Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Integración en ROS de herramientas de Web Scraping para análisis de vídeo

Web Scraping tools ROS integration for Video Analysis

Daniel Antonio Fernández Pérez

La Laguna, *15* de *Marzo* de 2019

D. **Néstor Morales Hernández** , con N.I.F. 54048001W profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Integración ROS con Web Scraping”

ha sido realizada bajo su dirección por D. **Daniel Antonio Fernández Pérez**,

con N.I.F. 79074318L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 21 de noviembre de 2018

Agradecimientos

Agradecimientos a mi tutor Néstor Morales Hernández por la ayuda durante todo el proceso de desarrollo del Trabajo de Fin de Grado, así como por haberme permitido extender el trabajo más allá de las ideas iniciales, introduciendo conceptos como las redes neuronales. También agradecer su colaboración con el desarrollo de esta memoria del proyecto.

Licencia

Resumen

Este TFG se centra en el desarrollo de un sistema de detección de personas listo para usar en distintas aplicaciones, como robótica o videovigilancia. El sistema es compatible con diferentes entradas de vídeo, incluyendo cámaras de vídeo, streaming e incluso información extraída en tiempo real de la web mediante técnicas de webscraping. Para ello se han usado herramientas y librerías como OpenCV, combinado con técnicas de Inteligencia Artificial.

El proyecto ha sido dividido en diferentes módulos, identificándose cada uno como nodos de ROS.

El primer módulo desarrollado es el de web scraping, mediante el uso de la herramienta Selenium, la cual ha permitido captar, abriendo el navegador Firefox en segundo plano, vídeos en directo de la plataforma Youtube, y enviar los mismos a través de un ROS Topic.

El segundo módulo implementado ha sido el de captar la entrada de la webcam del usuario y enviarla de la misma forma, mediante un Topic.

El tercer módulo, capta los outputs de los dos anteriores, y los procesa mediante el uso de una red neuronal, la cual es capaz de parametrizar objetos dentro de imágenes. Esta detección en tiempo real, se envía como salida para su visualización en otros nodos. Además, este módulo inicia una grabación de los inputs una vez detectado los objetivos, incluyendo entre otras funcionalidades, la de alertar al usuario de dichas detecciones, guardando la información en una base de datos local.

El último módulo, se encarga de visualizar los resultados obtenidos, usando un servidor local Apache, en el cual, entre otras funcionalidades, se puede visualizar los diferentes métodos de captación junto a su procesamiento.Todos estos módulos situados dentro de un package ROS, crean un sistema eficiente, portable y reusable de detección, ya que, los distintos nodos comprendidos en este proyecto se pueden reutilizar para otros.

Palabras clave: ROS, webscraping, OpenCV, Selenium, detección, Apache, Inteligencia Artificial.

Abstract

This TFG focuses on the development of a detection system for people ready to be used in different applications, such as robotics or video surveillance. The system supports different video inputs, including video cameras, streaming and even information extracted in real time from the web using webscraping techniques. To this end, tools and libraries such as OpenCV have been used, combined with Artificial Intelligence techniques.
The project has been divided into different modules, identifying each one as ROS nodes.
The first module developed is the web scraping one, through the use of the Selenium tool, which has allowed capturing, opening the Firefox browser in the background, live videos from the YouTube platform, and sending them through a ROS Topic .
The second module implemented is intended to capture the entry of the user's webcam and send it in the same way, using a Topic.
The third module captures the outputs of the previous two, and processes them through the use of a neural network, which is able to parameterize objects within images. This real-time detection is sent as an output for its visualization in other nodes. In addition, this module initiates the recording of the inputs once the objectives are detected, including among other functionalities, that of alerting the user of such detections, saving the information in a local database.
The last module is responsible for visualizing the results obtained, using a local Apache server, in which, among other functions, it is possible to visualize the different capture methods together with their processing. All these modules located within a ROS package, create an efficient, portable and reusable detection system, since the different nodes included in this project can be reused for others.

Keywords: ROS, webscraping, OpenCV, Selenium, detection, Apache, Artificial Intelligence

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Introducción

El objetivo de este TFG es elaborar un sistema eficiente y portable de detección de personas, definiendo diferentes entradas de datos, las cuales, pueden ser desde una webcam a un video en streaming. Mediante el uso del framework ROS y herramientas como Selenium.

Las principales aportaciones de este proyecto son las siguientes:

- Creación de un nodo capaz de extraer información de vídeo en tiempo real de una página web como Youtube.
- Creación de un nodo de ROS capaz de detectar personas con una alta precisión.
- Creación de un sistema modular de procesamiento de imágenes, compatible con diversas entradas.
- Aplicación de dicho sistema al caso de uso de un sistema de control de presencia.

1.2 Estructura

Este documento se divide de la siguiente manera:

- En la sección “Antecedentes y estado actual” se describirán las tecnologías usadas. Además, se introducirá a las redes neuronales artificiales y los algoritmos más usados en las mismas, así como los frameworks más utilizados.
- En la sección “Objetivo del trabajo” y “Metodología” se entrará más en profundidad en conocer el funcionamiento del proyecto y de todas sus funcionalidades, entre todo esto, los pasos necesarios para su correcta ejecución, así como ejemplos prácticos en los que se detalla el rendimiento del sistema, tanto para Selenium como la entrada de cámara web. Además, se describe la interfaz de usuario, la cual será utilizada para notificar las alertas que el sistema genere, esto implica tanto la interfaz web como la de mensajería instantánea, Telegram.
- En la sección “conclusiones y líneas futuras”, se propondrán mejoras al sistema actualmente desarrollado, incluyendo supuestos prácticos del mismo, en el cual se detalla presupuestos necesarios para su óptimo funcionamiento.
- En la sección de “Presupuesto”, se detalla el coste aproximado de llevar el proyecto a producción.

Capítulo 2 Antecedentes y estado actual

2.1 Historia

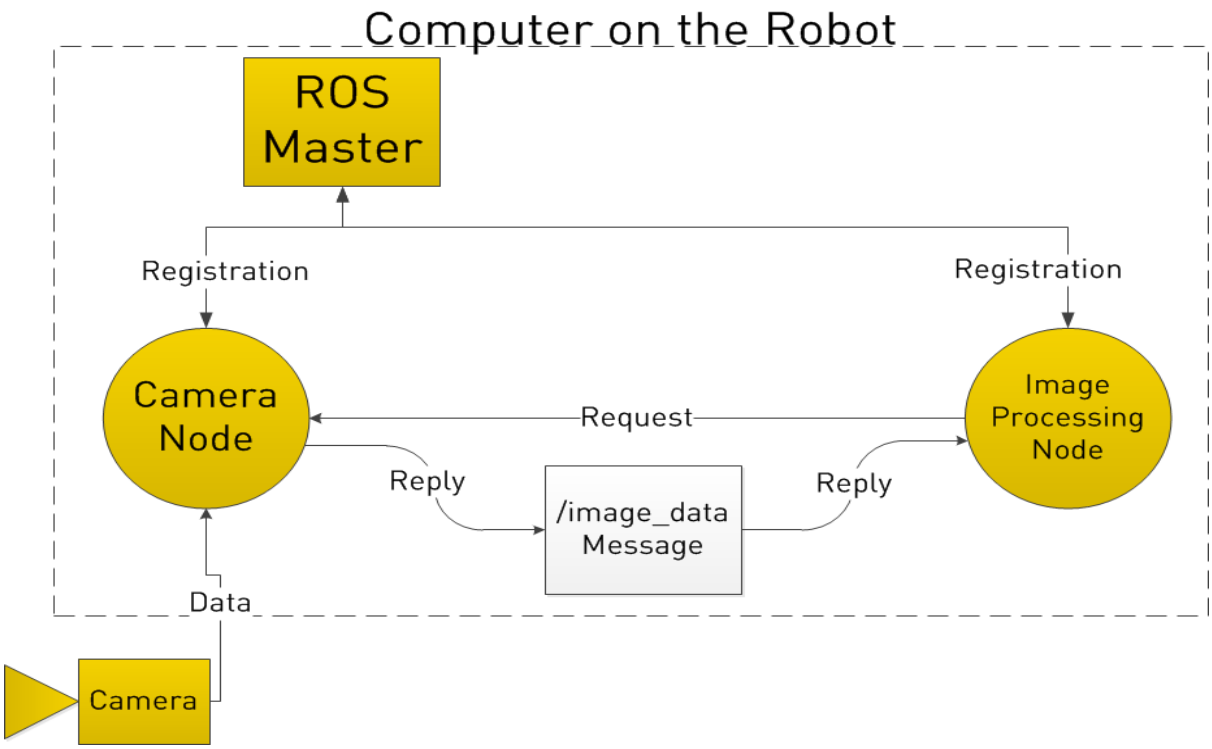
La detección de personas en videos, ha sido durante años el centro de muchas investigaciones, así como el desarrollo de sistemas que permitieran integrar dichas detecciones en los diferentes dispositivos inteligentes existentes en la actualidad. Hoy en día, existen sistemas bastante eficientes para llevar a cabo proyectos de detección de elevada exactitud, pudiendo diferenciar objetos a poca resolución. Dentro de estos sistemas, se han usado para el desarrollo de este proyecto, frameworks y herramientas de actualidad, así como conceptos avanzados de Inteligencia Artificial aplicados al tratamiento de imágenes

2.2 ROS

ROS (Robot Operating System) es un framework libre para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. En términos coloquiales, ROS es definido como un Sistema Operativo enfocado a la robótica, pudiendo crear proyectos escalables.

La principal estructura de ROS se divide entre nodos y tópicos. Los nodos son básicamente archivos ejecutables que realizan alguna funcionalidad concreta, dentro de un package. Los tópicos, son los canales de comunicación por los cuales intercambian información los diferentes nodos, de esta forma, un nodo puede conectarse con otro pudiendo así reutilizar funcionalidades.

ROS está basado en la arquitectura de grafos, y hoy en día es muy utilizado para el desarrollo de todo tipo de sistemas robóticos, tanto de uso privado como de uso comercial. Posee diferentes versiones, así como un gran repertorio de extensiones la cual permite su implementación junto a frameworks tan populares como OpenCV. Además, el proyecto ROS-Industrial permite ampliar las capacidades de estas versiones de ROS.



Ejemplo de framework ROS para un robot.

Es posible encontrar más información sobre ROS en https://www.ros.org.

2.3 Selenium

Antes de la definición de Selenium, hay que indagar en la definición de “web scraping”. Este término no es otro que la extracción de información de páginas webs de forma automatizada. Su principal aplicación es la de conseguir cantidades industriales de información, mediante algoritmos de búsqueda los cuales pueden rastrear centenares de webs.

Selenium es un entorno de pruebas de software para aplicaciones basadas en la web, el cual provee de una herramienta de grabar / reproducir para crear pruebas sin usar un lenguaje de scripting.

Dentro de Selenium, encontramos Selenium WebDriver, el cual permite abrir un navegador e interactuar con él mediante comandos, de esta forma, mediante el uso de lenguajes de programación como python, se pueden realizar scripts que permitan extraer información de diferentes webs de forma automatizada. Básicamente, WebDriver crea una instancia de un navegador concreto, y lo controla, haciendo a Selenium una potente herramienta de QA (Quality Assurance).

En la web, podemos encontrar numerosos ejemplos de Selenium para ejecutar pruebas de software, en diferentes lenguajes de programación tales como Ruby.

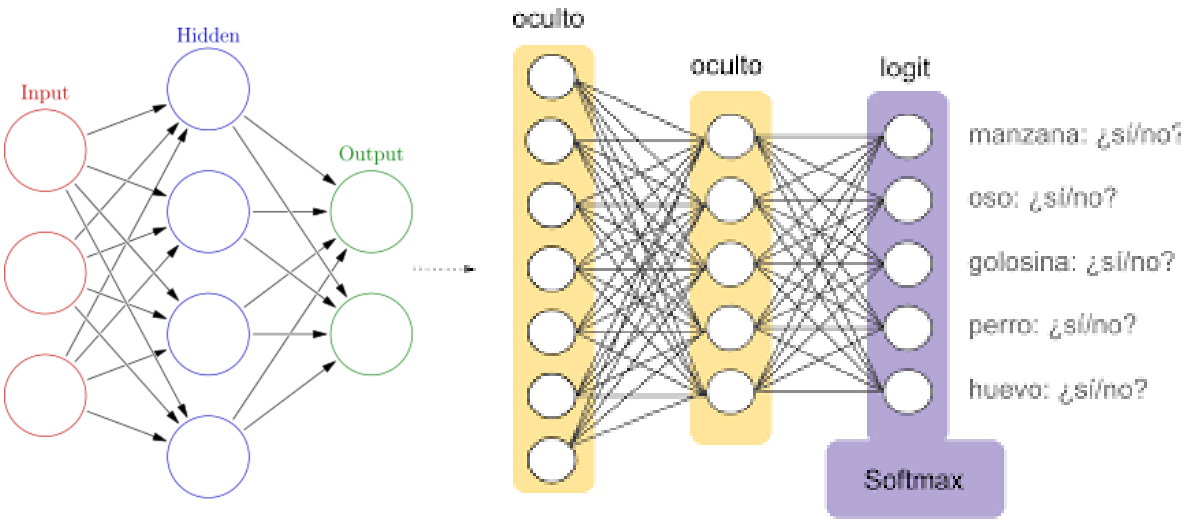
Es posible encontrar más información sobre Selenium en https://www.seleniumhq.org/.

2.4 Redes neuronales artificiales

Las redes neuronales artificiales son un modelo computacional vagamente inspirado en el comportamiento observado en su homólogo biológico. Consiste en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitirse señales. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida.

Las redes neuronales pueden están constituidas por una capa de input y otra de output, pero, entre estas dos, pueden haber numerosas capas denominadas “hidden layout”.

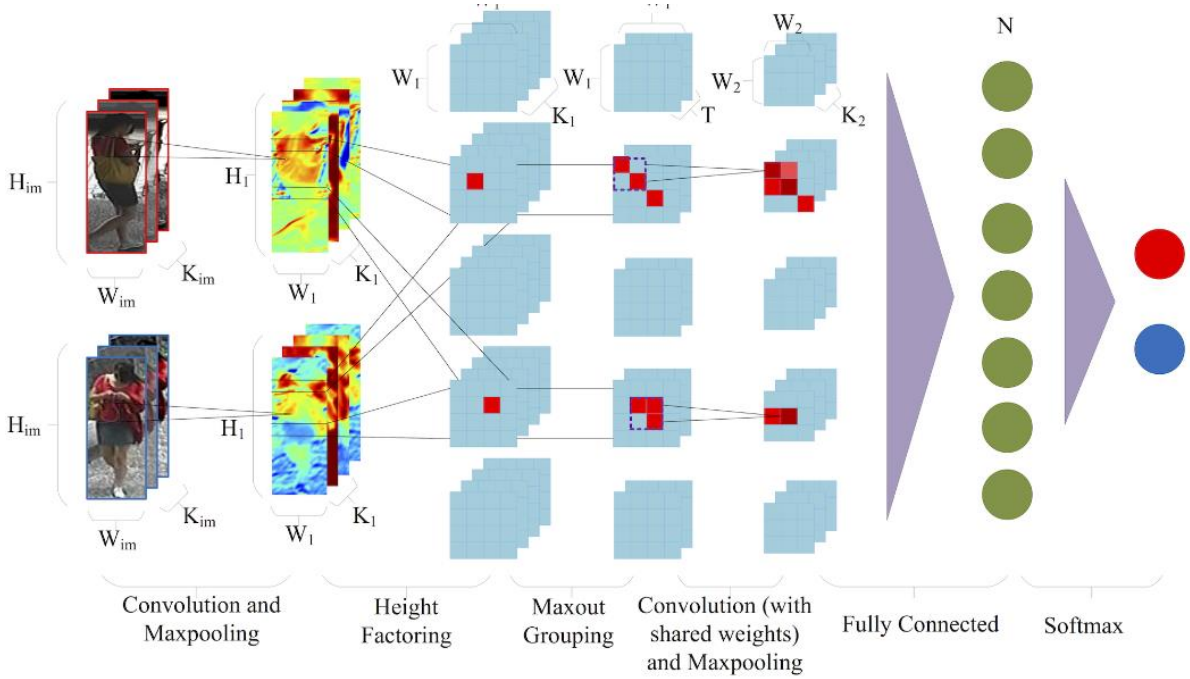
Cada neurona artificial es capaz de producir un output a partir de funciones matemáticas, y siempre teniendo en cuenta una función de activación la cual siempre intenta minimizar la función de pérdida. En su conjunto, varias combinaciones de neuronas conectadas entre sí, funcionan como puertas lógicas, siendo el peso que tengan establecidos en el enlace lo que modifique dichos resultados.
Justo antes de la capa final de resultados, se implementan funciones matemáticas como la Softmax, que traducen dichos resultados en, por ejemplo probabilidades para diferentes clases en nuestro problema a resolver.



Ejemplo de red neuronal con output Softmax.

Enfocándonos en el tratamiento de imágenes, las redes neuronales aprovechan grandes volúmenes de imágenes obtenidas a partir de datasets, de forma que se van entrenando a partir de dichas imágenes, acotando el margen de error y minimizando dicha función de pérdida comentada anteriormente. Para esto, existen diversos métodos de optimización, siendo el Gradient Descent uno de los más conocidos.
Las imágenes son tratadas atendiendo a sus pixeles, los cuales forman matrices de datos. Dichas matrices se pasan entre las diferentes capas de la red, de forma que, cuando más

adelante se pase una imagen que atienda a unos ciertos valores, la red sea capaz de diferenciar elementos dentro de la misma.

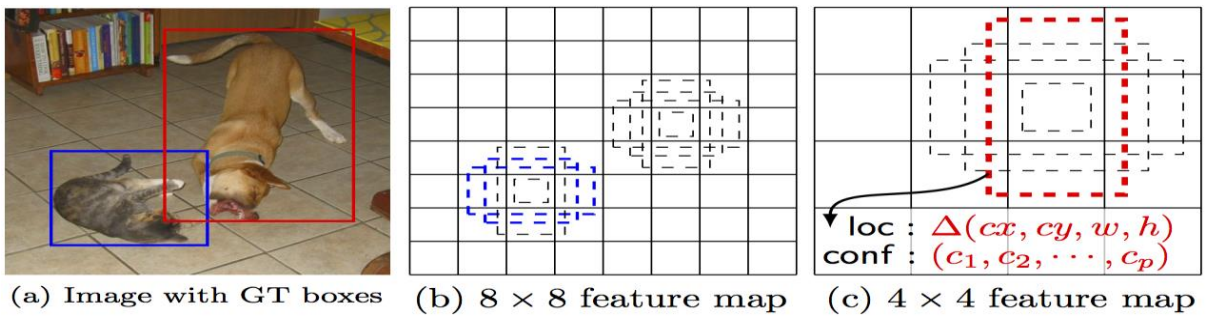


Ejemplo de red neuronal con output Softmax para tratamiento de imágenes.

Enfocándonos en el tratamiento de imágenes, las redes neuronales aprovechan grandes volúmenes de imágenes obtenidas a partir de datasets, de forma que se van entrenando a partir de dichas imágenes, acotando el margen de error y minimizando dicha función de pérdida comentada anteriormente. Para esto, existen diversos métodos de optimización, siendo el Gradient Descent uno de los más conocidos.
Las imágenes son tratadas atendiendo a sus pixeles, los cuales forman matrices de datos. Dichas matrices se pasan entre las diferentes capas de la red, de forma que, cuando más adelante se pase una imagen que atienda a unos ciertos valores, la red sea capaz de diferenciar elementos dentro de la misma.

Dentro de las redes neuronales hay varios tipos, pero en este proyecto nos centraremos en definir qué es la MobileNetSSD, la cual usaremos para la detección de personas.

En primer lugar, SSD (Single Shot MultiBox Detector) es un algoritmo que, sin entrar en complicaciones matemáticas, permite con una única red neuronal, diferenciar dentro de una misma imagen varios elementos (multibox) y su ubicación.



Ejemplo algoritmo SSD en imagen real.

El concepto de MobileNet, se aplica a redes neuronales adaptadas para dispositivos móviles, es decir, comprimidas para ocupar el menor espacio posible y consumir los mínimos recursos necesarios para obtener resultados eficientes.

Por lo que, la MobileNetSSD es un proyecto que a unificado estos dos conceptos para crear un framework ligero y eficiente de detección de múltiples objetos en imágenes.

Respecto a las API's existentes hoy en día para la implementación de redes neuronales, tenemos entre las más destacadas:

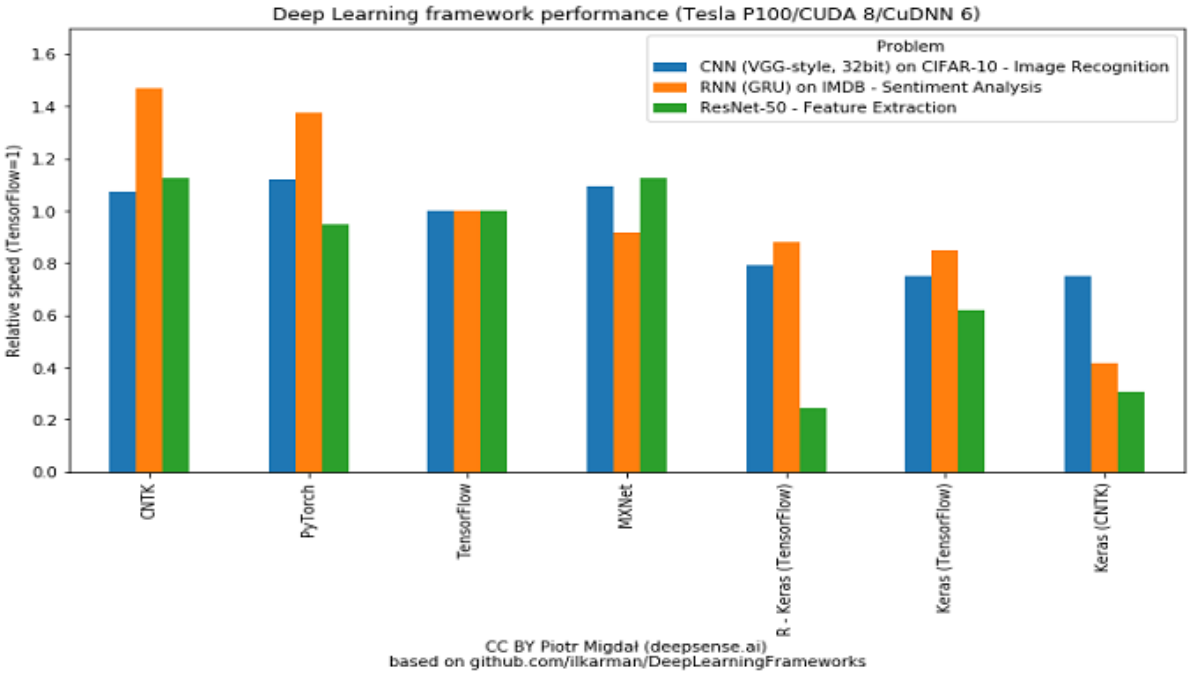
-Tensorflow (https://www.tensorflow.org/), es una API open source desarrollada por Google Brain. Es muy utilizada a nivel comercial debido a la potencia y la cantidad de funcionalidades que posee, además, con librerías como TFlearn se puede simplificar drásticamente las operaciones y el código necesario para el desarrollo de tus propias redes.

-Pytorch (https://pytorch.org/), es una API open source desarrollada por Facebook. Respecto a la anterior, es más utilizada a nivel de investigación o proyectos personales, ya que el framework es más sencillo e intuitivo, y no necesita un gran aprendizaje como Tensorflow.

-Caffe (http://caffe.berkeleyvision.org/), es una API open source desarrollada por la Universidad de California, y es también alternativa a las otras nombradas.

Todas estas API's permiten integración con librerías como NVIDIA cuDNN, que básicamente permiten usar la potencia de las tarjetas gráficas para el entrenamiento de las redes neuronales, ya que las GPUs están diseñadas para procesos matriciales en paralelo, por lo que son idóneas para este trabajo.

A continuación vemos una gráfica que muestra una comparativa aproximada de velocidad para, en este caso, la CNN, RNN y ResNet50, entre las diferentes API's existentes.



Comparativa Deep Learning Frameworks

2.5 Reconocimiento de regiones

En el campo de la visión artificial, el reconocimiento de regiones se refiere a las técnicas cuyo objetivo es detectar puntos o regiones más claras o más oscuras de la imagen. Hay dos clases principales de detectores de regiones (i) métodos diferenciales y (ii) métodos basados en extremos locales. Estos detectores también se denominan detectores de puntos interesantes, o detectores de regiones interesantes.

El estudio y desarrollo de estos detectores es importante por varias razones. La principal es dar información complementaria sobre regiones que no se puede obtener mediante detectores de bordes o detectores de esquinas. Los detectores de regiones se usan como paso previo para el reconocimiento de objetos o seguimiento de objetos. Otro uso habitual de estos detectores tiene que ver con el análisis de texturas y su reconocimiento. Recientemente, los descriptores de regiones han empezado a usarse para puntos de interés para informar de la presencia de determinados objetos en una imagen.

Estas técnicas, en combinación con otras, tienen ya aplicaciones de uso más cotidiano: por ejemplo para software de dispositivos táctiles, funciones de detección de rostros y sonrisas en cámaras de fotos, sistemas de vigilancia y seguridad, o para analizar imágenes médicas

Imagen donde se puede observar en combinación a otras técnicas, el uso del reconocimiento de regiones.

Capítulo3 Objetivos del trabajo

3.1 Objetivos

El objetivo de este trabajo es el desarrollo de un sistema capaz de detectar personas en un stream de video en tiempo real, y de actuar en consecuencia. El sistema está sustentado en ROS, permitiendo ser reaprovechado para otras tareas. Además, el módulo de entrada basado en Selenium permite capturar vídeo de cualquier plataforma basada en tecnología web.

Inicialmente, el trabajo se centraba en montar toda la estructura ROS y posteriormente, mediante alguna tecnología de tratamiento de imágenes, procesar la entrada y detectar a las personas de un streaming.

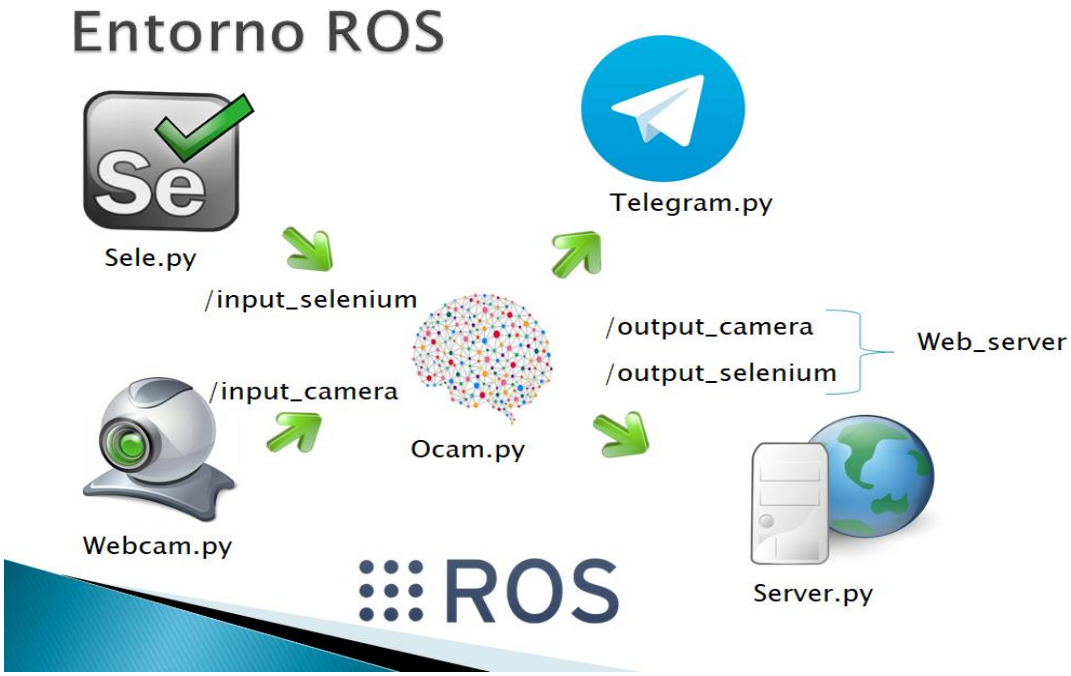
A medida que se realizó la investigación, el trabajo pasó a usar redes neuronales como base para el tratamiento de imágenes, ya que, estas están teniendo un gran auge en la actualidad. De esta forma, el objetivo se centró en encontrar o construir una red que tuviera el rendimiento necesario para tratar las entradas de video, principalmente entradas por streaming web. Posteriormente, dadas las características de ROS se planteó el uso de otras entradas de video, aunque debido a la falta de recursos hardware, se acabó usando la entrada estándar de video por webcam como prueba de concepto. Sin embargo, el sistema debería de ser compatible con todo tipo de entradas de vídeo compatibles directa o indirectamente con ROS, incluyendo cámaras IPTV, cámaras acopladas a un robot, drones, etc.; o incluso, gracias al uso de Selenium, sistemas de control de cámaras basadas en software privativo que haga uso de tecnología web.

Finalmente,una vez montada la estructura ROS y la red neuronal, el objetivo del trabajo se centró en enfocar dichas tecnologías en una aplicación eficiente. Para ello, se pasó al desarrollo de una interfaz de usuario interactiva en la cual un usuario pudiera realizar un seguimiento de las personas que entran a un determinado lugar, así como de un sistema de alertas, ya sea vía web o Telegram.

Capítulo 4 Metodología

4.1 Descripción del sistema

El sistema se describe a alto nivel en la siguiente imagen:



Representación de la estructura ROS del proyecto.

Dentro del paquete creado tenemos los siguientes nodos de ROS:

1. sele.py, este nodo escrito en Python se encarga de mediante Selenium, abrir un navegador Firefox usando el driver Gecko y extraer el video en directo de una determinada URL. Posteriormente, dicho video es enviado por frames mediante un Topic ROS. El navegador Firefox puede ser lanzado en background, de forma que su uso sea opaco de cara al usuario.
2. webcam.py, este nodo escrito en Python se encarga mediante OpenCV de abrir la webcam del ordenador host, de forma que dichos frames son enviados por un Topic ROS.
3. Ocam.py, este nodo escrito en Python se encarga, mediante OpenCV y una red neuronal implementada en Caffe, de realizar la detección de las personas a partir de un input enviado por los nodos nombrados anteriormente. También, este fichero

se encarga de conectarse con la base de datos MySQL para ir guardando la información relativa a los avisos. Este módulo produce una salida, en la cual veremos las personas detectadas bien diferenciadas, así como la fiabilidad de la detección, mostrada por consola.

4. Telegram.py, este fichero Python es invocado desde Ocam.py, y se encarga de conectarse con el bot de Telegram para enviar las notificaciones.
5. server.py, este fichero Python se encarga de abrir la interfaz web para interactuar con el usuario. Dicha URL se abrirá en localhost donde trabaja el servidor Apache.

Además, como vemos en la imágen, están representados los distintos tópicos por los cuales se enviará la información (/<tópico>). El módulo web_server se encarga de crear en tiempo real una visualización de los tópicos en un servidor local (localhost).

Por otro lado, tenemos el directorio con los archivos del servidor web, dentro del mismo, hay varios archivos PHP por sección, y en la carpeta record_videos tendremos almacenado los videos que se irán grabando cuando se detecte una persona en alguno de los diferentes inputs.

El código correspondiente a este proyecto puede encontrarse en:
https://github.com/alu0100812534/TFG

4.2 Obtención de la fuente de datos mediante Selenium

1. El módulo abre mediante el módulo Selenium en Python un navegador Firefox en segundo plano. Este navegador usa un perfil de usuario propio, completamente limpio. Esto implica que no va a tener almacenadas cookies, o información de navegación previa. Esto incluye la extensiones de Firefox. Como hay páginas como Youtube que añaden anuncios en los vídeos, ha sido necesario instalar la extensión Adblock para evitar posibles anuncios. Esto se hace en tiempo de ejecución.
2. Se abre la URL por defecto y de forma iterativa se van capturando frames. Se transforma cada frame captado a BGR. Además, se recorta de dicho frame las coordenadas donde está situado el video sobre el cual se desea realizar la detección.
3. Se va enviando frame a frame el video captado mediante un Topic ROS.

4.3 Módulo de detección de personas

4.4 Detección de personas

4.4.1 Funcionamiento

Para la detección, se ha usado el módulo de OpenCV para cargar y trabajar con redes de tipo DNN. Para ello se ha pasado una red ya entrenada desarrollada en Caffe. Dicha red posee dos ficheros fundamentales, el .prototxt y el .caffemodel, ambos se le deben pasar como parámetro al módulo de OpenCV. El .prototxt contiene información relativa a la topología de la red, mientras que el .caffemodel contiene la información de los pesos de las conexiones entre neuronas, una vez entrenada la red.

Una vez cargada la red, se aplica la función *blobFromImage* para preprocesar la entrada frente a cambios de iluminación, a la vez que se redimensiona al tamaño de entrada esperado (unos 300x300 pixels).

A continuación, una vez pasado a la red el blob, se realizan las detecciones, teniendo en cuenta un umbral de un 50% de probabilidad, es decir, 50% de probabilidad que el objeto detectado sea una persona. Hay que tener en cuenta, que la red inicialmente está entrenada para la detección de otros objetos, tales como sillas, coches o incluso trenes, pero, para nuestro caso la red ha sido acotada a personas, mediante el propio código Python. En algún momento del desarrollo del proyecto se planteó el uso de transfer learning para "especializar" una red previamente entrenada para múltiples clases, de forma que sólo fuera capaz de detectar personas pero con mucha mayor precisión. Esta idea finalmente se descartó por falta de tiempo.

Por último, las detecciones se dibujan sobre el video de entrada (una copia) y es enviado por un Topic ROS como salida, de tal forma que la salida será el mismo video de entrada pero con las detecciones ya dibujadas sobre el mismo, además, de poder ver la probabilidad de la detección.

4.4.2 Consideraciones

Como se ha descrito con anterioridad, se ha usado una red ya entrenada en Caffe, esto se debe, a que el entrenamiento de una red desde el inicio, puede conllevar una gran inversión de tiempo, y más teniendo en cuenta de redes para tratamiento de imágenes, solo hay que ver la documentación aportada en el Anexo sobre la cantidad de imágenes que se emplearon para entrenar dicha red, para darse cuenta de los volúmenes de datasets necesarios para conseguir un rendimiento óptimo de la misma.

Entrando en detalle, para el funcionamiento de la red en nuestro módulo, ha sido necesario prestar atención a los diferentes ficheros de la misma, principalmente al fichero prototxt. Este fichero, si lo abrimos con cualquier editor, en las primeras líneas vemos la siguiente configuración:

```
name: "MobileNet-SSD"
input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 300
  dim: 300
}
```

Configuración fichero prototxt.

Los campos *dim* pertenecientes al objeto *input_shape* indican los tamaños correspondientes a cada una de las 4 dimensiones de la entrada. Es importante tener en cuenta que debe coincidir con las dimensiones de la imagen de entrada proporcionada por los otros módulos, y en caso de ser diferente, se debe redimensionar, tal y como se muestra en el siguiente código Python del proyecto:

```
global lastime
global out
(h, w) = image.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 0.007843, (300, 300), 127.5)
```

Redimensionamiento de la imagen de entrada a 300x300.

Esta red, ha demostrado en la práctica tener un buen rendimiento para videos con una calidad bastante aceptable (480p o superior). Hay que tener en cuenta, que la red se puede volver a entrenar y mejorar (Fine-Tuning), pero también puede suceder que al realizar dicho entrenamiento, la red “olvide” y por tanto obtener peores resultados. También hay que tener en cuenta que al hacer la redimensión a 300x300 pixels, los objetos más alejados (por ejemplo persona al fondo de una calle larga) van a tener una representación en la imagen demasiado pequeña como para ser detectados.

4.5 Sistema de aviso al usuario

El segundo bloque es los avisos al usuario, básicamente, cuando la red detecta una persona, este módulo se conecta a una base de datos MySQL mediante PyMysql e inserta en dicha base de datos el tiempo, mensaje y tipo de input cuando una persona es detectada. Además, cuando una persona deja de ser detectada (pasados 5 segundos) se inserta dicha información en la base de datos, de tal forma que se recoge cuando entra y cuando sale una persona en concreto.
Respecto a esto, hay que tener en cuenta que la red no es capaz por sí sola de detectar cuando está detectando a una misma persona, es decir, no es persistente, si no lo que hace es avisar cuando detecta un objeto y cuando deja de detectar dichos objetos, pero realmente no es consciente de que se refiere al mismo.

A su vez, este bloque llama a otro módulo que se encarga de enviar dichos avisos anteriormente conectados a un Bot en Telegram, de forma que el usuario también tendría los avisos por mensajería instantánea.

4.6 Desarrollo de la interfaz de control

4.7 Resumen

La interfaz de control se puede dividir en dos bloques principales, el servidor web Xampp y el framework ROS. A la hora de ejecutar el código para su puesta en funcionamiento, es necesario en primer lugar, lanzar el Xampp, para que cuando el código Python intente acceder a él, esté todo funcionando correctamente. También, por último, tendríamos el bloque extra de comunicación vía Telegram de las notificaciones.

4.8 Servidor Web Xampp

El objetivo fundamental del servidor, es mostrar los Topics ROS que están ejecutándose en tiempo real. Concretamente, se muestra los Topics de output de la red neuronal, con las personas ya detectadas para cada input (Webcam y Selenium). Mediante la extensión ROS de web server, podremos hostear en local dicha salida de video, quedando (una vez ejecutado ROS como se indica más abajo) las siguientes rutas locales con cada Topic:

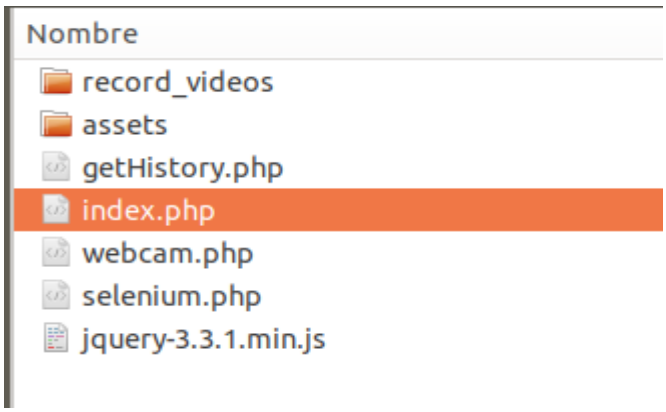
- http://localhost:8080/stream?topic=/output_camera , en la cual vemos el video captado por la webcam y procesado a posteriori por la red neuronal en tiempo real.
- http://localhost:8080/stream?topic=/output_selenium , en la cual vemos el video captado por Selenium y procesado a posteriori por la red neuronal.

Ambos enlaces estarán disponibles una vez lanzado ROS tal y como se explica más abajo, pero, como se nombra anteriormente, primero es necesario lanzar el servidor Xampp, ya que si se ejecuta ROS en primera instancia, este redireccionará al servidor Xampp automáticamente.

Para más información se puede visitar el siguiente enlace, en donde se explica y describe la instalación y puesta en marcha del módulo web server.

https://wiki.ros.org/web_video_server

4.8.1 Estructura

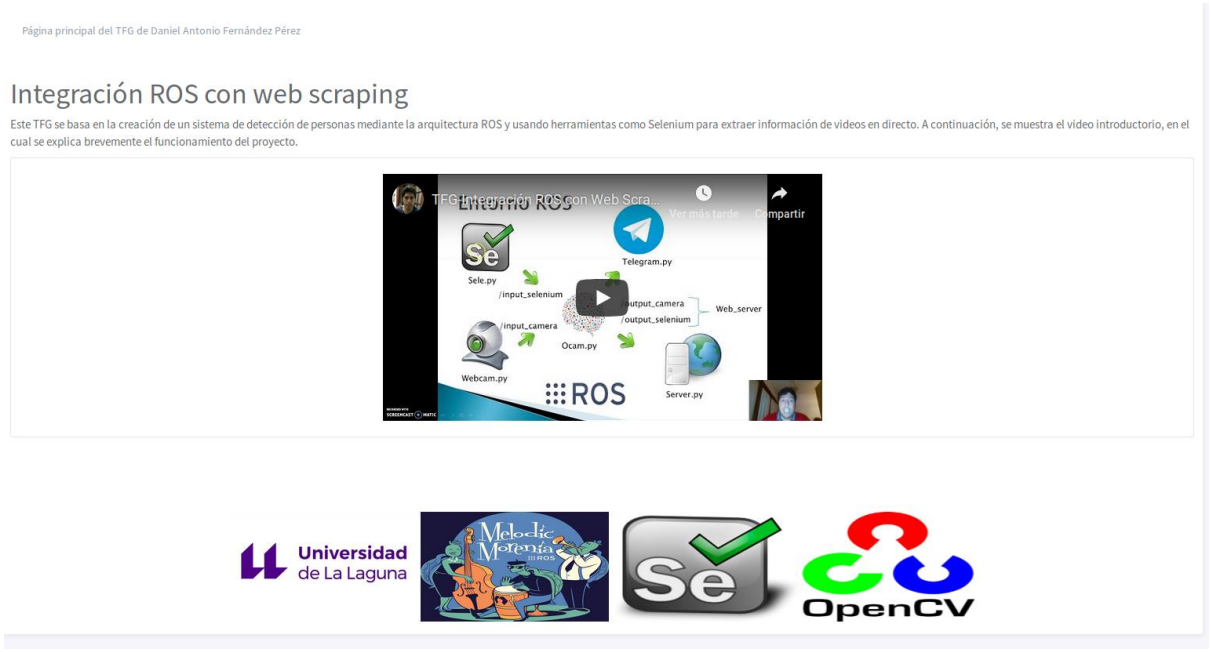


Estructura principal del servidor local.

En la anterior imagen vemos la estructura principal del servidor web, todos estos archivos se sitúan dentro la de carpeta htdocs de Xampp.

- En la carpeta record_videos, se guardan los vídeos cada vez que la red detecte a una persona, hasta que esta se deje de detectar, en formato MP4.
- La carpeta assets posee archivos de la plantilla web, visuales, tales como Javascript y CSS.
- GetHistory.php es un archivo PHP que se encarga de mediante el uso de JQuery cargar en tiempo real la información de los avisos del usuario, de forma que el usuario no necesite recargar la web local para ver nuevos avisos.
- Index.php es el archivo principal que se mostrará al abrir el servidor en el navegador, principalmente contiene créditos y el vídeo procedente de nuestro pipeline.
- webcam.php es el archivo que muestra el Topic de la salida de la red neuronal con las personas ya detectadas en la webcam. Además, tiene otras funcionalidades como poder ver vídeos guardados en línea y mostrar el historial de avisos.
- selenium.php es el archivo que muestra el Topic de la salida de la red neuronal con las personas ya detectadas en la entrada de Selenium. Además, tiene otras funcionalidades como poder ver vídeos guardados en línea y mostrar el historial de avisos.
- El último fichero es la versión ligera de JQuery 3.3.1, qué podemos descargarla en el siguiente enlace : https://jquery.com/download/ . Dicho archivo es necesario para poder ejecutar JQuery en nuestro código PHP.

4.8.2 Interfaz de Inicio

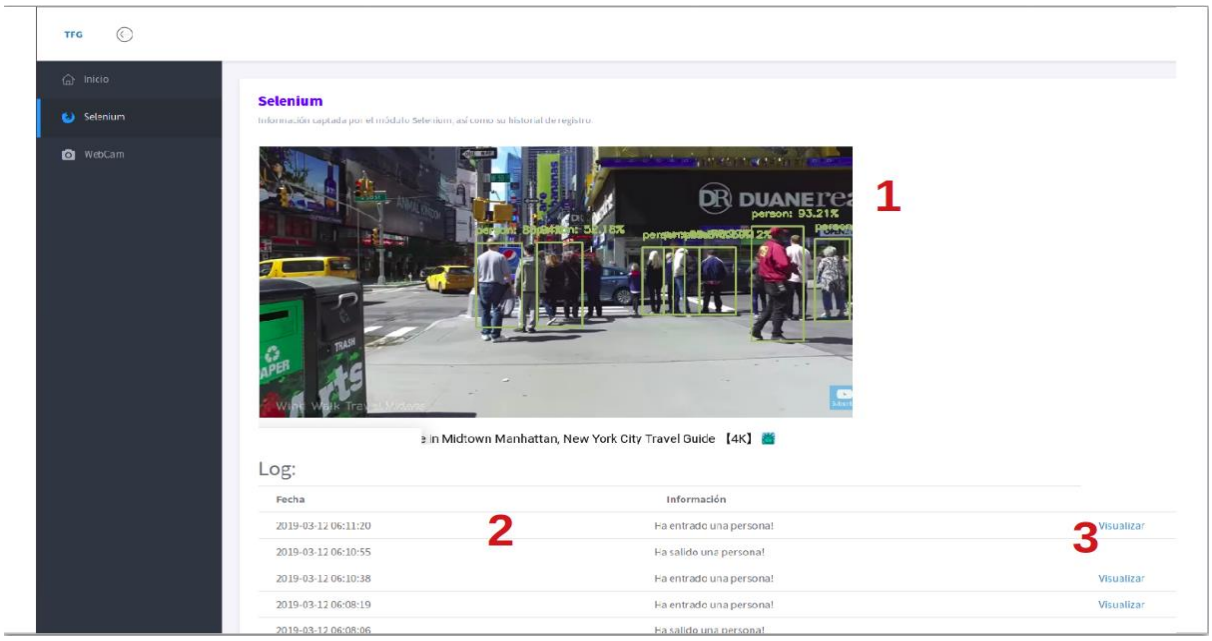


Diseño interfaz de inicio del servidor web.

La interfaz de inicio es bastante sencilla, en ella podemos observar el vídeo explicativo del TFG, así como el logotipo de las diferentes herramientas usadas. Hay que tener en cuenta, que el servidor Xampp debe estar correctamente ejecutado para poder visualizar esta interfaz.

El video se puede visualizar de forma independiente en el siguiente enlace: https://www.youtube.com/watch?v=l-utwfSnKS0

4.8.3 Interfaz de Selenium y WebCam

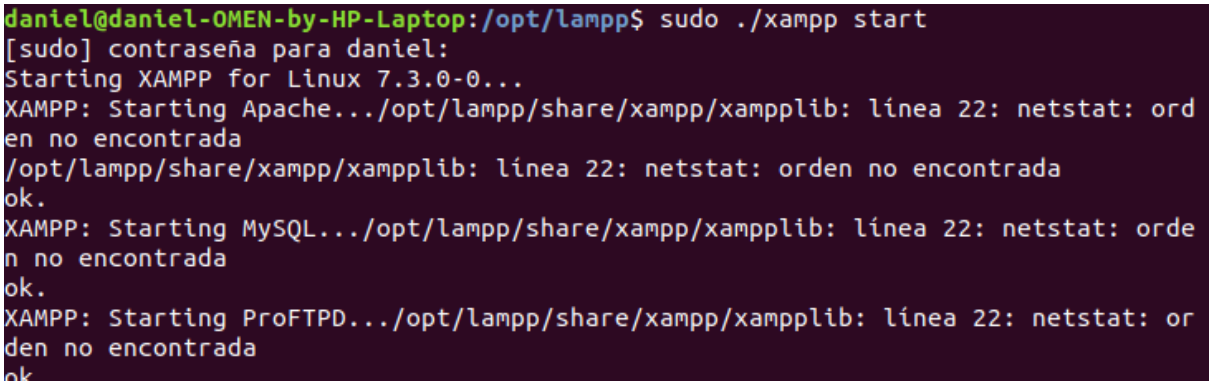


Diseño menú Selenium/webcam del servidor web.

1. En esta zona visualizamos el vídeo, dependiendo de si es webcam o selenium, nos saldrá una u otra imagen ya procesada por la red neuronal.
2. En esta sección veremos el registro de los usuarios que vaya detectando. Hay que tener en cuenta que para que aparezca como “el usuario ha salido” tienen que haber pasado 5 segundos sin detección de alguna persona, ya que no es persistente.
3. Aquí podremos visualizar los vídeos guardados en nuestro servidor, de forma que en todo momento podremos reproducir una secuencia de captura.

4.8.4 Inicio del servidor Apache

Para iniciar nuestro servidor Apache, debemos ir a la ruta /opt/lampp y ejecutar el siguiente comando:



Ejecución del servidor Xampp.

Como se puede observar en la imagen, el servidor se ha iniciado correctamente, y que, aunque dé algunos warnings de compatibilidad, estos no son problema para su correcto funcionamiento.

Una vez realizado lo anterior, el servidor Apache usará como raíz la carpeta htdocs/ , de forma que si tenemos una carpeta ubicada en htdocs/tfg, la ruta de acceso en nuestro caso será : http://localhost/tfg/

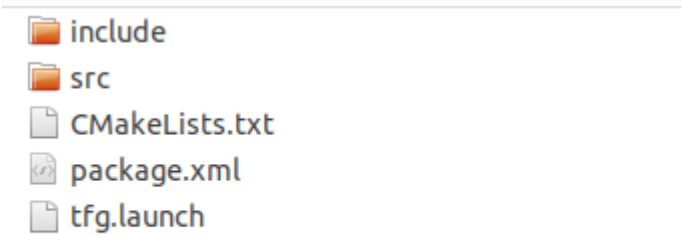
Nota: Hay que tener en cuenta que mediante ciertas configuraciones del Xampp, es posible abrir el servidor a Internet, de forma que quedaría público. El funcionamiento sería parecido pero ya no sería localhost, si no la IP de la máquina, para ello se tendría que poseer una IP fija para hostear nuestro servidor.

4.9 Framework ROS

Una vez iniciado el servidor web como se ha indicado con anterioridad, ahora es necesario iniciar el servicio de ROS y cargar todos los módulos asociados al proyecto, para ello hay que tener un par de consideraciones y conocer la estructura planteada.

4.9.1 Estructura

Los ficheros principales los encontramos dentro del directorio catkin_ws/src/tfg



Directorio principal del proyecto

En este directorio, en la carpeta src/ tendremos los diferentes nodos ROS del proyecto, open_cam, opencv_cam, selenium y server, y dentro de los mismos, los diferentes ficheros Python con sus funcionalidades.

Una consideración a tener, es que para facilitar la ejecución del proyecto y parsear ciertos valores de los módulos, se ha decidido lanzar los diferentes nodos mediante una única ejecución, a partir de un fichero .launch

4.9.2 ROS Launch

Roslaunch es una herramienta para lanzar múltiples nodos de ROS de forma cómoda y ordenada.
En este proyecto, el fichero .launch tiene la siguiente configuración:

```
<launch>
  <node pkg="tfg" type="webcam.py" name="webcam" args="0"
  output="screen">
     <remap from ="image_ch" to="input_camera" />
  </node>
  <node pkg="tfg" type="Ocam.py" name="Ocam_camara" output="screen" >
     <remap from ="image_ch" to="input_camera"/>
     <remap from ="net_topic" to="output_camera"/>
     <param name ="source" value="0" type="int" />
  </node>
  <node pkg="tfg" type="telegram.py" name="telegram"
  output="screen"/>
  <node pkg="tfg" type="sele.py" name="selenium" args="0"
  output="screen">
     <remap from ="image_selenium" to="input_selenium" />
  </node>
  <node pkg="tfg" type="Ocam.py" name="Ocam_selenium" output="screen" >
     <remap from ="image_ch" to="input_selenium"/>
     <remap from ="net_topic" to="output_selenium"/>
     <param name = "source" value="1" type="int"/>
  </node>
  <node pkg="web_video_server" type="web_video_server" name="web_video_server"
  output="screen"/>
  <node pkg="tfg" type="server.py" name="server"
  output="screen"/>

</launch>
```

Estructura del fichero tfg.launch

Dentro de este fichero, vemos como indicamos cada nodo a ejecutar con la etiqueta <node>. Según se muestra en la imagen algunos nodos se van a ejecutar duplicados, ya que para el proyecto interesa captar tanto la entrada de webcam como la de selenium a la vez, teniendo en cuenta a remapear los tópicos. También, se ha definido el parámetro “source” para pasarle al código que input se está procesando, de cara a la interacción con el usuario.
Si se quisieran usar más entradas, es tan sencillo como duplicar el grupo de nodos correspondiente.

4.9.3 Inicio del framework ROS

Para iniciar el framework ROS con la configuración y los módulos de nuestro proyecto, basta con ejecutar el siguiente comando:

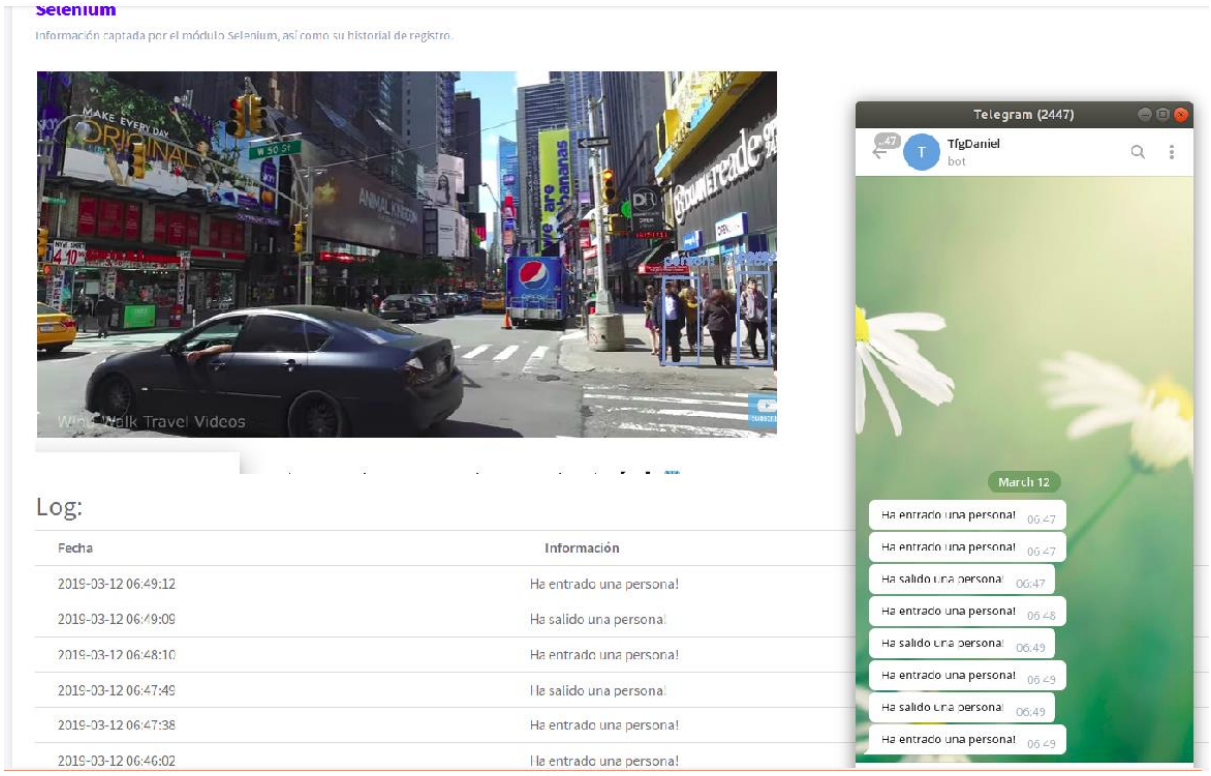
roslaunch tfg tfg.launch

Si todo va bien, se nos abrirá automáticamente una ventana en el navegador con la información para interactuar con el usuario.

Nota: Puede ocurrir que no detecte algún módulo, y por tanto no reconozca el fichero tfg.launch ni su ubicación, para solucionar este inconveniente basta con que antes de ejecutar el comando anterior, escribir en la carpeta catkin_ws/devel/, el comando **source setup.bash**. De este modo, habremos cargado nuestro workspace en el sistema.

4.10 Notificaciones vía Bot Telegram

Además de las notificaciones vía web, se ha implementado un sistema de notificaciones vía Telegram, de forma que, un bot asociado al proyecto puede comunicar al usuario mediante un canal de Telegram las alertas que genera su sistema. En el vídeo a continuación se puede ver el funcionamiento de este bot, la configuración e instalación del mismo puede verse en los anexos.



Funcionamiento del bot desde Telegram desktop, comparándolo con las alertas web.

Como se ve en la imagen, el bot envía las alertas de entrada y salida al usuario. Esta configuración se puede modificar en el módulo asociado al proyecto llamado “Telegram”. Hay que tener en cuenta, que si se genera un nuevo canal de charla con el bot, es muy probable que necesitemos incorporar un nuevo ID, ya que las alertas se envían a un bot y un canal que lleva asociado el mencionado ID.

4.11 Resultados

4.12 Resultados usando Selenium

A continuación se va mostrar el rendimiento del módulo Selenium, a partir de unas imágenes tomadas del procesamiento:

Imagen de detección a partir de Selenium, 1

.

En la primera imagen, se puede observar como nuestra red es capaz de detectar correctamente a las personas que están más próximas a la cámara, pero, al sistema le es difícil distinguir a las personas que están lejos, ya que el rendimiento de esta red neuronal está condicionado a la concentración de píxeles a analizar, cuanto mayor sea la calidad, mejores resultados se obtendrán.

Imagen de detección a partir de Selenium, 2

En nuestra segunda imagen con menor calidad, el sistema sigue manteniendo un buen rendimiento, aunque, se puede observar que aún se le escapa algo de información.

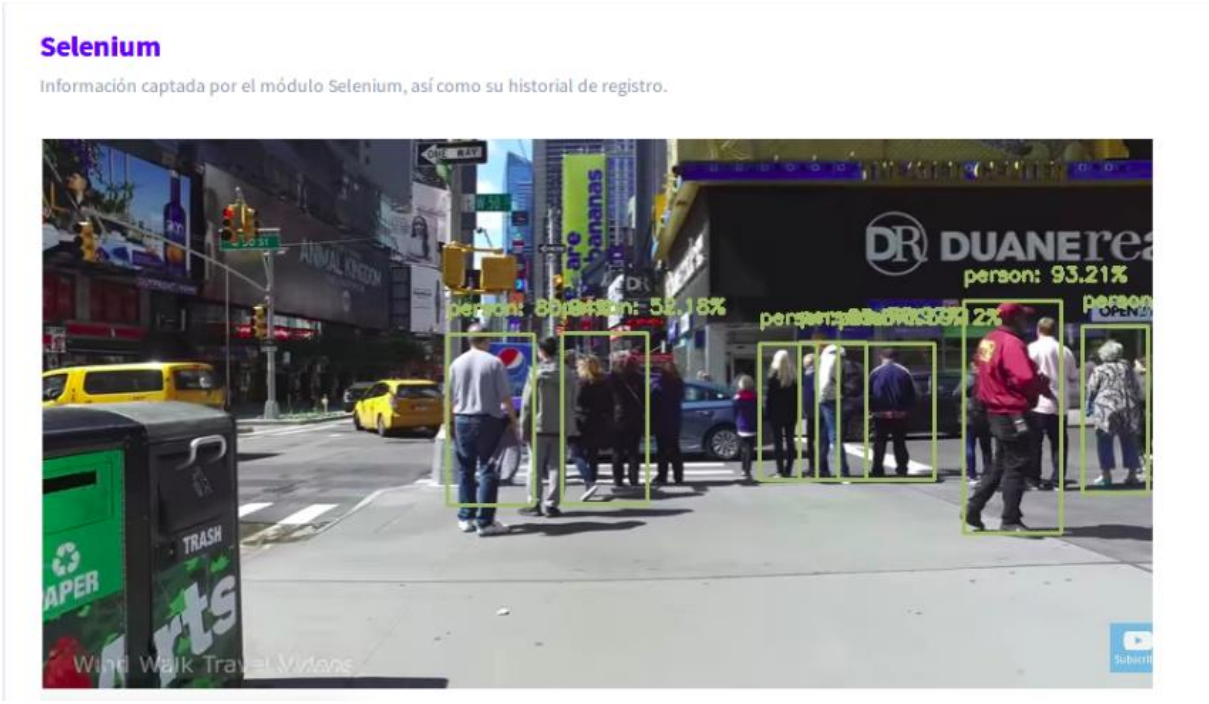


Imagen de detección a partir de Selenium, 3

En esta imagen, la red ha obtenido un rendimiento óptimo, ya que se da las circunstancias que las personas a identificar están a una distancia reconocible y la calidad de la imagen está en HD.

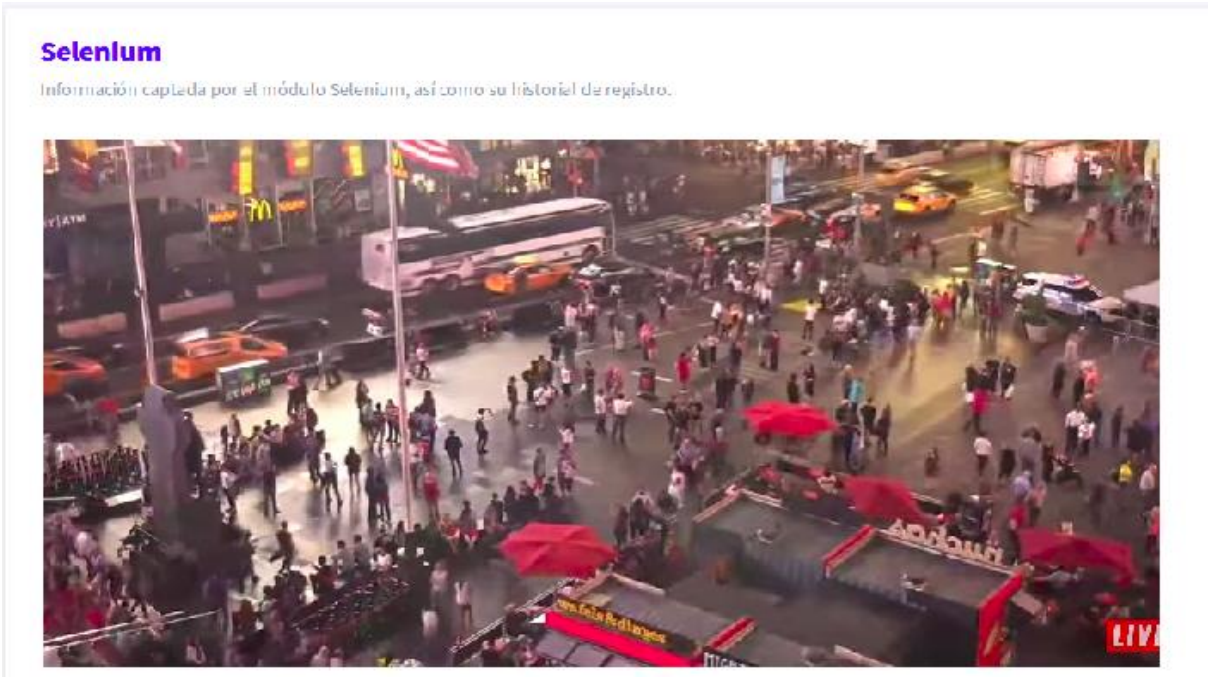
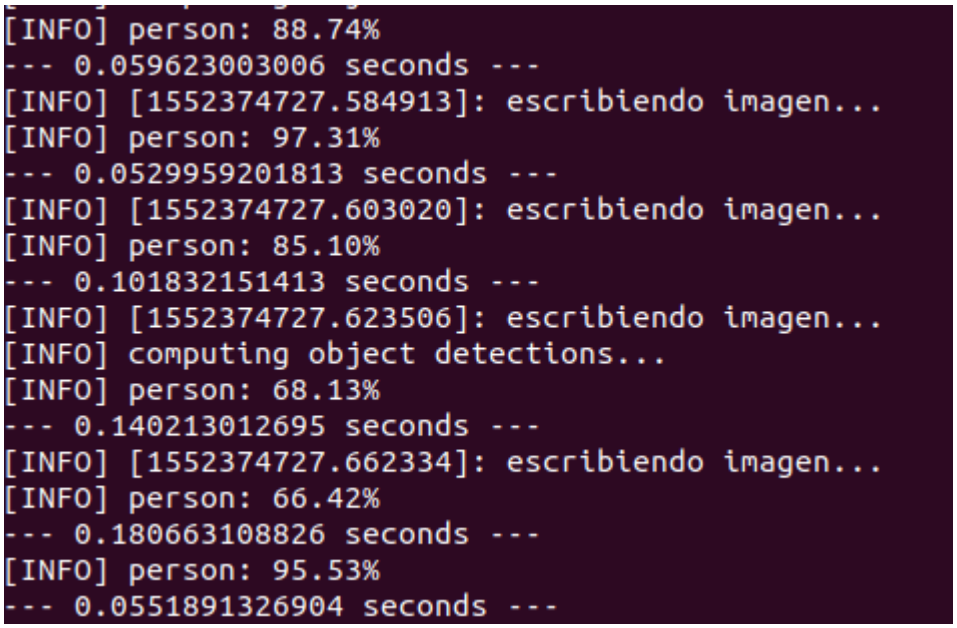


Imagen de mala detección a partir de Selenium, 4

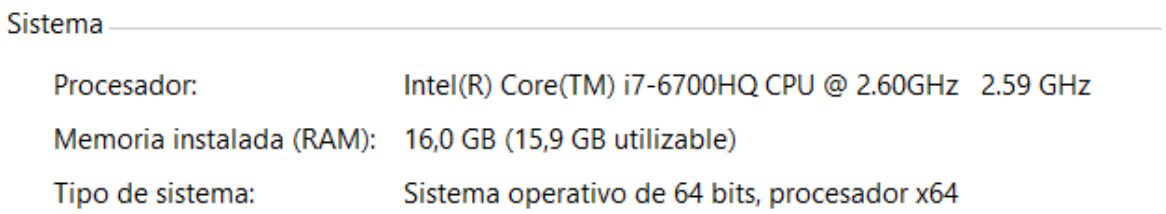
En nuestra última imagen, al contrario, la red no ha sido capaz de detectar a las personas, ya que la distancia es demasiado lejana y de baja calidad para su correcto procesamiento.

En general, el tiempo de procesamiento es bastante bajo y no llega ni al segundo por persona detectada, tal y como se ve en la siguiente imagen:



Consola del proyecto, con las personas detectadas, tiempo y probabilidad.

Todos los tiempos de este documento se han medido sobre un ordenador personal con las siguientes características:



Características ordenador personal.

El SO empleado es una Ubuntu 18.04, usando ROS Melodic. Además de lo ilustrado, el ordenador cuenta con 4GB de CPU dedicada.

Como se ha nombrado con anterioridad, el módulo Selenium recorta la sección donde se encuentra situado el vídeo dentro del navegador, para su procesamiento. Esto libera de bastante sobrecarga al sistema, ya que se tiene que centrar únicamente en esa región. A continuación, para mostrar esto, vamos a volver a usar la imagen ilustrada más arriba, y se va a comparar con la nueva imagen pero que esta vez estamos usando la ventana completa del navegador:

Resultado recortando la zona a examinar:

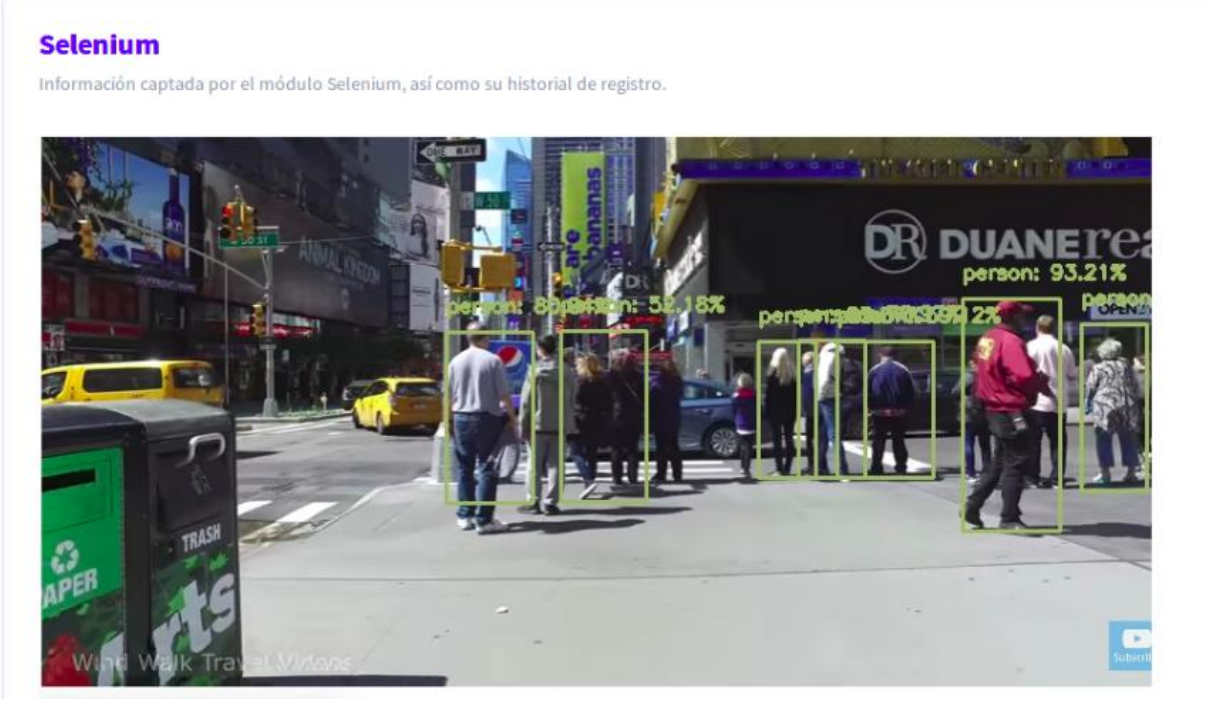


Imagen resultado con zona recortada.

Resultado sin recortar la zona, y por tanto usando la ventana completa del navegador:

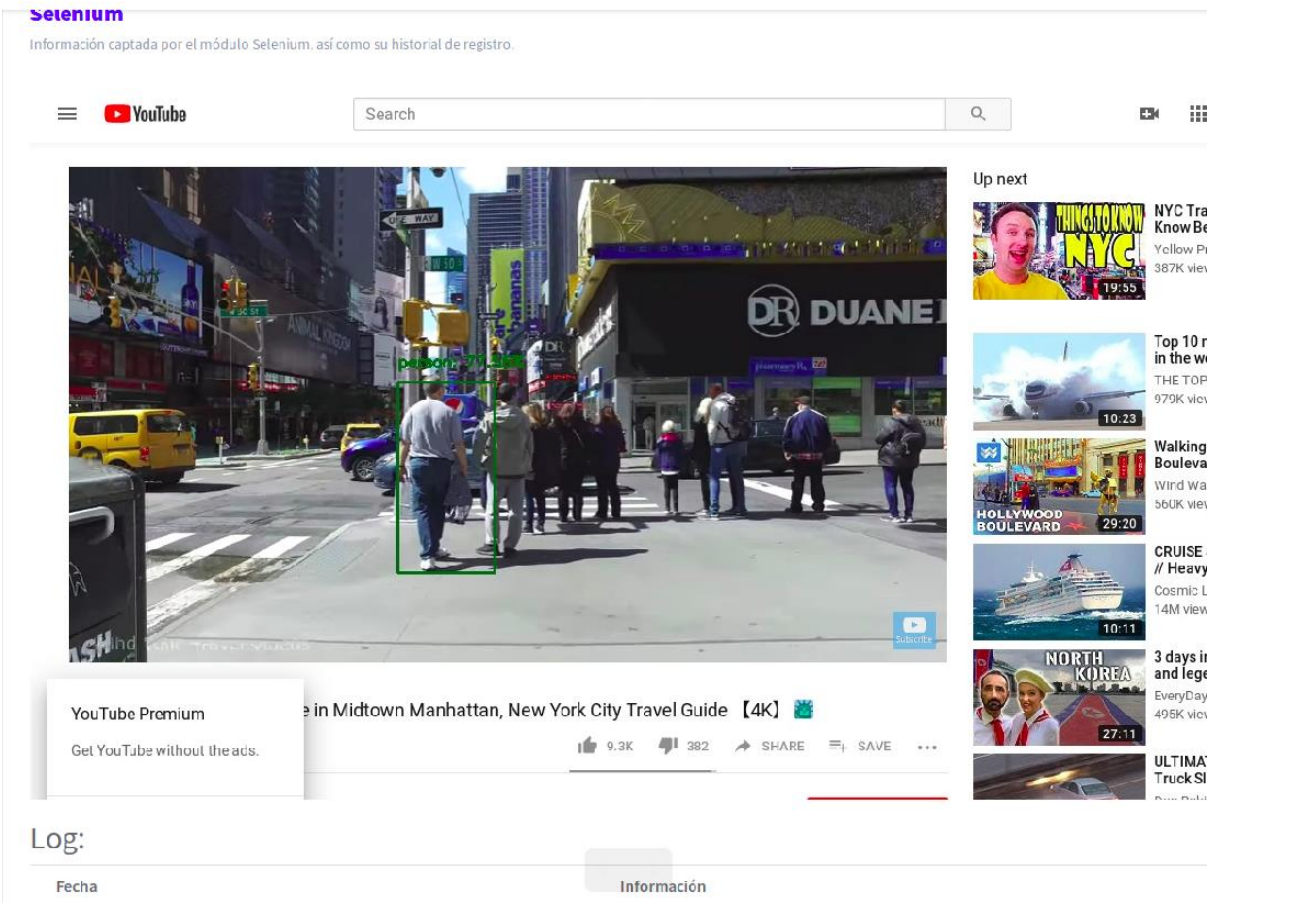


Imagen resultado con zona sin recortar.

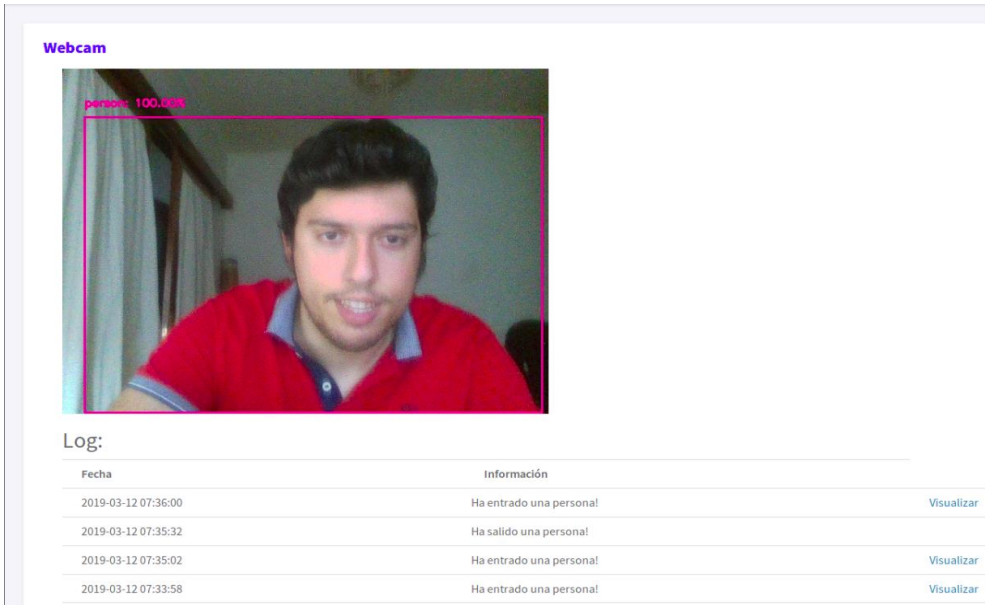
Como se ha observado, la tasa de detección es bastante inferior, por lo que es necesario seleccionar y recortar la zona de la cual se desea extraer información, para liberar de carga al sistema y conseguir el mejor resultado posible.

4.13 Resultados usando cámara real

El módulo de la webcam, en general, da mejores resultados que el de Selenium, ya que hay que tener en cuenta que normalmente va a procesar poca cantidad de elementos de entrada, y además, estos elementos estarán situados a una distancia reducida. En cambio, en el módulo Selenium, los vídeos de streaming que se le puedan pasar, pueden tener baja calidad y poseer todo tipo de elementos con sus respectivas distancias.

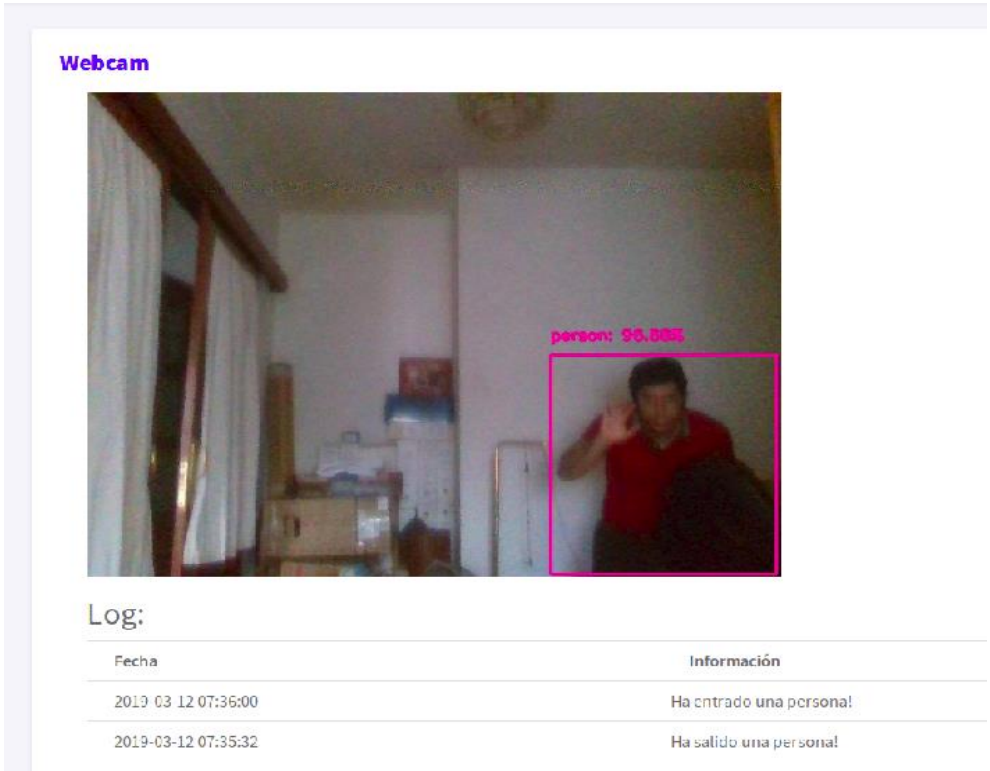
A continuación se muestra ejemplos de detección de este módulo, teniendo en cuenta que es una cámara de un ordenador portátil HD.

Distancia cercana a la webcam:



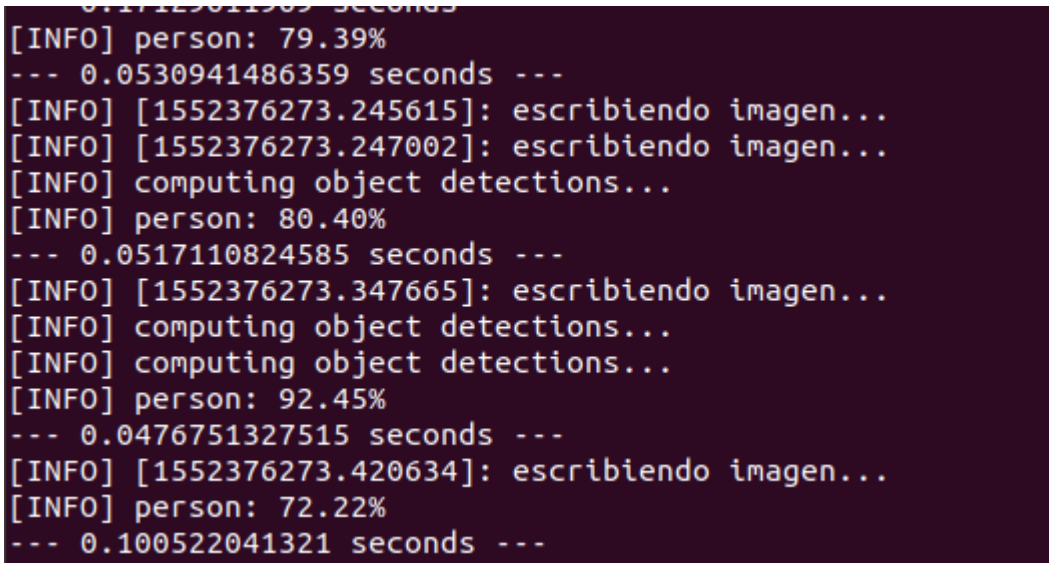
Procesamiento desde una distancia cercana a la webcam.

Distancia lejana con cuerpo parcialmente oculto:



Distancia lejana con cuerpo parcialmente oculto.

El rendimiento para el proceso de toma de los datos ha sido parecido, ambas detecciones han sido inferiores a un segundo:



Resultados de predicción y rendimiento de las imágenes de ejemplo.

4.14 Conclusión resultados

La red ha demostrado tener un buen rendimiento para ambos casos, selenium y la webcam del ordenador portátil. Donde notamos diferencias notorias de rendimiento es cuando situamos objetos a baja resolución y a distancias lejanas del objetivo de la cámara, principalmente en el módulo Selenium.

Respecto a los tiempos, en todos los casos el tiempo de procesamiento es bastante bajo, pero, hay que tener en cuenta que el proyecto tiene sus limitaciones impuestas en el código de OpenCV, es decir, el procesamiento tiene un pequeño delay intencionado para que el rendimiento sea más óptimo sin consumir tantos recursos de la máquina. Además, las alertas se envían cada cinco segundos de dejar de detectar a alguna persona en el entorno de procesamiento. Todas estas medidas han sido tomadas para asegurar el mejor rendimiento posible del programa.

En conclusión, el sistema demuestra ser robusto y flexible para cualquier mejora o implementación futura que se le quiera realizar, ya que la red neuronal escogida como núcleo posee un buen rendimiento de procesamiento para cualquier análisis de imágenes para la que se la quiera usar. Además, los diferentes nodos del proyecto demuestran tener una estructura sólida y diseñada para conseguir la mayor eficiencia posible.

4.15 Interfaz de control

Para ilustrar el funcionamiento general del sistema, se ha grabado un vídeo en el cual se puede observar cómo los diferentes módulos interactúan entre sí para dar como resultado final a una detección bastante óptima: https://youtu.be/Q7yD0pgkVb4

Capítulo 5 Conclusiones y líneas futuras

5.1 Conclusión

En conclusión, el proyecto permite la implementación de un sistema ROS de alta precisión para la detección de personas, con diferentes entradas de vídeo. Todo esto manteniendo una usabilidad e interacción con el usuario, el cual puede en todo momento tener un registro de las detecciones, ya sea mediante web o mediante mensajería instantánea.
Su ejecución es sencilla y de fácil exportación, lo único necesario es tener un entorno ROS compatible.

Hay que tener en cuenta, que el sistema se puede mejorar notablemente, ya que se pueden incorporar módulos extras así como desarrollar o implementar nuevas redes neuronales, ya sea en local o en sistemas distribuidos.

5.2 Red neuronal

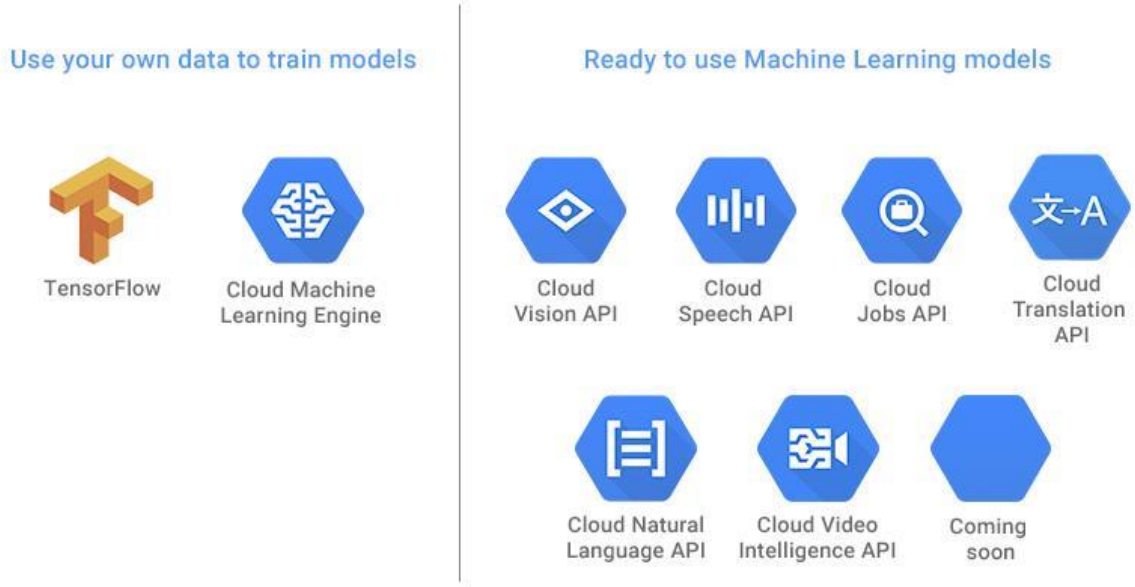
El sistema, según los resultados mostrados, obtiene un buen rendimiento de detección cuando se acota bien los parámetros y el margen de detección, pero, como se ha podido observar, se podría obtener un mayor rendimiento, teniendo en cuenta las siguientes consideraciones.

La red actual puede ser mejorada y re-entrenada eliminando las clases por defecto (detección de coches,etc) y centrándose únicamente en la detección de personas. Para esto, será necesario reestructurar el código Caffe tanto del archivo .prototxt como el de .caffemodel. Una vez modificado, se necesitará obtener Datasets de personas, y posiblemente, será necesario tener que entrenarla desde el principio (aunque se puede partir de los pesos de la red original en aquellas partes que no hayan cambiado para acelerar el proceso de aprendizaje). También se podría plantear traducir la red existente a otras herramientas, como Tensorflow o PyTorch, aprovechando así algunas de las novedades que traen asociadas.

También es posible sustituir la red actual por otra que sea capaz de hacer object tracking, como por ejemplo las redes SORT (https://github.com/abewley/sort), o DeepSORT (https://github.com/nwojke/deep_sort), entre muchos otros ejemplos. El objetivo sería permitir que en todo momento la red detecte cuando una misma persona aparece en la

imagen y cuando deja de aparecer, de forma que esto ahorrará códigos implementados en la actual red, mientras que mejoramos la información que el usuario recibe.

Fuera cual fuese nuestra elección, lo óptimo es usar tecnologías en la nube, como Google Cloud o Amazon AWS, ya que estas permiten alto rendimiento de cálculos y entrenamiento para redes neuronales enfocadas al ámbito científico. De esta forma, evitaremos la necesidad de encontrar un Hardware con prestaciones adaptadas a este tipo de desarrollos (principalmente hardware con varias GPUs). La única dificultad añadida, sería el aprendizaje de estas plataformas, y que, aunque todas ellas poseen documentación para ayudar al usuario, siempre tendríamos que estar controlando y adaptando nuestro desarrollo a la estructura de dichas plataformas.



Ejemplo funcionalidades para Inteligencia Artificial de Google Cloud.

Para más información, se pueden visitar los siguientes enlaces:

https://cloud.google.com/products/ai/
https://aws.amazon.com/es/machine-learning/

Aunque se ha nombrado AWS y Google Cloud, existen otras plataformas que también pueden servir para ampliar y mejorar las funcionalidades de este proyecto

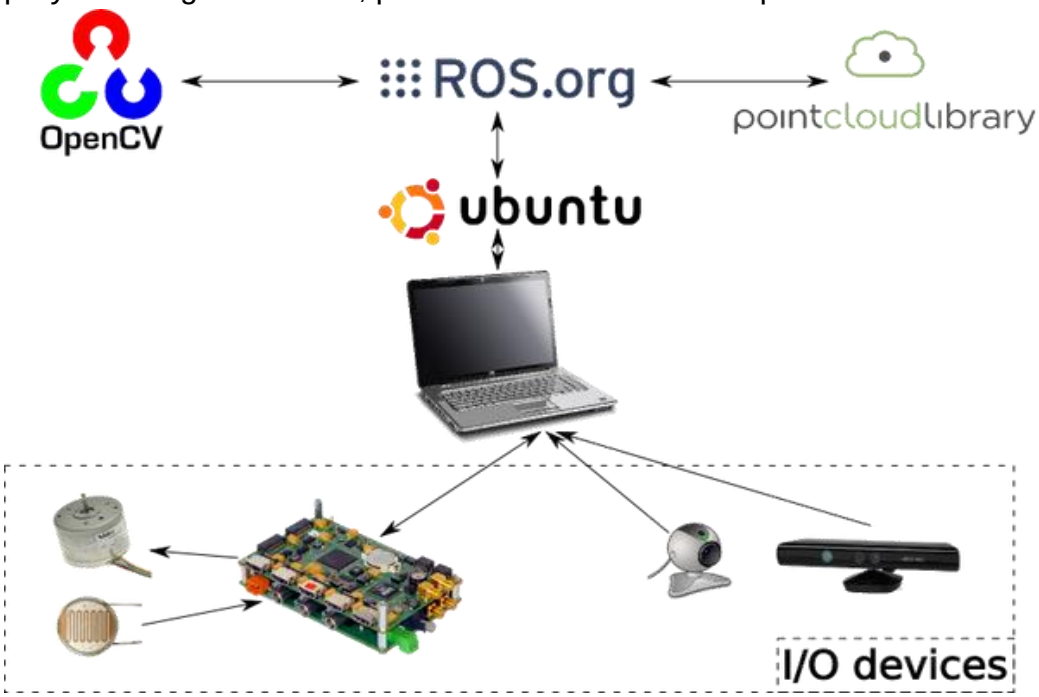
5.3 Módulos ROS

El proyecto es modular, ya que está desarrollado sobre el framework ROS, por lo tanto los diferentes nodos se pueden reutilizar para otros proyectos. Además, como es modular, podemos desarrollar mejoras al sistema creando nuevos nodos tanto de input como de output

Como nodos de input, se podría captar las imágenes para el procesamiento a partir de drones, de tal forma que, un nodo ROS podría encargarse de obtener las imágenes mediante WiFi o GPRS para su posterior tratamiento.
También, en vez de usar la entrada de webcam, se podría usar una cámara de videovigilancia para realizar la captación, de forma que situandola por ejemplo en una esquina de una habitación, se podría mediante el sistema desarrollado, hacer un seguimiento de las personas que entran a la misma.

Otro uso, podría ser el sistema KINECT para la detección del entorno. Este sistema ha demostrado ser bastante efectivo y no requiere de un elevado coste para su implementación.

En general, ROS, debido a su modularidad, permite ampliar las capacidades de este proyecto en gran medida, pudiendo valer como base para el desarrollo de otros.



Ejemplo de ROS con diferentes nodos input/output.

5.4 Servidor web y nube

El servidor web también puede ser ampliado, tanto con un nuevo diseño del mismo como en funcionalidades.

Hay que tener en cuenta que, actualmente el servidor Apache se ejecuta en local, pero, posiblemente se requiera poder acceder a estos datos externamente, para ello el servidor se puede abrir a Internet.

También, es posible el requerir conectar algún nodo ROS con el proyecto actual desde otro servidor, para ello será necesario realizar ciertas configuraciones en el framework ROS.

Para más información, ver los anexos.

5.5 Sistema de avisos

El sistema de avisos también se puede mejorar. Actualmente, el sistema alerta al usuario mediante web (con un log) y a través de Telegram, con un bot.

Una mejora podría ser el envío de alertas al correo electrónico o incluso, mediante alguna librería de voz, realizar llamadas al usuario. También, se podría implementar una aplicación móvil asociada al proyecto, de forma que los avisos quedarían englobados dentro de la misma, y el usuario podrá consultar en todo momento la actividad de detección de su sistema implementado.

5.6 Conclusión final sobre mejoras

El proyecto dota de una base sólida y eficiente para implementar cualquiera de las mejoras nombradas anteriormente, por ello, es importante mantener actualizado los componentes del mismo y estar pendiente de las novedades que ROS vaya ofreciendo de cara a sus usuarios, así como posibles migraciones de versiones o nuevas funcionalidades multiplataforma.

El código correspondiente a este proyecto puede encontrarse en:
https://github.com/alu0100812534/TFG

Capítulo 6 Summary and Conclusions

6.1 Conclusion

In conclusion, the project allows the implementation of a high precision ROS system for the detection of people, with different video inputs. All this while maintaining a usability and interaction with the user, which can at any time have a record of detections, either through web or through instant messaging.
Its execution is simple and easy to export, the only thing necessary is to have a compatible ROS environment.

It should be borne in mind that the system can be improved significantly, as extra modules can be incorporated as well as develop or implement new neural networks, either locally or in distributed systems.

6.2 Neuronal network

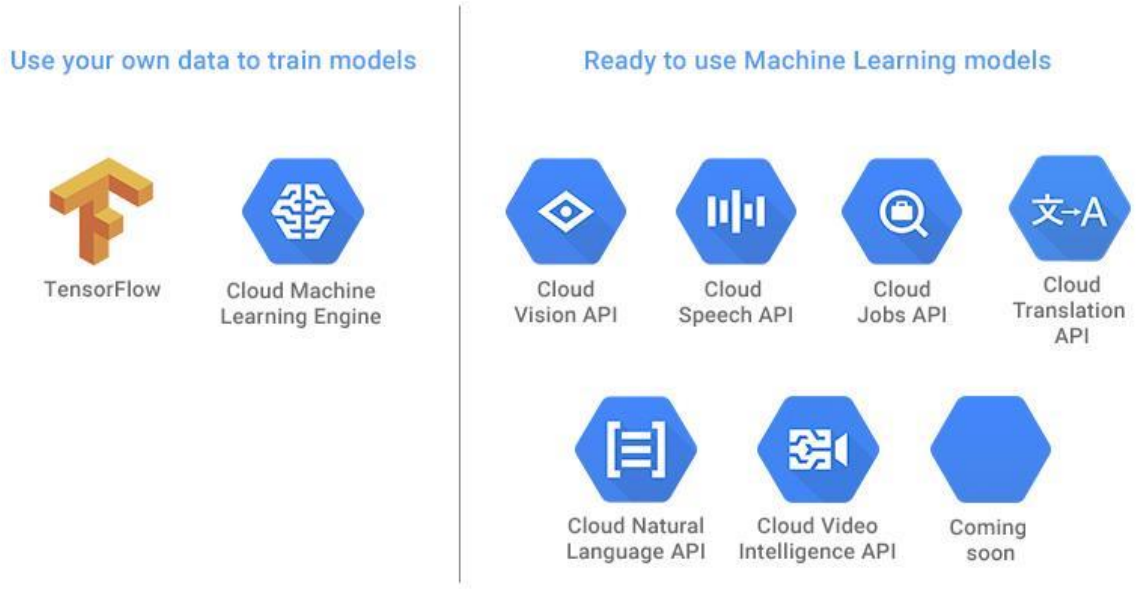
The system, according to the results shown, obtains a good detection performance when the parameters and the detection margin are well dimensioned, but, as it has been observed, a higher performance could be obtained, taking into account the following considerations.

The current network can be improved and re-trained by eliminating the default classes (detection of cars, etc.) and focusing only on the detection of people. For this, it will be necessary to restructure the Caffe code of both the .prototxt file and the .caffemodel file. Once modified, it will be necessary to obtain datasets of people, and possibly, it will be necessary to have to train it from the beginning (although you can start from the weights of the original network in those parts that have not changed to accelerate the learning process). You could also consider translating the existing network to other tools, such as Tensorflow or PyTorch, taking advantage of some of the novelties that they bring with them.

It is also possible to replace the current network with another one that is able to do object tracking, such as the SORT networks (https://github.com/abewley/sort), or DeepSORT (https://github.com/nwojke/deep_sort), among many other examples. The objective would be to allow the network to detect when the same person appears in the image and when it

stops appearing, so that this will save codes implemented in the current network, while improving the information that the user receives.

Whatever our choice, the best is to use technologies in the cloud, such as Google Cloud or Amazon AWS, since these allow high performance of calculations and training for neural networks focused on the scientific field. In this way, we will avoid the need to find a Hardware with features adapted to this type of development (mainly hardware with several GPUs). The only added difficulty would be the learning of these platforms, and that although they all have documentation to help the user, we should always be controlling and adapting our development to the structure of these platforms.



Example functionalities for Artificial Intelligence of Google Cloud.

For more information, you can visit the following links:

https://cloud.google.com/products/ai/
https://aws.amazon.com/en/machine-learning/

Although AWS and Google Cloud have been named, there are other platforms that can also be used to expand and improve the functionalities of this project.

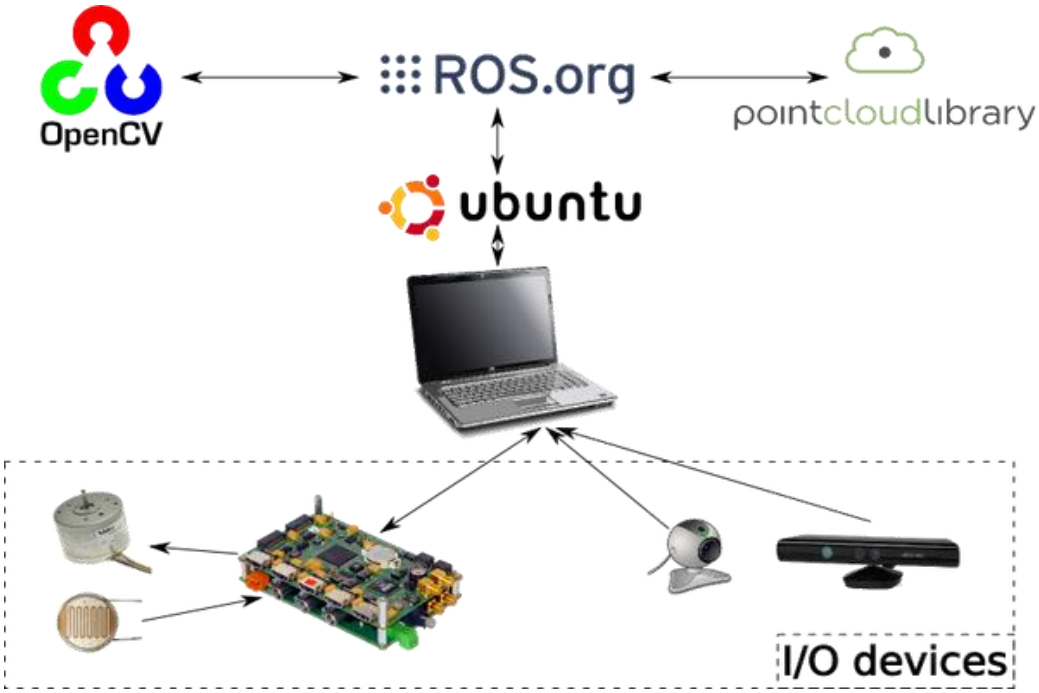
6.3 ROS modules

The project is modular, since it is developed on the ROS framework, therefore the different nodes can be reused for other projects. In addition, as it is modular, we can develop improvements to the system by creating new nodes of both input and output

As input nodes, the images could be captured for processing from drones, in such a way that a ROS node could take charge of obtaining the images via WiFi or GPRS for further processing.
Also, instead of using the webcam input, a video surveillance camera could be used to perform the capture, so that placing it, for example, in a corner of a room, could be followed by the system developed, tracking the people who enter it.

Another use, could be the KINECT system for the detection of the environment. This system has proven to be quite effective and does not require a high cost for its implementation.

In general, ROS, due to its modularity, allows to expand the capabilities of this project to a large extent, being able to act as a basis for the development of others.



Ros examples with differents input/output nodes.

6.4 Web server and cloud

The web server can also be extended, both with a new design of the same as in functionalities.

It must be borne in mind that currently the Apache server runs locally, but it may be required to access this data externally, for which the server can be opened to the Internet.

Also, it is possible to require connecting some ROS node with the current project from another server, for this it will be necessary to make certain configurations in the ROS framework.

For more information, see the annexes.

6.5 Warning system

The warning system can also be improved. Currently, the system alerts the user via web (with a log) and through Telegram, with a bot.

An improvement could be sending alerts to email or even, through a voice library, make calls to the user. Also, a mobile application associated with the project could be implemented, so that the notices would be included within it, and the user could consult the detection activity of his implemented system at all times.

6.6 Final conclusion on the improvements

The project provides a solid and efficient base to implement any of the improvements mentioned above, therefore, it is important to keep the components updated and be aware of the news that ROS is offering to its users, as well as possible migrations of versions or new multiplatform functionalities.

The code related to this project can be found at:
https://github.com/alu0100812534/TFG

Capítulo 7 Presupuesto

Para el cálculo del presupuesto, se van a tener en cuenta varias posibilidades. En primer lugar, se va a realizar un presupuesto en base a reproducir este proyecto a nivel de servidor de una empresa de seguridad, es decir, vamos a suponer que un cliente necesita esta detección de personas, tanto por cámara de videovigilancia como por streaming, y el otro caso, se va a realizar un presupuesto para realizar una estadística de varias cámaras en streaming, por lo que para obtener el mayor rendimiento posible, el procesamiento se va a realizar en Google Cloud.

7.1 Presupuesto materiales

7.2 Presupuesto materiales establecimiento

Para este caso, vamos a suponer que nuestro cliente tiene un establecimiento y la empresa de seguridad va a montar nuestro proyecto en este, a nivel local. Para ello, harán falta una cámara de videovigilancia, un ordenador con ciertas prestaciones a nivel de servidor local y conectado a un sistema de alertas, entre estas alertas, incluyendo las ya presentadas en el proyecto, se va a incluir un sistema por voz que alertará en la entrada de una persona al establecimiento.

A continuación, se hará un desglose del presupuesto material necesario para montar dicho sistema, usando como base nuestro proyecto. Es importante tener en cuenta, que todas las tecnologías usadas son Open Source, por lo que el software base necesario no necesitará de ningún coste adicional.

En la siguiente tabla, se desarrolla el coste del ordenador que trabajará como servidor local:

Elemento	**Cantidad**	**Precio total**
CPU (+6 núcleos, 3.50 GHz)	1	~200 Euros
GPU (+4Gb dedicada)*	1 o 2	~300 Euros
Memoria RAM 8gb	2	~100 Euros

Monitor	1	~140 Euros
Placa base*	1	~100 Euros
Disco duro 1TB*	1	~40 Euros
Resto de componentes*	-	~200 Euros

Tabla 1: Presupuesto ordenador servidor local

- Para el procesamiento de redes neuronales, lo recomendado es tener varias Tarjetas gráficas (GPUs), multiplicando el rendimiento cada una. Esto, es más importante que cualquier otro elemento.
- La placa base tiene que tener en cuenta ese número de GPUs a establecer.
- Si no se van a realizar numerosas lecturas / escrituras de ficheros el rendimiento del proyecto será muy similar tanto en SSD como en HDD. A lo mejor, se puede separar el nodo de creación de videos de detección en un SSD si se considera.
- El resto de componentes no se han indicado al no ser relevantes, serían ratón, caja de la Torre, etc.
- El sistema operativo será Ubuntu, la versión recomendada es 18.04 o superior.
- Se ha considerado un buen rendimiento del procesado, intentando mantener un equilibrio con el precio.

En el caso de nuestro supuesto, hará falta una cámara (o varias) de videovigilancia, las cuales transmitirán en directo a nuestro proyecto mediante input ROS las imágenes. Se considerará un coste de ~60 euros por cámara HD, además, en caso de que el cliente quiera visualizar la información de su servidor desde fuera del establecimiento, será necesario contratar una IP fija para el servidor Apache, el cual ronda en unos 20 euros mensuales. El sistema de alerta acústica por voz (que funciona como nodo acoplado a nuestro proyecto) tendrá un coste de 50 euros.

En total, el presupuesto para este supuesto será de **1250 euros.**
Se ha tenido en cuenta para este, lo indicado en la tabla, y además se le han sumado 2 cámaras de videovigilancia, así como el sistema acústico de alerta. Por último faltaría la cuota mensual de la IP Fija.

7.3 Propuesto despliegue en la nube

Para este supuesto, se va a tener en cuenta, que se va a solicitar el servicio de Google Cloud para procesamiento de imágenes de video, así como la red neuronal, para ello en este caso los inputs serían varios nodos Selenium (igual definidos en el proyecto) y estos enviarán al nodo de procesamiento de la red dicha información, pero, en vez de tener una red local, esta red estará en la nube de Google.

Si tenemos en cuenta, que a lo mejor tenemos que usar la herramienta de AI & Machine Learning de Google Cloud, los precios que podemos encontrar en cuanto a nuestras necesidades son:

Niveles de escalabilidad predefinidos: precio por hora (y unidades de preparación)	
BASIC	0,3212 $ (0,5948)
STANDARD_1	3,3609 $ (6,2239)
PREMIUM_1	27,9794 $ (51,8138)
BASIC_GPU	1,3578 $ (2,5144)
CUSTOM	Si seleccionas un nivel de escalabilidad personalizado, podrás controlar el número y el tipo de máquinas virtuales utilizadas para la tarea de preparación. Consulta la tabla de tipos de máquinas.

Tipos de máquinas: precio por hora (y unidades de preparación)	
standard	0,3212 $ (0,5948)
large_model	0,8001 $ (1,4816)
complex_model_s	0,4795 $ (0,8879)
complex_model_m	0,9589 $ (1,7758)
complex_model_l	1,9179 $ (3,5516)
standard_gpu	1,3578 $ (2,5144)
complex_model_m_gpu	4,1464 $ (7,6785)
complex_model_l_gpu	8,2928 $ (15,357)
standard_p100 *(Beta)*	2,9784 $ (5,5156)
complex_model_m_p100 *(Beta)*	10,6288 $ (19,6830)

Precios por hora de las máquinas de Google Cloud para Machine Learning.

Si necesitamos conectar otra herramienta de Google Cloud a nuestro proyecto ROS, necesitaremos consultar la base de datos de herramientas disponibles y sus precios.

https://cloud.google.com/pricing/list

7.4 Presupuesto trabajo realizado

Para ambos supuestos, el trabajo realizado girará entorno al tiempo requerido para la instalación de los diferentes componentes del sistema, así como el posible entrenamiento de una red neuronal para el propósito especificado.

7.5 Presupuesto trabajo establecimiento

Para este supuesto se va a tener en cuenta el tiempo necesario para la puesta en marcha del ordenador servidor con el proyecto en su interior, para ello se va a dividir en varias etapas por hora de trabajo.

Trabajo	cantidad personas	horas
Creación del equipo de trabajo, puesta en marcha del hardware específico.	1	3
Configuración Ubuntu y entorno ROS	1	2
Instalación de dependencias y módulos necesarios*	1	2
Configuración red neuronal, ejemplo, Caffe	1	1
Instalación cámaras de videovigilancia y alerta acústica	1	1
Puesta en marcha del sistema y testeo	1	3

Tabla 2: Presupuesto trabajo establecimiento

- Esto es usando el proyecto actual como base. Si incluimos el tiempo de desarrollo del proyecto desde 0 (creación de códigos, etc.) el tiempo de realización del mismo va de 4 a 5 meses.
- Es una estimación en horas del tiempo que se necesitará para montar el sistema en el ordenador indicado.
- El coste dependerá del salario del trabajador, teniendo en cuenta que un técnico podría instalar dicho sistema en el establecimiento en menos de 1 día, por lo que la puesta en marcha del sistema podría tener un presupuesto inferior a **40 euros.**

7.6 Presupuesto trabajo despliegue en nube

En este caso, el presupuesto del trabajo irá relacionado con el tipo de estadísticas a realizar, así como del conocimiento de las herramientas necesarias para el mismo mediante Google Cloud.

El proyecto, necesitará el desarrollo de un módulo de conexión (un nodo ROS) o adaptación del módulo Ocam.py para que la red neuronal pase de estar en local a en la nube de Google.
Esto puede traer sus dificultades iniciales y requiera de una investigación, por lo que es coste puede ser superior a **1000 euros** en tiempo necesario para su desarrollo.

Si queremos sustituir la red neuronal actual del proyecto e implementar otra nueva, el tiempo necesario de entrenamiento puede variar notablemente teniendo en cuenta los siguientes factores:

1. Framework a usar para el desarrollo y entrenamiento, y su conexión con la nube de Google Cloud.
2. Estructura que tendrá la red y algoritmos necesarios para su óptimo resultado.
3. Implementación de la red y entrenamiento. Esto, será lo que más tiempo consuma, ya que es probable que nuestra red necesite una gran cantidad de datasets para conseguir un rendimiento óptimo. Por lo cual primero se necesitaría la búsqueda de dichos datasets y posteriormente conseguir a partir de ellos un buen resultado.

El presupuesto de trabajo en el caso de desarrollarse o entrenar una red propia, se puede incrementará exponencialmente, y podría ser superior a los **3000 euros** en muchos casos, en tiempo necesario para su realización.

Nota: Los presupuestos anteriores han sido calculados aproximadamente teniendo en cuenta el posible salario de un trabajador cualificado en el campo, no es un precio a tener como referencia exacta.

7.7 Presupuesto total del proyecto para los supuestos

En conclusión, el presupuesto dependerá de los supuestos anteriores y de la cualificación de las personas que lo implementen.

En el caso del primer supuesto, el presupuesto se puede calcular con mayor exactitud, ya que para dicho caso se va a usar el proyecto como base para su implementación, sin prácticamente necesitar ninguna implementación extra. El mayor coste sería el servidor local que alojaría el framework ROS.
Se puede implementar el proyecto actual en un establecimiento, como sistema de seguridad, por menos de **1300 euros**, teniendo como base el ordenador indicado en la

tabla superior. El precio se puede reducir drásticamente implementando mejoras en el proyecto y enfocado a otra posible plataforma o hardware propio.

El segundo supuesto, para la investigación estadística, es más complejo. Requiere de una mayor cualificación y de invertir horas en la adecuación del proyecto actual a un entorno en la nube.
Arriba se indicaba un coste superior a **3000 euros** debido a la necesidad de ese aprendizaje extra, pero, si contamos de base con los conocimientos necesarios, el coste del proyecto se puede reducir drásticamente, es decir, solo sería necesario crear una instancia en al nube y conectarla con nuestro nodo modificado Ocam.py. Esto se podría realizar en pocas horas teniendo los suficientes conocimientos, por lo que el coste se puede reducir a menos de **100 euros.**

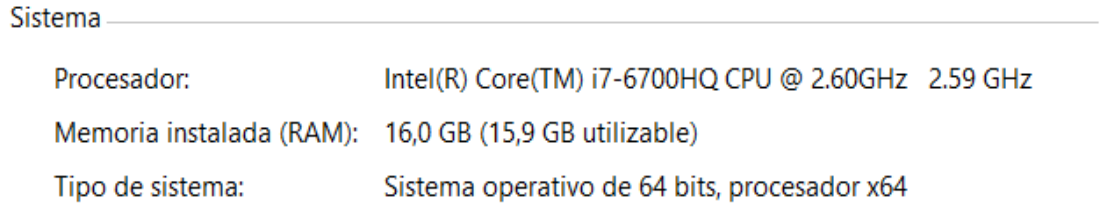
En general se ha generado un presupuesto teórico, en base a precios aproximados, ya que es difícil establecer un presupuesto exacto sin tener en cuenta todas las posibles combinaciones de software y hardware que se podría realizar en ambos supuestos.

Capítulo 8 Apéndice: Instalación y configuración

8.1 Configuración del sistema

En esta sección se va a explicar las herramientas que necesita tener instalada la máquina Ubuntu para el correcto funcionamiento de este proyecto, así como aclaraciones para su optimización, más adelante, se explicará la instalación y funcionamiento de estas, más detalladamente.
Antes de explicar las herramientas necesarias, se va tener en cuenta un par de consideraciones:

- El proyecto se ha desarrollado, como se nombra más arriba en el informe, sobre el siguiente hardware.



  El proyecto también puede funcionar sobre un hardware inferior.

- El sistema operativo utilizado es Ubuntu 18.04.
- Es **MUY** recomendable realizar el proyecto sobre **Python virtualenv**, ya que, al instalar las diferentes herramientas, pueden haber errores de compatibilidades con otros proyectos que pueda tener la máquina host, para ello, es muy recomendable aislar recursos como librerías y el entorno de ejecución, del sistema principal o de otros entornos virtuales.

8.2 Instalación python virtualenv

Para crear instalar python virtualenv, podemos seguir los pasos en el siguiente enlace: https://gist.github.com/Geoyi/d9fab4f609e9f75941946be45000632b

Además, en ese enlace instalaremos PIP, herramienta necesaria para instalar otros componentes del proyecto.

Hay que tener en cuenta, que si vamos a usar Python3, es necesario cambiar la configuración de ROS, ya que por defecto ejecutará Python.

8.3 Instalación ROS Melodic

Para instalar ROS Melodic, podemos utilizar la información suministrada por la web oficial: http://wiki.ros.org/melodic/Installation/Ubuntu

Es **muy importante** tener claros qué módulos de ROS se van a instalar, ya que hay algunos que solo son compatibles con ciertas versiones de ROS, por ejemplo, en nuestro proyecto, el módulo web_video_server no tiene soporte para la versión Lunar, tal y como vemos:

http://wiki.ros.org/web_video_server

Algunos módulos, aunque no sean compatibles con alguna versión de ROS se pueden implementar, si se sabe que dependencias fallan y se arreglan de forma manual.

Una vez instalado ROS, es recomendable, si no se tiene experiencia previa, seguir los siguientes tutoriales, concretamente del punto 1 al 7.

http://wiki.ros.org/ROS/Tutorials

8.4 Instalación Selenium

Para instalar selenium, podemos encontrar la información necesaria en :
https://selenium-python.readthedocs.io/installation.html

Será necesario, instalar el geckodriver, que básicamente nos permite operar con el navegador Firefox, dicho geckodriver se puede descargar en:

https://github.com/mozilla/geckodriver/releases

Ahora vamos a tener en cuenta el código de nuestro proyecto en :
https://github.com/alu0100812534/TFG/blob/master/catkin_ws/src/tfg/src/selenium/sele.py

En la línea, **browser = webdriver.Firefox(....)** vamos a tener en cuenta que el path debe ser ruta absoluta del fichero geckodriver, y para evitar problemas de permisos es recomendable guardar dicho geckodriver en la ruta donde está nuestro fichero sele.py .

Además, durante nuestro proyecto, nos dimos cuenta que los anuncios en los vídeos podrían ser un problema, ya que no aportan información útil al objetivo del proyecto, para ello instalamos adblock mediante Selenium, para que cada vez que lo ejecutaremos se instale, para ello se descargó el fichero .xpi, dándole a “guardar como” en donde pone Firefox:
https://adblockplus.org/en/download
Dicho ruta absoluta del fichero .xpi se incorporó al código mediante la línea **browser.install_addon(.....)**

Esto parecía funcionar bien inicialmente, pero, a los pocos segundos de abrir el navegador los vídeos extraídos se paraban, y el problema es que, al instalar el adblock al abrir el navegador con Selenium, la url principal de donde extraemos el vídeo pasa a segundo plano, es decir, pierde el foco,al abrirse la url de instalación del adblock.
Para solventar este inconveniente, se añadieron las siguientes líneas

current_window = browser.current_window_handle
browser.switch_to_window(current_window)

Esto, nos devuelve a la pestaña principal de donde queremos extraer la información, y por tanto el foco vuelve a estar donde debe, además, se le puso un sleep para que diera tiempo que la instalación de adblock terminara y generase la nueva pestaña.

8.5 Instalación CUDA

Cuda, sin entrar en la definición formal, una herramienta de Nvidia que permite aprovechar el potencial de sus tarjetas gráficas para, entre otras cosas, machine learning. Las tarjetas gráficas tienen la características que están diseñadas para procesamiento en paralelo, por lo tanto son muy útiles para entrenamiento de redes neuronales.

A la hora de instalar CUDA, hay que tener claro que tarjeta o tarjetas gráficas tenemos en nuestro ordenador y ver la compatibilidad con CUDA. Podemos usar el siguiente enlace para comprobarlo:

https://www.geforce.com/hardware/technology/cuda/supported-gpus

Para una información más extendida, tenemos el siguiente enlace:

https://developer.nvidia.com/cuda-gpus

Podemos instalar CUDA desde el siguiente enlace, para Ubuntu 18.04:

https://www.pugetsystems.com/labs/hpc/How-to-install-CUDA-9-2-on-Ubuntu-18-04-1184/

Nota: La versión del enlace es la 9.2 de CUDA, hay que comprobar que nuestro framework (ya sea Tensorflow u otro) sea compatible con esa versión y no de problemas de compatibilidad, ya que, hay ciertos problemas relacionados con CUDA 9.0 que en algunos casos obliga a instalar la versión 8.0, por eso es importante revisar la compatibilidad e incluso investigar su estabilidad con dicho framework de creación de redes neuronales.

8.6 Instalación Caffe, datasets y referencias de la red del proyecto

Para nuestro proyecto, se ha usado una red ya pre-entrenada en Caffe,de la cual podemos tener información y descargar en el siguiente enlace:

https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/

El código de dicha dirección web, está pensado para imágenes, por lo que se han realizado varios cambios para adaptarlo a vídeos, así como que funcione en conjunto al framework ROS.

Antes de seleccionar esa red, se intentó implementar una desde cero, y se realizó una investigación para ello, pero, al final debido a que entrenar una red desde el principio podía consumir mucho tiempo y al final no ser del todo óptima (además de necesitar grandes cantidades de imágenes), se decidió usar la anterior red como núcleo del fichero Ocam.py. De todos modos, se indicará en esta sección los resultados de dicha investigación, entre ellos como instalar y entrenar una red Caffe.

Para instalar la red caffe, podemos seguir la información que nos suministra el siguiente enlace:

http://caffe.berkeleyvision.org/install_apt.html

Al igual que se indicará en la sección de Tensorflow, la red puede ser instalada de dos formas, CPU-only o Cuda versión.

Como se describió anteriormente, CUDA está pensado para aprovechar el potencial de las GPUs Nvidias.

Para este proyecto, se instaló la versión de CUDA y se realizaron pruebas, para ello, los siguientes enlaces pueden ser de utilidad:

http://caffe.berkeleyvision.org/gathered/examples/imagenet.html
https://chunml.github.io/ChunML.github.io/project/Training-Your-Own-Data-On-Caffe/

Lo anterior, es para entrenar la red con tus propias imágenes. Lo recomendable, es usar un dataset de imágenes, ya que poseen una gran cantidad de imágenes ya listas para ser

procesadas por redes neuronales, de lo contrario, tratar nuestras propias imágenes puede ser bastante laborioso. Durante el desarrollo de la investigación, se encontraron los siguientes datasets:

http://cocodataset.org/#home
http://deeplearning.net/datasets/

Concretamente, se probó a descargar los datasets de COCO. Para pasarlos a nuestra red Caffe, podemos seguir el siguiente enlace:

https://xiaoyuliu.github.io/2018/02/23/combine-datasets-into-lmdb-for-caffe/

Otra opción que se planteó en la investigación de redes neuronales, fue mejorar la red seleccionada para el proyecto (la primera indicada en esta sección). Para esto, entramos en otro concepto llamado “Fine-Tuning”, del cual tenemos más información en los siguientes enlaces:

https://www.quora.com/How-do-I-fine-tune-a-Caffe-pre-trained-model-to-do-image-classification-on-my-own-dataset

http://caffe.berkeleyvision.org/gathered/examples/finetune_flickr_style.html

Para realizar nuetro proyecto, debemos tener una versión de OpenCV que cuente con las últimas incorporaciones en el campo de la Inteligencia Artificial, como en neustro caso, el fichero Ocam.py (https://github.com/alu0100812534/TFG/blob/master/catkin_ws/src/tfg/src/opencv_cam/Ocam.py) debe poseer la función **cv2.dnn.readNetFromCaffe(...).** Por esto, se debe instalar la última versión disponible de openCV, la podemos encontrar aquí:

https://docs.opencv.org/3.4/d2/de6/tutorial_py_setup_in_ubuntu.html

Nota: en ese mismo enlace, en la parte superior izquierda podemos seleccionar la versión a instalar.

8.7 Instalación Tensorflow

Cuando se inició el proyecto, y se buscó información respecto a la instalación de Tensorflow, la verdad es que la documentación oficial no era del todo clara, y obligaba a instalar numerosas dependencias que podían ocasionar errores.
Actualmente, Google ha cambiado la documentación de su web oficial, haciéndola más clara y precisa, la podemos encontrar aquí:

https://www.tensorflow.org/install

Para “Fine-Tuning” una red en Tensorflow, el siguiente enlace es de utilidad:

https://www.quora.com/How-do-I-fine-tune-a-pre-trained-CNN-model-in-TensorFlow-to-do-image-classification-on-my-own-data

8.8 Instalación Xampp

Como se ha indicado en el documento, para interactuar a nivel de usuario, la salida de los tópicos del Ocam.py son mostrados a través de un servidor, así como las alertas de detección.

Para esto, se ha instalado un servidor local Xampp, a partir de la siguiente información:

https://simplecodetips.wordpress.com/2018/06/14/instalacion-de-xampp-en-ubuntu-18-04/

Para ejecutarlo vale con ir a la ruta opt/lampp y ejecutar sudo start ./xampp

Si queremos que nuestro servidor local pase a ser público, podemos seguir el siguiente tutorial:

https://stackoverflow.com/questions/822902/access-xampp-localhost-from-internet

Tal y se cuenta en el enlace, será necesario tener una IP estática, que se puede contratar con la compañía de internet que tengamos.

Capítulo 9 Apéndice: Módulo Web_Server

9.1 Cómo incrustar un video web a partir de un tópico, mediante el módulo web_video_server

En nuestro proyecto, se ha usado el módulo web_video_server para ver en tiempo real lo que está transmitiendo un tópico, para ello se han seguido los siguientes pasos de instalación y puesta en marcha:

0) Nos vamos a la carpeta de código de nuestro workspace

```
$> roscd && cd ../src
```

1) Descargamos el módulo correspondiente a web_video_server.

```
$> git clone https://github.com/RobotWebTools/web_video_server.git
$> git clone https://github.com/GT-RAIL/async_web_server_cpp
```

2) Instalamos el módulo necesario para leer de la cámara

```
$> sudo apt-get install ros-kinetic-cv_camera
```

O bien, si lo anterior no funciona:

```
$> git clone https://github.com/OTL/cv_camera.git
```

4) Compilamos

```
$> roscd && cd .. && catkin_make
```

5) Arrancamos la cámara

```
$> rosparam set cv_camera/device_id 0  # El dispositivo puede cambiar en función de la cámara que empleemos como entrada
$> rosrun cv_camera cv_camera_node
```

6) Arrancamos el servidor web:
`$> rosrun web_video_server web_video_server`

7) Si abrimos el navegador en http://localhost:8080, veremos la lista de streams disponible. Ahora podemos usar la etiqueta <img> de HTML para ver dicho stream, que tendrá una ruta como esta : http://localhost:8080/stream?topic=/cv_camera/image_raw.

Bibliografía

Los enlaces a continuación, han aportado contenido al desarrollo de cada sección de este informe.

Antecedentes y estado actual

ROS

https://es.wikipedia.org/wiki/Sistema_Operativo_Rob%C3%B3tico

http://www.clearpathrobotics.com/assets/guides/ros/_images/ros101four.png

Selenium

https://www.seleniumhq.org

Redes neuronales artificiales

https://es.wikipedia.org/wiki/Red_neuronal_artificial

https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax?hl=es-419

https://www.tensorflow.org/

https://pytorch.org/

http://caffe.berkeleyvision.org/

https://u01.appmifile.com/images/2018/06/30/8c014fac-1c8f-4c14-94bf-13773f3ac76e.png

Reconocimiento de regiones

https://es.wikipedia.org/wiki/Reconocimiento_de_regiones

Bot Telegram

https://gist.github.com/dlaptev/7f1512ee80b7e511b0435d3ba95d88cc

Apéndices

Los enlaces nombrados en el apéndice, de los cuales también se ha extraído información para desarrollar el proyecto en su totalidad.

https://gist.github.com/Geoyi/d9fab4f609e9f75941946be45000632b

http://wiki.ros.org/melodic/Installation/Ubuntu

http://wiki.ros.org/web_video_server

http://wiki.ros.org/ROS/Tutorials

https://selenium-python.readthedocs.io/installation.html
https://github.com/mozilla/geckodriver/releases
https://github.com/alu0100812534/TFG/blob/master/catkin_ws/src/tfg/src/selenium/sele.py
https://adblockplus.org/en/download
https://www.geforce.com/hardware/technology/cuda/supported-gpus

https://developer.nvidia.com/cuda-gpus
https://www.pugetsystems.com/labs/hpc/How-to-install-CUDA-9-2-on-Ubuntu-18-04-1184/
https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/
http://caffe.berkeleyvision.org/install_apt.html
http://caffe.berkeleyvision.org/gathered/examples/imagenet.html
https://chunml.github.io/ChunML.github.io/project/Training-Your-Own-Data-On-Caffe/
http://cocodataset.org/#home
http://deeplearning.net/datasets/
https://xiaoyuliu.github.io/2018/02/23/combine-datasets-into-lmdb-for-caffe/
https://www.quora.com/How-do-I-fine-tune-a-Caffe-pre-trained-model-to-do-image-classification-on-my-own-dataset
http://caffe.berkeleyvision.org/gathered/examples/finetune_flickr_style.html
https://www.tensorflow.org/install

https://www.quora.com/How-do-I-fine-tune-a-pre-trained-CNN-model-in-TensorFlow-to-do-image-classification-on-my-own-data
https://simplecodetips.wordpress.com/2018/06/14/instalacion-de-xampp-en-ubuntu-18-04/
https://stackoverflow.com/questions/822902/access-xampp-localhost-from-internet
https://github.com/RobotWebTools/web_video_server.git
https://github.com/GT-RAIL/async_web_server_cpp