviews and opinions. Disponible on-line: https://www.tripadvisor.es/ (Accedido Diciembre 2020)

38. VoTT (2020). Visual Object Tracking. Dsiponible on-line: https://www.jesusninoc.com/12/06/vott-visual-object-tagging-tool/ (Accedido Diciembre 2020).

39. Wang, W., Yang, J., Chen, M., Wang, P. (2019). A Light CNN for End-to-End Car License Plates Detection and Recognition. IEEE Access, vol.7, 173875-173883.

40. Yelp (2020). User Reviews and Recommendations of Best Restaurants, Shopping, Nightlife, Food, Entertainment, Things to Do, Services and More at *Yelp*. Disponible on-line: https://www.yelp.es/ (Accedido Diciembre 2020)

41. Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... & Xie, F. (2018). Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.*, *41*(4), 39-45.

42. Zhao, A., Zheng, P., Xu, S.T., Wu, X. (2019). Object Detection with Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems, 30(11), 3212-3232.

APÉNDICES

Apéndice A - Licencias del software

En el presente apéndice se enumeran las licencias del software utilizado para el desarrollo de este trabajo, todas ellas de uso libre con fines docentes.

Software	Licencia
MLflow	Apache License 2.0
Node-RED	Apache License 2.0
Eclipse Mosquitto	EPL/EDL
MongoDB	SSPL
Sanic Framework	MIT License
Docker	Apache License 2.0
VoTT	MIT License
TrainYourOwnYOLO	CC Attribution 4.0
keras-yolo3	MIT License
Tensorflow	Apache License 2.0
Imgaug	MIT License
JMeter	Apache License 2.0

A 1. Licencias del software

Apéndice B - Código fuente y *dataset* empleado

Se facilitan en este apéndice las URL de los recursos generados durante el desarrollo de este trabajo,

Recurso	URL
Código fuente	https://github.com/frburrue/tfm.git
Dataset	https://drive.google.com/drive/folders/14x4d6TDIGx6EICVAzrAbK-i86p64H8Ch?usp=sharing

B 1. Recursos generados

Código fuente

El código fuente queda liberado con licencia MIT. Se citan a continuación los recursos de mayor relevancia del repositorio de *GitHub* proporcionado.

Módulo IA

- Data augmentation.ipynb. Código fuente de la técnica de *data augmentation* empleada.
- Train.ipynb. Código fuente del entrenamiento del modelo.
- Evaluation.ipynb. Código fuente que permite la evaluación del modelo de detección de objetos.
- mlflow. Recursos que permiten el despliegue del contenedor Docker de MLflow UI.

Módulo Propietario

- mqtt_broker. Recursos que permiten la configuración *ad-hoc* del contenedor Docker que habilita al *broker* MQTT.

- node-red. Recursos que permiten el despliegue del contenedor Docker de Node-RED, así como la habilitación de todos los flujos del Módulo Propietario.

Módulo Consumidor

- standalone. Código fuente del Módulo Consumidor. Contiene también los recursos que permiten crear imágenes de Docker de esta aplicación.
    - detection. Código fuente de la fase de detección.
    - rekognition. Código fuente de la fase de reconocimiento.
    - processing. Código fuente de la fase de procesamiento.
    - app.py. Código fuente del servicio que atiende a las peticiones de los usuarios.
- Front-end.ipynb. Código fuente donde se ilustran las distintas fases de la aplicación que constituye al Módulo Consumidor.
- Mongo.ipynb. *Playground* donde se ilustra la recolección en el Módulo Consumidor de la información proporcionada a través del Módulo Propietario.
- MLflow.ipynb. *Playground* que permite descargar nuevas versiones del modelo de detección de objetos.
- Rendimiento.ipynb. Código fuente que permite la evaluación del rendimiento del servicio de información turística del Módulo Consumidor.

General

- docker-compose.yml. Configuración de docker-compose que permite el despliegue del Módulo Propietario y de la interfaz web de MLflow UI. Adicionalmente consta la configuración que hace posible el levantamiento del Módulo Consumidor.
- launch.sh. Script que ejecuta las instrucciones necesarias para habilitar todos los recursos que se contemplan en el fichero docker-compose.yml.

Dataset

En el fichero dataset.zip se encuentran dos carpetas train-validation y test que constituyen al *dataset* con el que se ha entrenado el modelo de detección de objetos y al propio que ha permitido realizar la evaluación sobre el mismo, respectivamente.

Apéndice C - Guías de despliegue

Este apéndice enumera las configuraciones necesarias que permiten desplegar los distintos componentes de la solución que este trabajo propone en los distintos entornos. Se recomienda tener como referencia a la Figura 3.12 e ir provisionando los servicios según se van enumerando en los siguientes apartados.

Amazon RDS

La base de datos Amazon RDS alojará la información que proporciona MLflow Tracking acerca del registro de cada uno de los entrenamientos que se lleven a cabo. Este registro contempla desde hiperparámetros seleccionados hasta métricas resultado de haber entrenado el modelo de detección de objetos.

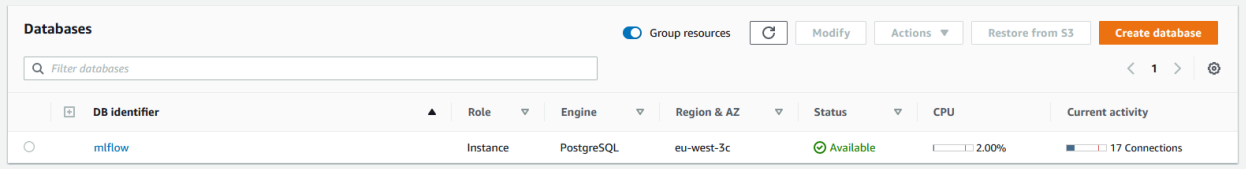
Este servicio estará a disposición de Google Colab, del Módulo IA, para este fin y del clúster de AWS Fargate, del Módulo Consumidor, donde se encontrarán los contenedores que requieren de esta información a través de MLflow Registry.

Para desplegar una instancia de Amazon RDS, desde la consola de administración de AWS seleccionamos el servicio RDS:

1. Creamos una instancia RDS.
2. Seleccionamos PostgreSQL. PostgreSQL es la base de datos recomendada por MLflow siendo v.11.x una versión idónea.
3. Nombramos a la instancia RDS (mlflow). El nombre DNS que apunte a la base de datos contendrá este sufijo.
4. Introducimos un nombre de usuario.
5. Seleccionamos una contraseña.

6. Elegimos tipo de instancia db.t3.small.
7. Creamos una base de datos con nombre mlflow.
8. Provisionamos 5gb de almacenamiento en un disco magnético.
9. Comprobamos la región donde se despliega para seguir desplegando servicios en la misma zona de disponibilidad.
10. Comprobamos que la base de datos ya se encuentra disponible.

Es preciso almacenar la DNS de la instancia de base de datos creada, el nombre de la base de datos, el usuario y la contraseña. Esta información será proporcionada al clúster de AWS Fargate y al entorno de ejecución de Google Colab.



C 1. Instancia RDS disponible

Amazon S3

Otro de los requisitos de MLflow Tracking es el de disponer de espacio de almacenamiento para guardar los artefactos. Dentro de la terminología de MLflow, los artefactos es todo aquel objeto, *dataset* o modelo entrenado, que han sido utilizados o generados durante la fase de entrenamiento del modelo de detección de objetos.

A fin de que estos artefactos, cuando son producidos por MLflow Tracking en Google Colab puedan ser consumidos por MLflow Registry, y por tanto por el clúster de AWS Fargate, es necesario disponer del espacio requerido en un entorno accesible por los distintos subsistemas.

Precisamos crear un *bucket* en S3 donde poder comenzar a almacenar estos objetos. Accedemos al servicio Amazon S3 a través de la consola de administración de AWS:

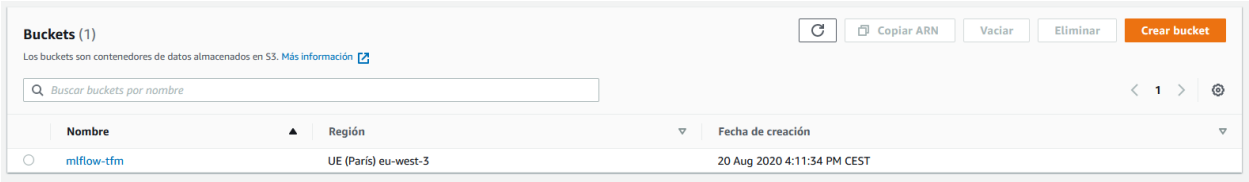
1. Creamos un *bucket*.

2. Seleccionamos un nombre para el *bucket* (mlflow-tfm). Este debe de ser único a nivel universal.
3. Escogemos la región en la que estamos trabajando.
4. Bloqueamos el acceso público a este *bucket*.

El acceso a este *bucket* se realizará siguiendo los siguientes procedimientos, dependiendo del subsistema:

1. Dentro de la plataforma AWS se otorgarán permisos para que Amazon EC2, que aloja el Módulo Propietario, y AWS Fargate, donde se ejecutan las réplicas de la aplicación que constituyen al Módulo Consumidor, puedan acceder a él en modo lectura/escritura.
2. Se utilizarán credenciales específicas de acceso a Amazon S3, con permisos de lectura/escritura, desde fuera del entorno de AWS, en particular desde Google Colab; desde el Módulo IA.

Es preciso almacenar tan solo la URI del *bucket* creado.



C 2. Bucket S3 disponible

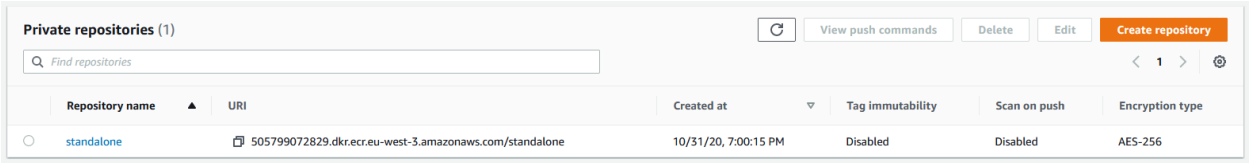
Amazon ECR

Después de desplegar el espacio de almacenamiento necesario, tanto en forma de base de datos relacional como en almacenamiento de objetos, requeridos para sincronizar al clúster de AWS Fargate con Google Colab, a través de los componentes de MLFlow, queda por poner a disposición de la plataforma un registro de contenedores. Este registro es el que permite publicar las últimas versiones de la aplicación que constituye al Módulo Consumidor para que sea replicada por todo el clúster de AWS Fargate.

Basta realizar la siguiente configuración dentro de la sección Amazon ECR de la consola de administración de AWS:

1. Creamos un repositorio.
2. Elegimos un nombre de repositorio (standalone).

Una vez creado el repositorio es necesario copiar las instrucciones de Docker, facilitadas por la sección Amazon ECR, que permiten publicar las imágenes en este registro.



C 3. Repositorio ECR disponible

Amazon EC2

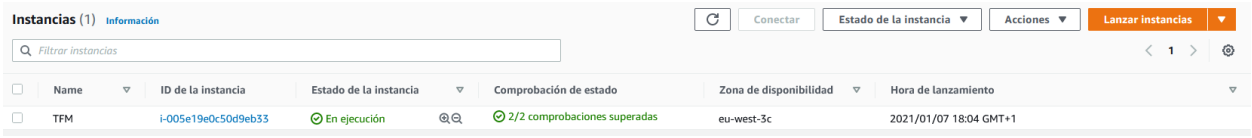
La máquina virtual Amazon EC2 se utilizará para albergar todos los procesos que constituyen el Módulo Propietario. Los procesos contenidos dentro de esta instancia se comunicarán con el resto de los componentes desplegados en AWS. Resulta entonces imprescindible dotar a la misma de los permisos requeridos y seleccionar el mismo entorno de red durante la configuración.

Desde la consola de administración seleccionamos el servicio EC2.

1. Creamos una instancia EC2.
2. Seleccionamos el sistema operativo. Ubuntu 20.04 LTS cumple con el requisito que nos permite poner en ejecución el software necesario para el propósito de este trabajo.
3. Denominamos a la instancia EC2 (TFM).
4. El tipo de instancia t3a.medium con 2 vCPUs y 4 gb de RAM soporta la ejecución simultánea de todo el software requerido.
5. Seleccionamos la misma zona de disponibilidad donde se encuentra albergada la base de datos Amazon RDS, el servicio Amazon S3 y el registro de contenedores Amazon ECR.

6. Precisamos de añadir a esta instancia permisos para acceder a Amazon S3, a Amazon RDS y a Amazon ECR. Seleccionamos para ello las políticas proporcionadas por AWS que hacen posible la interacción con estos servicios.
7. Provisionamos 30 gb de almacenamiento en un disco magnético.
8. Habilitamos la opción de encriptado de disco.
9. Habilitamos el tráfico en el puerto 22/tcp que nos permitirá entrar en la máquina virtual con el fin de instalar todo el software requerido y el puerto 80 para poder realizar la descarga de las dependencias necesarias. Adicionalmente ha de concederse el acceso a los puertos 60222/tcp (MongoDB), 60250/tcp (Flujo HTTP), 60251/tcp (Flujo MQTT) y 60260/tcp (MLflow UI)

Comprobamos que la máquina virtual se encuentra disponible y en la misma zona de disponibilidad que el resto de los servicios.



C 4. Instancia EC2 disponible

El script *launch.sh* que se ubica en la raíz del proyecto de *GitHub* facilitado, levanta todos los procesos necesarios para dejar en funcionamiento al Módulo Propietario. Es necesario almacenar el nombre DNS de esta máquina virtual, así como la relación de puertos abiertos con el prefijo 602.

AWS Fargate

El clúster de AWS Fargate ejecutará un número variable de contenedores definidos en imágenes Docker almacenadas en Amazon ECR que contendrán la última versión de la aplicación del Módulo Consumidor, que hará uso a su vez del servicio OCR de Amazon Rekognition.

Como ya se ha especificado, este módulo se conecta con Amazon RDS para obtener información de MLflow Registry con la cual descargar la última versión del modelo de detección de objetos disponible de Amazon S3; requerido por la aplicación.

Por otra parte, mantiene la conexión con la máquina virtual Amazon EC2 que contiene los procesos que constituyen al Módulo Propietario.

La configuración de AWS Fargate se divide en dos pasos. En primer lugar, es preciso crear una definición de tarea que viene a ser el entorno de ejecución de la aplicación y la definición de la aplicación en sí misma:

- Creamos una nueva definición de tarea.
- Seleccionamos un nombre (tfm-task).
- Concedemos permisos para acceder a otros subsistemas de AWS. Como se introducía en esta sección, se concederán permisos para acceder a Amazon ECR, Amazon Rekognition, Amazon RDS, Amazon S3 y a Amazon EC2. Seleccionamos para ello las políticas proporcionadas por AWS que hacen posible la interacción con estos servicios del mismo modo que se concedieron los permisos en la configuración de Amazon EC2.
- Especificamos cuál es la imagen de ECR de la cuál queremos ejecutar un contenedor.
- Indicamos cuántos recursos computacionales estarán reservados para el contenedor.
- Añadimos las variables de entorno necesarias.
- Configuramos la política de *healthcheck* con la ruta de la aplicación definida para ese fin.

Después de haber configurado la aplicación es turno de configurar el clúster que ejecutará un número variable de contenedores cuya definición de tarea ya hemos realizado:

1. Creamos un clúster.
2. Asignamos un nombre al clúster (tfm-cluster-spot).
3. Accedemos al clúster y creamos un servicio. El concepto de servicio engloba a la aplicación, en particular a una definición de tarea, así como a los siguientes parámetros que han de ser atendidos.

a. Seleccionamos recursos *on spot*.

b. Después de seleccionar la definición de tarea previamente creada y asignar un nombre al servicio (standalone) introducimos el número de réplicas de esta definición que van a ser ejecutadas en el clúster.

c. Establecemos que los contenedores se reinicien si hay una nueva versión de la imagen en Amazon ECR para que el clúster la descargue y comience a distribuirla.

d. Se debe conceder acceso al puerto 60210/tcp; puerto al que escucha el servicio de información turística.

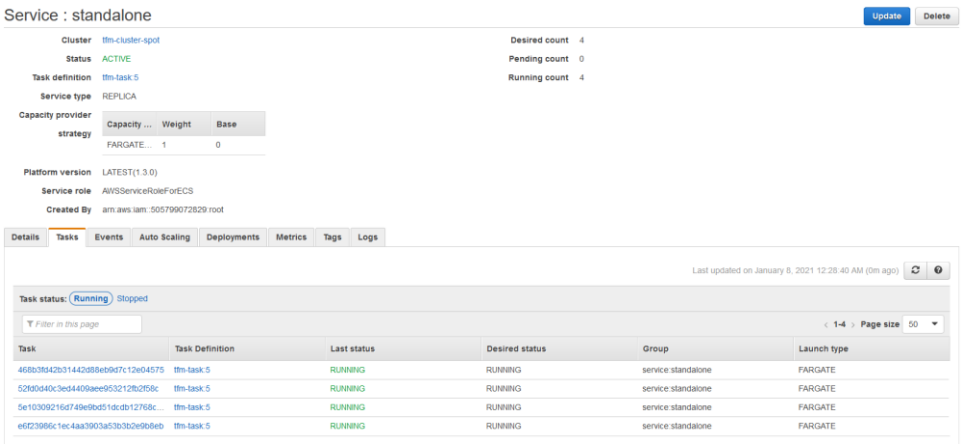
e. Habilitar un Application Load Balancer concediendo el acceso al puerto 80/tcp para hacer después port-forward al puerto 60210/tcp del pool de réplicas.

4. Configuramos políticas de *autoscaling*.

a. Introducimos un número de réplicas mínimo y otro máximo.

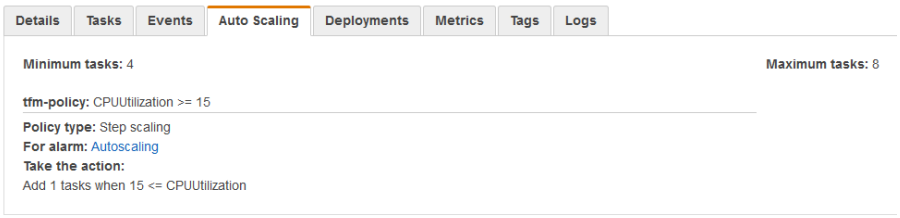
b. Establecemos la política de *autoscaling* en función del uso de vCPU o memoria RAM. Esta define cuántas réplicas han de ser dispuestas en qué circunstancias y cuándo debe producirse la interrupción de aquellas habilitadas de forma adicional a las configuradas por defecto.

Puede verse la definición del servicio standalone que ejecuta 4 réplicas de la tarea tfm-task dentro del clúster tfm-cluster-spot.



C 5. Clúster habilitado

Adicionalmente, en la pestaña *Auto Scaling* podemos ver la política de *autoscaling* configurada. En este caso se añadirá una réplica adicional, hasta un máximo de 8, cuando el porcentaje de utilización de todas ellas supere o iguale un 15% de media durante 1 minuto. Toda aquella réplica adicional será suspendida a los 5 minutos de no haberse presentado nuevamente esta circunstancia.



C 6. Integración de política de autoscaling

Google Colab + Google Drive

El notebook *Train.ipynb* contiene el código fuente necesario que permite conectar al entorno de entrenamiento, al Módulo IA, con los distintos componentes de MLflow; cuyas dependencias son los servicios Amazon RDS y Amazon S3. Este recurso puede encontrarse en la raíz del proyecto de *GitHub*.

Apéndice D - Guías de usuario

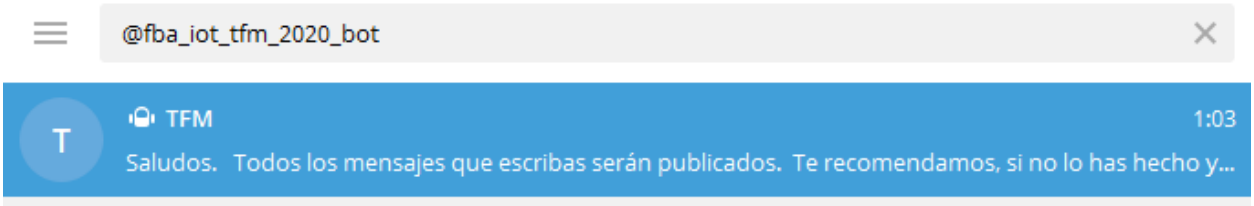
Este apéndice pretende ilustrar a los usuarios del Módulo Propietario y del Módulo Consumidor cómo interactuar con cada uno de los servicios publicados a los que se hace mención en este documento.

Módulo Propietario

Este módulo habilita distintos canales por los cuales es posible recopilar información de interés turístico para ser proporcionada a través del Módulo Consumidor.

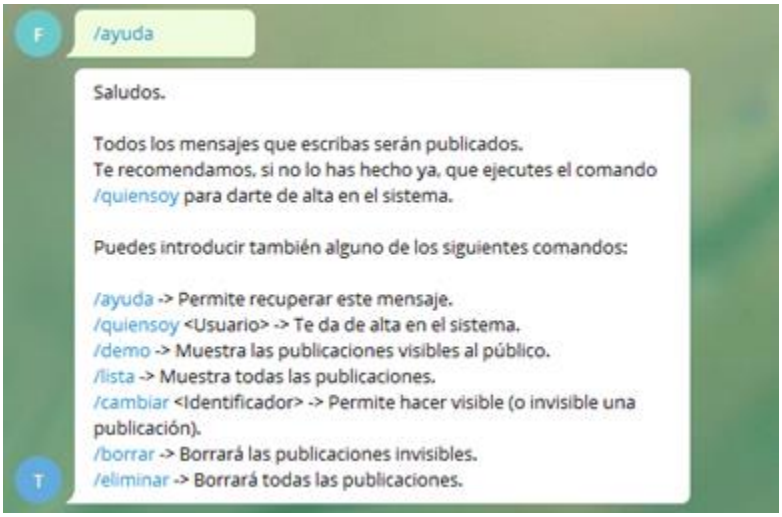
Canal Telegram

Desde la aplicación de escritorio o de *smartphone*, es preciso iniciar una conversación con el *bot* @fba_iot_tfm_2020_bot.



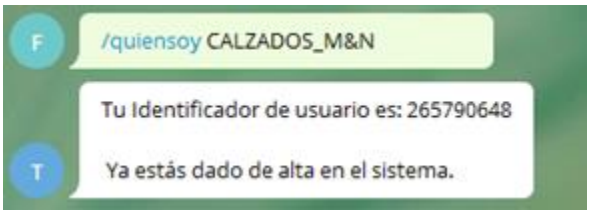
D 1. Pantalla de búsqueda de bot

Se recomienda encarecidamente utilizar el comando /ayuda tal y como se ilustra en la siguiente imagen si es la primera vez que se interactúa con el *bot*.



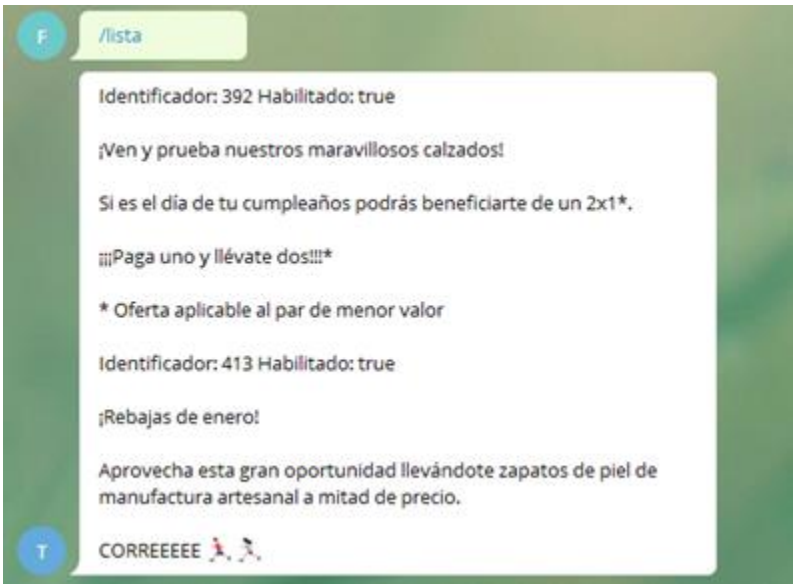
D 2. Pantalla de ayuda

Como reza la respuesta al comando /ayuda, es necesario utilizar el comando /quiensoy <Usuario> para darse de alta en el sistema. <Usuario> debe ser el nombre del servicio turístico donde deben sustituirse los espacios por el símbolo (_); aquel que aparece en el panel que lo identificará inequívocamente sobre el resto. El *bot* concederá un identificador de usuario único que deberá ser utilizado en todas las peticiones que se realicen mediante el canal HTTP como se ilustrará más adelante.



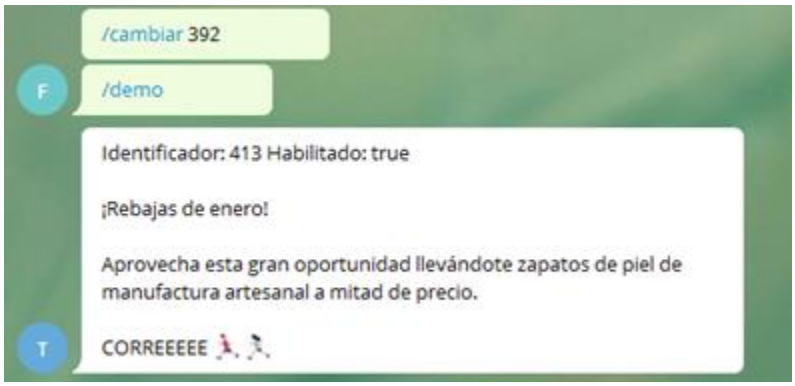
D 3. Pantalla de alta

Cualquier mensaje introducido que no contenga ninguno de los comandos que aparecen en la imagen se tomará como información turística que ha de ser publicada por defecto. Con el comando /lista podemos ver la totalidad de esta información previamente almacenada.



D 4. Pantalla con el listado de publicaciones

Todas las publicaciones se encuentran en uno de los dos estados posibles, habilitadas o inhabilitadas al público. El comando /demo muestra tan solo aquellas habilitadas al público; si se desea dejar de mostrar alguna, o comenzar a mostrarla, solo es necesario introducir el comando /cambiar <Identificador> para cambiar su estado donde <Identificador> es el número que se encuentra por encima de cada mensaje después de utilizar el comando /lista o /demo, indistintamente.



D 5. Pantalla de cambio de estado y publicaciones habilitadas

Los comandos /borrar y /eliminar, tal y como se especifica en la ayuda, eliminan definitivamente los mensajes inhabilitados o la totalidad almacenada, respectivamente.

Canal HTTP

Paso previo a la utilización de este canal es necesario obtener un identificador de usuario único a través del Canal Telegram tal y como allí de describe. Ese identificador debe ser proporcionado en todas las interacciones con este canal, exceptuando el servicio de ayuda.

Se presentan a continuación todos los servicios habilitados, así como el modelo de datos requerido para consumir cada uno de ellos:

- **GET** http://<host>:60250/ayuda: Devuelve la información presente en este apartado.
    - Entrada: No aplica.
    - Salida: La información presente en este apartado.


```
{
    "content": "\n/ayuda -> Permite recuperar este mensaje.\n\n/publish -> Permite publicar un anuncio.\n\n{\"chatId\": <Identificador de usuario>,\"content\": <Anuncio>}\n\n/lista -> Muestra todas las publicaciones.\n\n
        {\"chatId\": <Identificador único de usuario>}\n\n/cambiar -> Permite hacer visible (o invisible una publicación).\n\n{\"chatId\": <Identificador único de usuario>, \"messageId\": <Identificador de mensaje obtenido a través
        del servicio /lista>}\n\n/borrar -> Borrará las publicaciones invisibles.\n\n{\"chatId\": <Identificador único de usuario>}\n\n/eliminar -> Borrará todas las publicaciones.\n\n{\"chatId\": <Identificador único de usuario>}
        \n"
}
```


D 6. Respuesta servicio de ayuda

- **POST** http://<host>:60250/publish: Permite publicar información de interés turístico.
    - Entrada: `{"id": <Identificador único de usuario>,"content": <Información>}`
    - Salida: Respuesta en formato JSON.


```
{
    "chatId": 265790648,
    "content": "Guardado!",
    "messageId": "414",
    "show": true,
    "date": 1610072568
}
```


D 7. Respuesta servicio de publicación

- **POST** http://<host>:60250/lista: Muestra todas las publicaciones; habilitadas o no.
    - Entrada: `{"id": <Identificador único de usuario>}`

- Salida: Respuesta en formato JSON.


```
[
    {
        "_id": "5fdb85bf4ed3af001076a8a6",
        "chatId": 265790648,
        "messageId": 392,
        "type": "message",
        "content": "¡Ven y prueba nuestros maravillosos calzados!\n\nSi es el día de tu cumpleaños podrás beneficiarte de un 2x1*.\n\n¡¡Paga uno y llévate dos!!!*\n\n* Oferta aplicable al par de menor valor",
        "date": 1608222143,
        "show": false
    },
    {
        "_id": "5ff7a58d5f50310010f523b0",
        "chatId": 265790648,
        "messageId": 413,
        "type": "message",
        "content": "¡Rebajas de enero!\n\nAprovecha esta gran oportunidad llevándote zapatos de piel de manufactura artesanal a mitad de precio.\n\nCORREEEEE 🏃‍♀️🏃‍♀️",
        "date": 1610065293,
        "show": true
    },
    {
        "_id": "5ff7c1f878cb000010426548",
        "chatId": 265790648,
        "content": "El tiempo estimado de espera en este establecimiento es de 10 minutos",
        "messageId": "414",
        "show": true,
        "date": 1610072568
    }
]
```


D 8. Respuesta servicio de listado de publicaciones

- **PUT** http://<host>:60250/cambiar: Permite habilitar o inhabilitar una publicación en función de su estado.
    - Entrada: `{"id": <Identificador único de usuario>, "messageId": <Identificador de mensaje obtenido a través del servicio /lista>}`
    - Salida: Respuesta en formato JSON.


```
[
    {
        "_id": "5ff7c1f878cb000010426548",
        "chatId": 265790648,
        "content": "El tiempo estimado de espera en este establecimiento es de 10 minutos",
        "messageId": "414",
        "show": true,
        "date": 1610072568
    },
    {
        "$set": {
            "show": false
        }
    }
]
```


D 9. Respuesta servicio de cambio de estado

- **DELETE** http://<host>:60250/borrar: Borrará las publicaciones inhabilitadas.
    - Entrada: `{"chatId": <Identificador único de usuario>}`
    - Salida: Por seguridad, el método DELETE no devuelve ningún resultado.
- **DELETE** http://<host>:60250/eliminar: Borrará todas las publicaciones, habilitadas o no.
    - Entrada: `{"chatId": <Identificador único de usuario>}`

- Salida: Por seguridad, el método DELETE no devuelve ningún resultado.

Canal MQTT

La solución habilita un broker MQTT en el puerto 60251/tcp. Adicionalmente, ejecuta un proceso que queda suscrito al topic *sensor* el cuál procesa información en formato JSON atendiendo a esta estructura:

```
{"chatId":<Identificador único de usuario>, "content":<Información de sensores en formato JSON>}
```

D 10. Ejemplo de publicación

Se almacenarán todas las claves de primer nivel relativas a la información de sensores proporcionada junto con los respectivos valores asociados. Mensajes consecutivos que compartan estas claves sobrescribirán los valores almacenados a ellas.

Canal Twitter

Desde la aplicación de escritorio o de *smartphone*, tan solo es preciso escribir un *tweet* con dos hashtags, seguidos al principio del mensaje, siendo el primero #fba_iot_tfm y el segundo el nombre del servicio turístico sobre el que se desea mostrar una opinión donde deben sustituirse los espacios por el símbolo (_) y el símbolo (&) por la cadena (ând) tal y como como se ilustra en la imagen. A continuación de estas dos etiquetas, el usuario puede escribir el mensaje que considere oportuno atendiendo a las restricciones impuestas por Twitter atendiendo a sus términos y condiciones.

D 11. Pantalla de publicación de opinión

La publicación de una opinión a través de este canal es irreversible. Es decir, no hay opción para que los usuarios de los servicios turísticos puedan modificarlas o eliminarlas.

Módulo Consumidor

El notebook *Front-end.ipynb,* recurso que puede encontrarse en la raíz del proyecto de *GitHub*, pretende ilustrar cuáles son los procedimientos necesarios para consumir el servicio de información turística así como para tratar la respuesta del mismo.

Adicionalmente, se habilitan dos servicios:

- **POST** http://<host>:60210/update: Comprueba la última versión del modelo de detección de objetos publicada y descarga una nueva versión si esta es posterior a la registrada en la última comprobación.
    - Entrada: No aplica.
    - Salida: Respuesta en formato JSON. El valor booleano dentro de la clave response/Hands indica si se ha descargado una nueva versión del modelo.

D 12. Respuesta servicio de actualización del modelo

- **GET** http://<host>:60210/healthcheck: Este es un requisito de AWS Fargate para comprobar si el servicio se encuentra activo a fin de reiniciarlo si fuese necesario.
    - Entrada: No aplica.
    - Salida: No devuelve contenido. Es preciso comprobar el código HTTP de la respuesta, siendo el código 200 el resultado que se espera si el servicio funciona correctamente.