Trabajo de Fin de Grado en Ingeniería de las Tecnologías de la Telecomunicación

Implementación de sistema de videovigilancia con YOLOv7 y Amazon Web Services.

Autor: Francisco Berdejo Arenas

Tutor: Antonio Jesús Sierra Collado

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo de Fin de Grado
en Ingeniería de las Tecnologías de la Telecomunicación

Implementación de sistema de videovigilancia con YOLOv7 y Amazon Web Services.

Autor:
Francisco Berdejo Arenas

Tutor:
Antonio Jesús Sierra Collado
Profesor

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Trabajo Fin de Grado: Implementación de sistema de videovigilancia con YOLOv7 y Amazon Web Services.

Autor: Francisco Berdejo Arenas

Tutor: Antonio Jesús Sierra Collado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigos y compañeros

Agradecimientos

Quería agradecer en primer lugar a toda mi familia por haberme apoyado siempre en mis decisiones y haberme ayudado a crecer como persona y llegar a donde estoy ahora mismo.

También quería dedicárselo a todos mis compañeros que he conocido a lo largo de estos años en mi carrera universitaria y a mi pareja. Ya está hecho, hemos sobrevivido.

Y por último agradecer a mis profesores y a la ETSI por la formación obtenida.

Gracias.

Francisco Berdejo Arenas

Sevilla, 2023

Resumen

Nos encontramos en pleno auge de las **Inteligencias Artificiales,** donde hemos visto avances revolucionarios en campos como la **Visión por Ordenador**, **Procesamiento Natural del Lenguaje**, **Machine Learning** entre otros. Las utilizamos en nuestro día a día en aplicaciones como ChatGPT, programas de edición de fotos (Photoshop), asistentes domésticos virtuales (Siri, Alexa) o sin darnos cuenta en servicios de recomendación de contenidos y publicidad en páginas de compras por internet y redes sociales (Amazon, Instagram).

Es por esto que para este proyecto se ha decidido implementar una solución que hace uso del modelo de detección de objetos en tiempo real **YOLOv7** junto a la infraestructura de **computación en la nube Amazon Web Services**. La aplicación realiza la función de **sistema de videovigilancia** con una infraestructura hibrida entre AWS realizando las funciones de notificación y un servidor local que procesará las imágenes con Yolov7 y activará las notificaciones.

Una vez se realice una detección en el servidor local, se notificará al usuario mediante mensaje SMS y correo electrónico con una captura de la detección. Finalmente se almacenará en clip de vídeo la grabación de todo lo ocurrido mientras esté detectando alguna cámara para que el usuario pueda revisar los vídeos y ver lo sucedido.

Se detallará los servicios utilizados, su implementación, la puesta en marcha y como se integran entre ellos. Se explicarán los componentes que forman parte de la aplicación junto a ejemplos y pruebas de funcionamiento. Finalmente se hará un estudio del rendimiento de diferentes modelos de Yolov7 con distinto hardware para saber cómo dimensionar la cantidad de cámaras que se pueden conectar al mismo tiempo. Y se hará una valoración de los objetivos conseguidos y líneas futuras a implementar.

Abstract

We are currently in the middle of the **Artificial Intelligence** boom, where we have seen revolutionary advancements in fields such as **Computer Vision, Natural Language Processing, Machine Learning**, among others. We use them in our day-to-day lives through applications like ChatGPT, photo editing programs like Photoshop, virtual home assistants like Siri and Alexa, or unknowingly through content recommendation and advertising services on e-commerce websites and social media platforms like Amazon and Instagram.

That's why, for this project, it has been decided to implement a solution that utilizes the real-time object detection model **YOLOv7**, along with the **Amazon Web Services cloud computing infrastructure**. The application functions as a **video surveillance system** with YOLOv7 as the intruder detector, employing a hybrid infrastructure between AWS being responsible of the notification process and Yolov7 the one that process all images in search of a detection at a local server.

Once detection is made user will be notified by SMS and by email with a screenshot of the detection. Finally, a video will be saved with the recordings of what happened while the cameras were detecting, and user could check those videos at the local server.

We will provide a detailed description of the services used, their implementation, setup and integration between them. We will explain all the components that made the application with examples. There will be a study of performance of different Yolov7 models and hardware to see how many cameras could handle at the same time depending in which hardware is running. Finally, we will check if all goals were satisfied and let some ideas for future works in this project.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

La educación es el pasaporte hacia el futuro, el mañana pertenece a aquellos que se preparan para él en el día de hoy.

- Malcolm X -

En esta sección se dará una breve introducción al mundo de la Inteligencia Artificial (IA). Se explicará resumidamente los diferentes campos de estudio dentro de esta, junto algunos ejemplos del mundo real. Con esta visión general se procederá a enmarcar donde se encuentra YOLOv7 dentro de los campos de la IA y las funciones que realiza y los casos de uso más frecuentes para lo que es utilizada.

Se detallarán los objetivos y requisitos a satisfacer en el proyecto. Una vez establecidos los objetivos, se introducirá el escenario propuesto. En los siguientes capítulos de este documento se detallará en profundidad el funcionamiento y detalles más técnicos de cada parte.

Se hará un recorrido por los trabajos realizados en el departamento que tengan relación con este. Y se mostrarán los elementos diferenciadores de este trabajo respecto a los demás.

Para terminar, se proporcionará una descripción de la estructura de este documento con los principales puntos que se encontrarán en este.

1.1 Introducción a la Inteligencia Artificial

Podría compartir diversas definiciones, como la del profesor John McCarthy de la Universidad de Stanford en su artículo WHAT IS ARTIFICIAL INTELLIGENCE? (1). Pero para esta ocasión prefiero compartir la definición creada por una propia inteligencia artificial, ChatGPT: *"La inteligencia artificial (IA) es un campo de estudio y desarrollo que busca crear sistemas y programas capaces de imitar o mejorar la inteligencia humana en tareas complejas. Utilizando algoritmos y modelos matemáticos, la IA permite a las máquinas procesar datos, aprender de la experiencia, reconocer patrones y tomar decisiones. El aprendizaje automático, el procesamiento del lenguaje natural y la visión por computadora son áreas clave dentro de la IA. La IA se aplica en diversos campos como medicina, industria, robótica y servicios financieros, automatizando tareas, optimizando procesos y mejorando la toma de decisiones. Sin embargo, la IA aún enfrenta desafíos en términos de adquirir un verdadero entendimiento humano y aborda cuestiones éticas relacionadas con la privacidad y el sesgo algorítmico."* (2)

Dentro del mundo de la IA existen diferentes campos o áreas de trabajo centrados en diferentes tareas en las que un programa convencional no es capaz de desenvolverse, o al menos no con la calidad con la que una IA puede hacerlo. Aunque no hay un consenso global de los campos ya que están en constante evolución y muchos campos son dependientes de otros o tienen cosas en común, podemos identificar estos como los principales y los más populares:

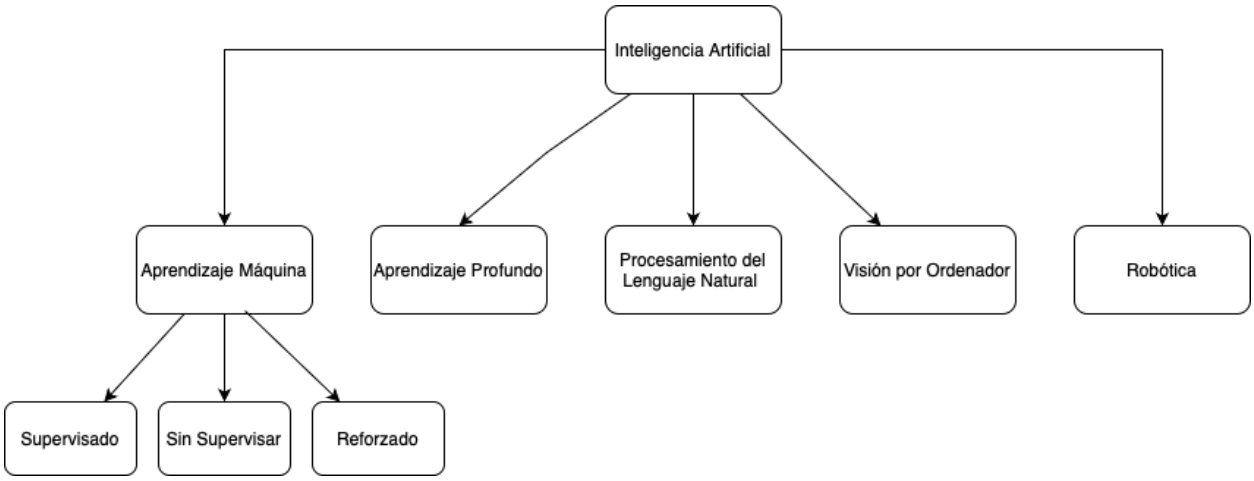


Figura 1 Principales campos de la IA

- **Aprendizaje máquina (Machine Learning):** Probablemente la más popular y en gran medida porque es la que establece los cimientos en los que se sustentan el resto de las categorías, gran parte de ellas podrían englobarse dentro de esta. Este campo se dedica al desarrollo de algoritmos y modelos estadísticos que imitan el aprendizaje humano. Esto se consigue mediante un gran número de datos que permiten al modelo identificar y reconocer patrones mediante un entrenamiento. Una vez entrenado el modelo puede emplearse para realizar predicciones.

  Los conjuntos de datos empleados para esta tarea pueden llegar a ser enormes y uno de los principales desafíos de la IA es obtener estos sets de datos. Normalmente solo las grandes empresas como Google, OpenAI, Amazon o instituciones como grandes universidades cuentan con la cantidad de datos para poder entrenar modelos grandes. Aunque hoy en día ya se pueden encontrar grandes sets de datos en internet ofertados para poder ser usados.

  No todos los modelos utilizan el mismo tipo de datos para su entrenamiento. Podemos distinguir tres tipos de entrenamiento diferentes según el tipo de datos empleados:

  - **Supervisado:** Utilizan sets de datos clasificados y etiquetados previamente por humanos. Una vez están clasificados se dividen en dos conjuntos. Uno contendrá los datos para el entrenamiento y será los que utilice el modelo para analizarlos y aprender. Y una pequeña porción que se empleará únicamente para poner a prueba el modelo y ver cuan efectivo es, nunca para entrenamiento. Un caso de uso pueden ser modelos de clasificación de imágenes.
  - **Sin supervisar**: El programa busca por patrones y similitudes entre todo el conjunto de datos sin clasificar, incluso aunque dichos patrones no sean explícitos. Se suele emplear en modelos que buscan tendencias dentro de un set de datos o información oculta.
  - **Reforzado**: En este caso se utiliza el mecanismo de prueba y error. Se establece un sistema de recompensa en el que un agente interactúa con el entorno y cuando se toma una acción correcta se le otorga una "recompensa". Con esta retroalimentación se intenta que el sistema maximice la toma de decisiones correcta en busca de recompensa. Se suele emplear para modelos de conducción autónoma o videojuegos

- **Redes neuronales y aprendizaje profundo (Deep Learning):** Las redes neuronales son células o nodos organizados en capas. Cada célula de una capa está interconectada con las células de la capa anterior y siguiente. Las células se encargan de recibir entradas, procesarlas mediante una función de activación y producir una salida. Esta salida se transmite a otra célula mediante una conexión la cual tiene un peso, dicho peso determina la influencia de la célula.

  El entrenamiento se encarga de optimizar y ajustar el peso de cada conexión para determinar patrones y obtener resultados, comparando los resultados a la salida con los esperados. Están compuestas por una capa de entrada y otra de salida además de las llamadas capas ocultas que se encuentran en medio. Estas redes pueden no tener ningún tipo de retroalimentación, en este caso la información circula de la entrada a la salida. O tener la salida interconectada a la entrada lo que le permite tener memoria y procesar datos con dependencias temporales.

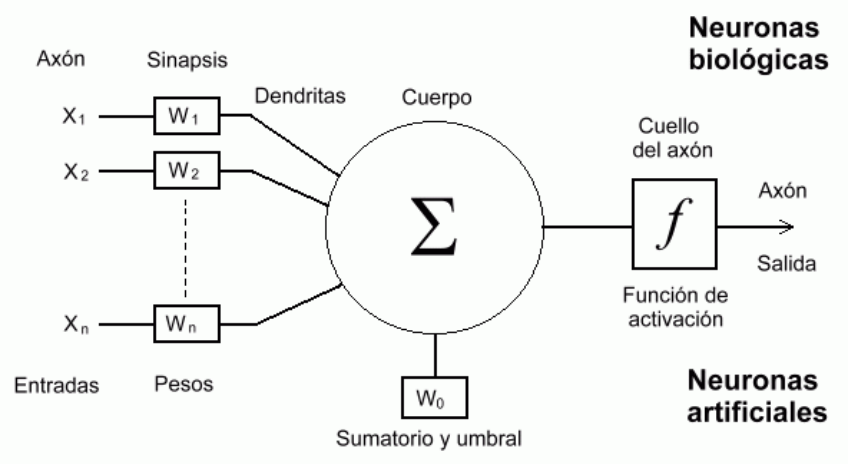


Figura 2 Comparación neurona humana con artificial (3)

- **Procesamiento del lenguaje natural (Natural Lenguage Processing, NLP):** Este campo es uno de los que ha causado mayor impacto en la sociedad hasta el momento de redactar este documento, con herramientas como ChatGPT de OpenAI (4) o Google Bard (5). Se centra en adoptar, identificar e imitar la capacidad humana de hablar y escribir. Es por esto por lo que es realmente útil para asistentes o bots de conversación. Utiliza técnicas de aprendizaje maquina mencionadas anteriormente para su entrenamiento.
- **Robótica:** Este campo se complementa junto a la robótica tradicional para la creación de sistemas autónomos que pueden realizar tareas en el mundo real. Los casos de uso más frecuentes pueden darse en cadenas de montaje, fabricas o centros logísticos. Donde el uso de la IA proporciona un mayor nivel de entendimiento del entorno y capacidad de reacción a este.
- **Visión por ordenador (Computer Vision):** Este ámbito de estudio se encarga de la percepción, interpretación y comprensión de imágenes y vídeos. Existen varios casos de uso de la visión por ordenador que son ampliamente adaptados y utilizados en nuestro entorno. Entre los cuales podemos encontrar:

  Detección de objetos: La capacidad de detectar objetos dentro de imágenes y vídeos. Dentro de este campo se encuentra YOLOv7. Las aplicaciones que hacen uso de detección de objetos pueden ser tantos como la imaginación y creatividad de cada persona. Goza de gran popularidad dentro de los sistemas de videovigilancia, diagnosis médica, reconocimiento de señales de tráfico, lector de matrículas, reconocimiento de lugares y edificios históricos, control de afluencia de personas y tráfico junto a un largo etcétera.

  Algunos de los algoritmos más utilizados son YOLO, Single Shot Detector, R-CNN o RetinaNet. Todos cuentan con ventajas e inconvenientes que serán estudiados con mayor detalle en los siguientes capítulos de este trabajo.

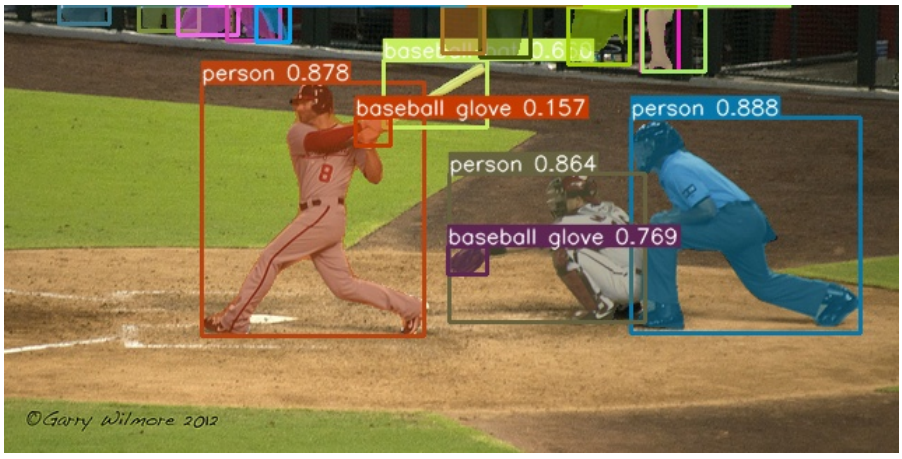


Figura 3 Ejemplo de una imagen procesada por YOLOv7

Reconocimiento facial: Como su nombre indica los mecanismos de reconocimiento facial hacen uso de este tipo de IA para detectar rasgos únicos de nuestro rostro e identificarnos. Combinado junto a otros métodos de seguridad se trata de una de las formas de autenticación mas seguras que existen hasta el momento.

Segmentación: Se trata de la identificación de cada píxel de una imagen para delimitar silueta de los objetos detectados. Se utiliza mucho en cine donde es de gran utilidad para la creación de efectos especiales.

Reconocimiento de texto en imágenes: En la actualidad la podemos ver implementada en la mayoría de los dispositivos inteligentes y otras herramientas de nuestro día a día para extraer texto a partir de una imagen o vídeo.

Superresolución y edición de fotografía: Al igual que la anterior este tipo de IA están integradas en multitud de herramientas de edición de fotografía y vídeo. Desde la restauración de vídeos antiguos, edición de fotografías hasta herramientas más sofisticadas como DLSS de Nvidia (6).

1.2 Objetivos

En el departamento de Ingeniería Telemática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla se han realizado diferentes trabajos relacionados con Inteligencia Artificial y con Yolo en varias de sus versiones. Dichos trabajos han servido como punto de partida a la hora de decidir la aplicación que se va a llevar a cabo, así como donde se iba a ejecutar.

La mayor parte de los trabajos presentados se basan en versiones de Yolo 3 y 5. Y son ejecutados en dispositivos con bajo poder de cómputo como Raspberry Pi. Como el Trabajo Fin de Grado de Ángel Moreno Prieto titulado "Detección de Objetos con TinyYOLOv3 sobre Raspberry Pi 3" (18) y el Trabajo Fin de Grado de Luis Muñiz García titulado "Viabilidad y rendimiento de YOLOv5 en Raspberry Pi 4 modelo B" (19)

En el TFG de Ángel ha demostrado obtener resultados en la velocidad de inferencia insatisfactorios para aplicaciones en tiempo real. Nos muestra una media de 0,132 FPS usando Yolov3 en una Raspberry Pi 3 usando imágenes de baja resolución a 448x448.

El TFG de Luis nos muestra que realizar el entrenamiento en un equipo de bajo rendimiento como Raspberry Pi 4B de un modelo Yolov5 tarda cerca de 94 horas frente a los escasos 9 minutos de utilizar hardware más potente como los que proporciona Google Colab.

Por otro lado en el TFG de Alberto García Hernández titulado "Ajuste de la eficiencia en redes neuronales para la detección de señales de tráfico. Comparativa de rendimiento de Yolov-3 y EfficientDet." (20) nos muestra una velocidad de inferencia de 0,89 segundos o 1,12 FPS utilizando algún tipo de instancia en Google Colab no especificada con Yolov3.

Tras los resultados anteriores vemos que no es viable implementar aplicaciones que necesiten procesamiento en tiempo real bajo cierto hardware. Por lo que ha servido como motivación para este trabajo implementar una aplicación que haga uso de Yolov7 y de realizarlo en un hardware más potente para comprobar si con las nuevas versiones de Yolo y hardware especifico resulta viable una aplicación en tiempo real.

A parte de los trabajos anteriormente comentados se ha observado una tendencia a utilizar este tipo de modelos para la conducción autónoma y reconocimiento de señales. Por lo que para destacar este trabajo del resto se ha decidido orientar la aplicación a un sistema de videovigilancia que sea funcional. A demás de implementar otras tecnologías como los servicios de AWS, haciendo que su estructura y funcionamiento se asemeje de forma parecía a las soluciones comerciales ya existentes en el mercado.

Como principal objetivo a conseguir en este proyecto es realizar una aplicación de videovigilancia que sea funcional. Para ello será necesario que realice dos funciones imprescindibles, la detección y la notificación. Se desean que ambas funciones sean independientes y desacoplada una de otra.

La función de detección deberá de ser capaz de analizar las imágenes obtenidas de cámaras en busca de personas y mandar una petición de notificación cuando se detecte alguien. Del mismo modo, guardará en vídeo la grabación de lo ocurrido mientras se esté detectando.

La función de notificación ha de ser implementada "en la nube" bajo algún proveedor de servicios tal como Amazon Web Services, Azure o Google Cloud. Ha de aceptar peticiones de notificación desde la función de notificación e implementar algún método de seguridad para que solo pueda ser activada por nuestra aplicación y no por agentes externos.

Para poder dimensionar la aplicación, conocer el rendimiento y saber si es capaz de funcionar en tiempo real se deberá realizar diversas pruebas de rendimiento con distinto hardware y número de cámaras. Así se obtendrá cuantas cámaras simultaneas es capaz de analizar la aplicación y poder conocer si es funcional para distintos casos de uso.

Para terminar se ha querido añadir una serie de requisitos para poder favorecer la replicación de la aplicación para líneas futuras de alumnos que quieran ampliar este proyecto. La primera es mantener el coste de uso al mínimo. Utilizar servicios de proveedores como AWS en ocasiones conlleva gastos y puede resultar a ser costosos, por lo que se tendrá en cuenta las capas de uso gratuitas que ofertan estos proveedores y se hará el uso de estas en lo máximo posible.

El segundo requisito es que la aplicación ha de ser fácilmente replicable sin necesidad de utilizar hardware específico como tarjetas gráficas lo que puede ser muy costoso e incompatible con algunos sistemas.

1.3 Escenario propuesto

La solución que se ha llevado a cabo para cumplir con los objetivos y requisitos previos es implementar un sistema de videovigilancia con YOLOv7 como sistema principal de detección de personas y usar los servicios de AWS[1] para enviar las alertas y notificaciones. Para ello se hará uso de una arquitectura hibrida donde el análisis y procesamiento de las imágenes se harán en un servidor local. Mientras que en AWS se implementará los servicios de notificación por SMS[2] y por correo electrónico.

[1] Amazon Web Services. Servicio de computación en la nube de Amazon.
[2] Servicio de mensajes cortos, utilizado ampliamente por dispositivos móviles.

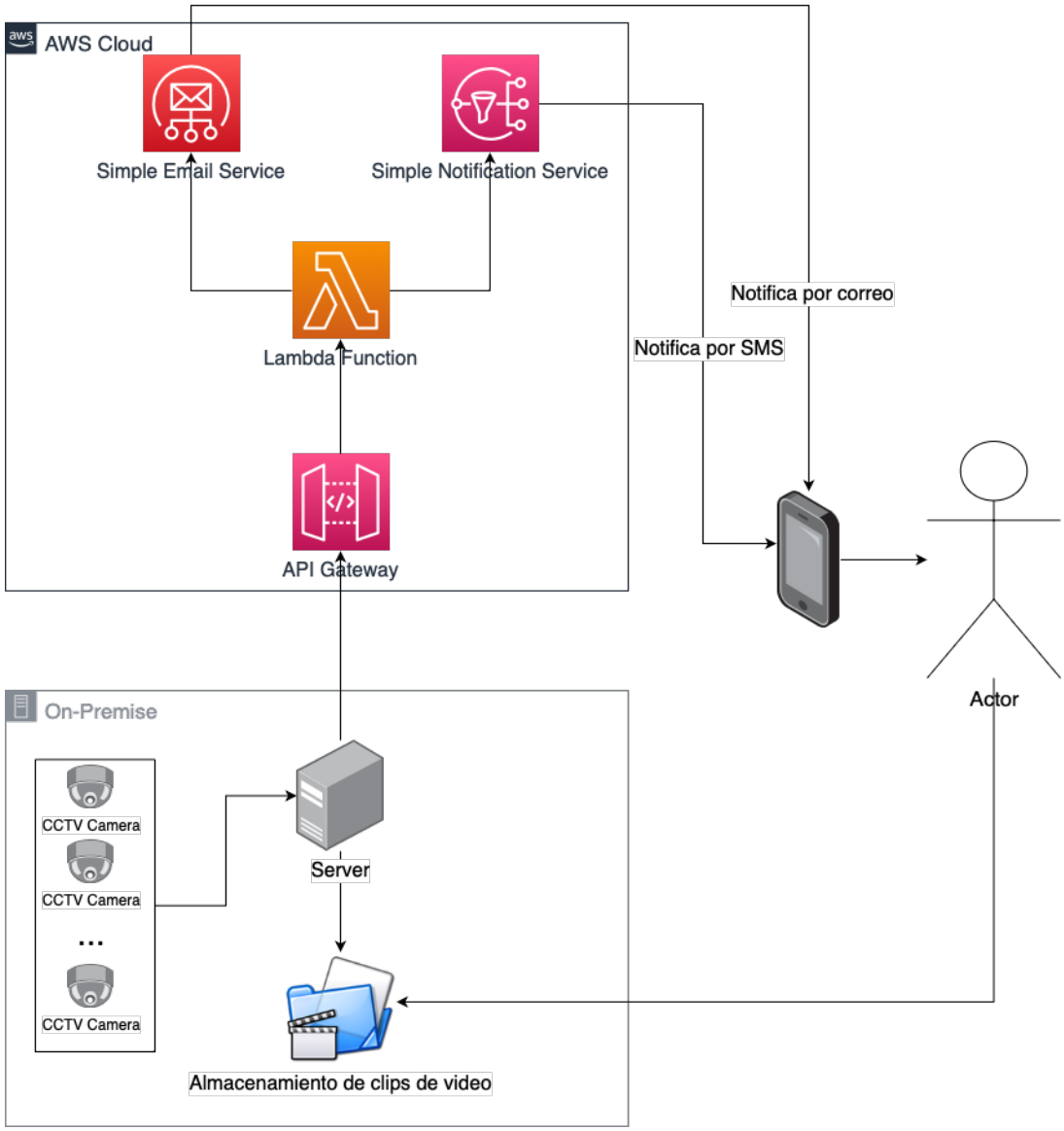


Figura 4 Diagrama Arquitectura de la Aplicación

Para que la aplicación sea funcional la principal limitación es la potencia de cómputo, por lo que un sistema como una Raspberry Pi no es viable, ya que para procesar varios flujos de vídeo a más de 15 fotogramas por segundo es necesario un hardware de procesamiento de vídeo como una tarjeta gráfica. Por eso se utilizará un ordenador personal con este hardware. Para obtener los flujos de vídeo se hará uso de una cámara de videovigilancia doméstica compatible con RTSP[3].

Estos flujos de vídeo son capturados por el servidor local y procesado en él. Cuando se realiza una detección de una persona, el sistema guardará localmente un clip de vídeo de todo lo que esté ocurriendo mientras se esté detectando al intruso. Al mismo tiempo se enviará una captura a los servicios de AWS los cuales se encargarán de enviar un correo electrónico con la captura de vídeo y un SMS al usuario alertándole de la detección.

Se ha optado por utilizar los servicios de API Gateway, Lambda Function, SNS y SES de Amazon Web Services para implementar las funciones de notificación en la nube. AWS cuenta con una capa de uso gratuita que abarca estos servicios. Hasta 1 millón de solicitudes Lambda al mes, 1 millón de publicaciones en SNS mensuales, 62.000 mensajes de correo mensuales en SES y 1 millón de llamadas a la API Gateway. Estos límites son más que suficientes para llevar a cabo el proyecto y tan solo se factura unos pocos céntimos al enviar SMS a números fuera de EE. UU.

[3] Protocolo de Transmisión en Tiempo Real. Utilizado para transmitir vídeo en directo.

1.4 Estructura

En los próximos capítulos de este documento nos enfocaremos en diferentes aspectos. Desde la elección de las tecnologías y hardware utilizado en comparación al resto de alternativas que existen, hasta la puesta en marcha y resultados obtenidos. Dichos capítulos son los siguientes:

- Estado del arte: Se llevará a cabo una comparativa de las tecnologías disponibles, los motivos por los que se ha elegido las empleadas y un vistazo a cómo funcionan.
- Aplicación: Se expondrá la configuración, puesta en marcha y funcionamiento de la aplicación. Se realizará una explicación paso a paso siguiendo el orden lógico y se mostrará los resultados de la ejecución.
- Pruebas de rendimiento: Se detallarán las pruebas realizadas, hardware utilizado, los resultados y los entornos donde se han llevado a cabo.
- Conclusiones: Para concluir se llevará a cabo una valoración de los objetivos cumplidos y resultados obtenidos además del planteamiento de posibles líneas futuras.

2 ESTADO DEL ARTE

Partiendo de las bases explicadas en el punto *1.1-Introducción a la Inteligencia Artificial* se llevará a cabo un estudio de los modelos de detección de objetos disponibles (funcionamiento, arquitectura y casos de uso), comparativas y los motivos por los que se optó por elegir YOLOv7.

Del mismo modo se realizará una breve comparativa entre los principales proveedores de computación en la nube, los servicios que ofrecen y las razones por las que se eligió a Amazon Web Services.

Se indicará el hardware utilizado para la implementación y los requisitos que se necesitan para el correcto funcionamiento.

2.1 Algoritmos de detección de objetos

Los algoritmos de detección de objetos llevan existiendo desde hace décadas y han ido evolucionando conforme el paso del tiempo. En la actualidad existen múltiples algoritmos para la detección de objetos. Pero los más famosos y utilizados son los siguientes:

- **R-CNN & Faster R-CNN** (7) (8)**:** Estos algoritmos realizan la detección de objetos en dos fases. La imagen se divide en regiones, basándose en características de la imagen de entrada mediante una red neuronal convolucional llamada Region Proposal Network en el caso de Faster R-CNN. Mientras que R-CNN utiliza un algoritmo denominado 'selective search' el cual forma alrededor de 2000 regiones.

  Una vez obtenidas las regiones se pasan por una red neuronal convolucional pre entrenada la cual produce un vector de características de dimensión 4096. Dicho vector contiene las características de los de la región como los objetos dentro de esta. Luego se clasifica cada vector mediante un SVM (Support Vector Machine) para clasificar la presencia de objetos en la región y finalmente se dibuja (de forma opcional) las cajas que enmarcan al objeto dentro de la imagen.

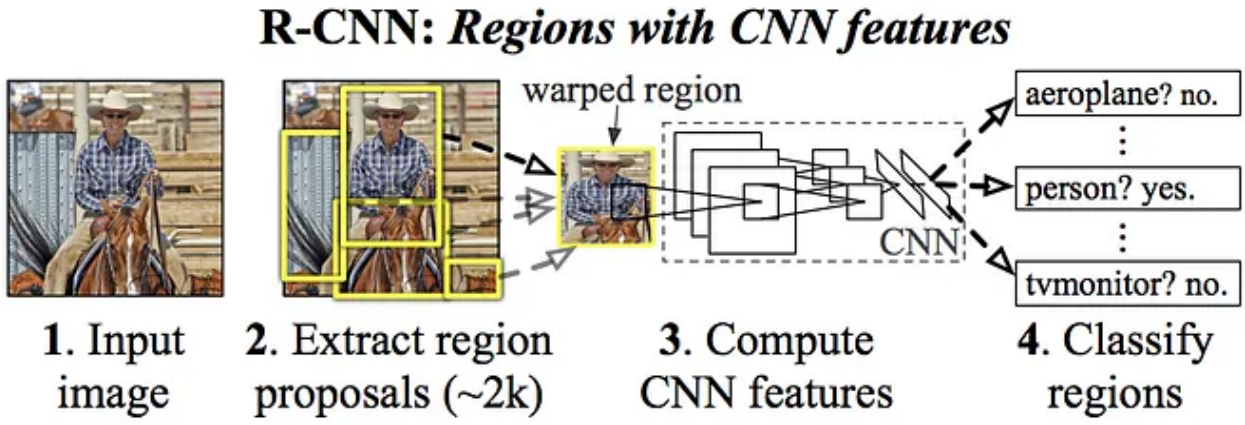


Figura 5 Funcionamiento R-CNN (7)

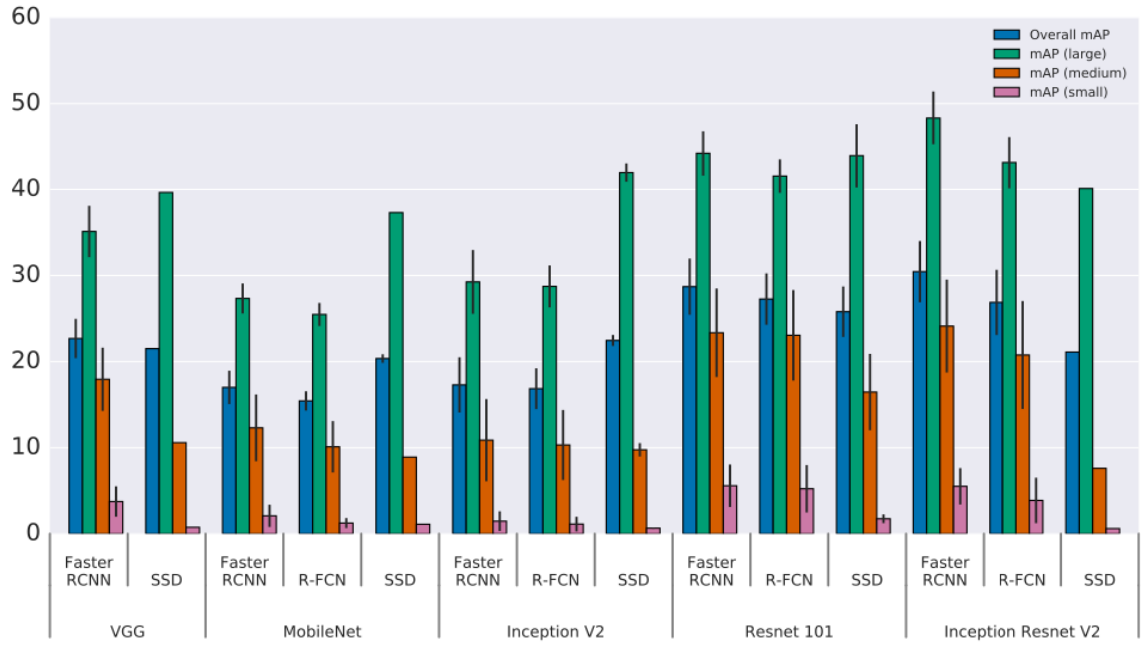
Aunque ofrecen buenos resultados y el tiempo de procesamiento ha disminuido bastante entre R-CNN y su versión más reciente Faster R-CNN, existen otros modelos más aptos para aplicaciones en tiempo real. En pruebas comparativas con SSD y YOLOv3 muestran un mayor nivel de precisión media promedio (mAP). Esto es debido a que son algoritmos de dos fases, es decir, primero realizan una preselección de posibles objetos en regiones y después se realiza otra pasada región a región para clasificar cada objeto.

Tabla 1 Comparación de precisión entre diferentes modelos de detección de objetos. (9)

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

También demuestra mejor desempeño en la detección de objetos de menor tamaño o que están muy próximos entre si. Factores a tener en cuenta para los casos de uso a los que se vaya a emplear el modelo.

Tabla 2 Precisión en la detección de objetos de tamaño pequeño de RCNN comparada con SSD y R-FCN (10)



- **SSD (Single Shot Detector)** (11)**:** Tal como el nombre indica SSD es un algoritmo que realiza la detección en una sola etapa al igual que YOLO y a diferencia de R-CNN vistos anteriormente. Esto hace que el tiempo requerido para procesar una sola imagen sea mucho menor al no tener que realizar la selección de regiones previas a la detección.

La imagen pasa a través de la red una sola vez. A media que avanza las capas van reduciendo progresivamente su tamaño (ver figura 8). El emplear capas de diferente tamaño permite realizar detecciones de objetos de diferente tamaño. Cada capa genera un mapa de características, el cual contiene una representación de lo que hay en dicha región junto a las coordenadas de las cajas delimitadoras relativas al mapa.

Esto hará que para un mismo objeto tengamos diferentes cajas superpuestas a distintos tamaños y formas dependiendo del mapa de características, (ver figura 7). Para finalizar se ajustan las cajas entre todas las candidatas al tamaño y forma del objeto y se aplica la técnica de supresión de no máximos para eliminar las cajas con baja probabilidad o repetidas.

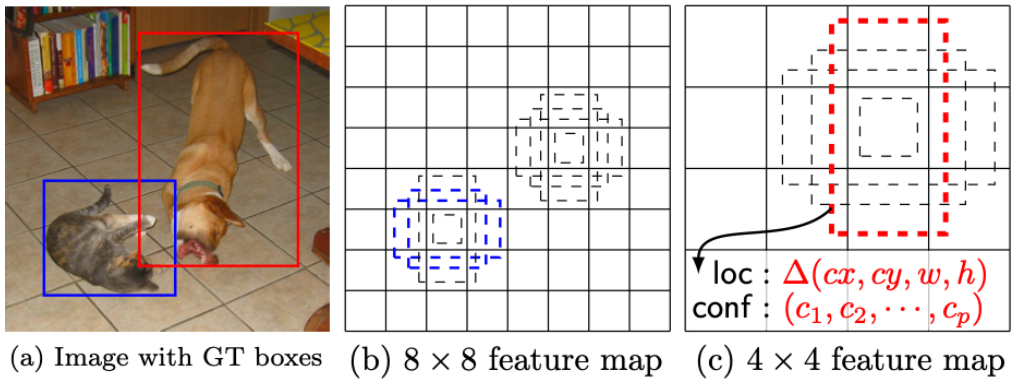


Figura 6 mapas de características SSD (11)

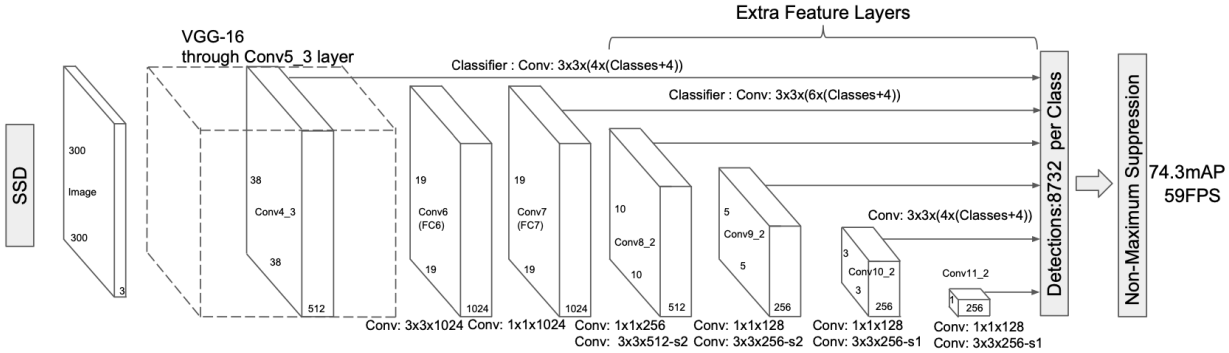


Figura 7 Arquitectura de SSD (11)

- **YOLO (You Only Look Once)** (12)**:** YOLO al igual que SSD es un algoritmo denominado "one stage", es decir, la imagen se procesa de una sola pasada a través de una red neuronal convolucional. Es probablemente el más popular y utilizado para detección en tiempo real, ya que sobre todo es sus versiones más recientes ofrece un gran rendimiento con bastante precisión. Aunque tiene algunos inconvenientes como la detección de objetos más pequeños.

  Consta de varias versiones, la más actual al momento de escribir este documento es la versión 8. A partir de la versión 3 el autor original dejó el mundo de la Inteligencia Artificial y se convirtió en un proyecto comunitario. Donde diferentes autores contribuyen en avances y lanzan nuevas versiones con mejoras. Las arquitecturas han ido cambiando por cada nueva iteración agregándole nuevas capas o sustituyendo bloques enteros por otros.

  En sus versiones iniciales YOLO divide la imagen de 448x448 pixeles en una cuadricula de SxS donde cada celda predice B cajas delimitadoras o "bounding boxes". Se pasa por el backbone donde una red convolucional pre-entrenada realiza las predicciones y por cada celda crea un vector que representa la probabilidad de que un objeto esté en la imagen junto a su bounding box.

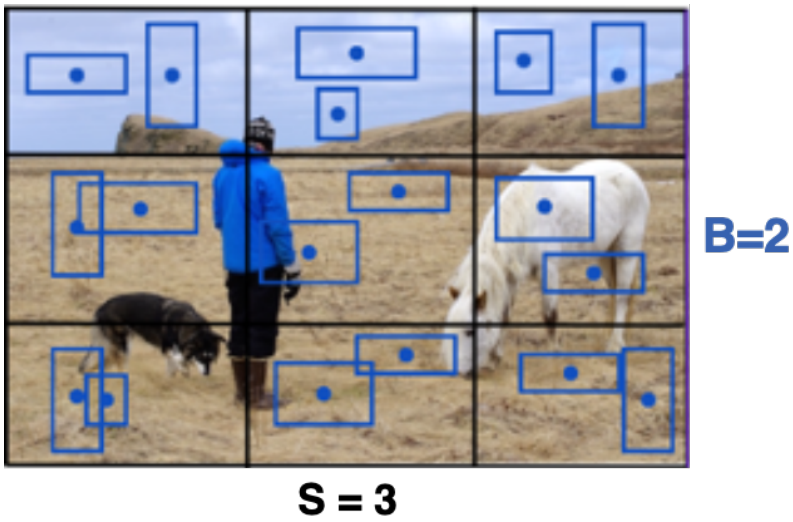


Figura 8 ejemplo de una imagen dividida en 3x3 celdas con 2 bounding boxes cada una

Pongamos un ejemplo donde la imagen se divide en S = 3, es decir 3x3 celdas. Con B=1, una sola bounding box por celda y es capaz de detectar dos clases distintas, c_1 y c_2. El vector resultante de cada celda tras pasar por la red neuronal tendrá la siguiente forma:

$$\mathbf{y = [p_c, b_x, b_y, b_h, b_w, c_1, c_2]}^{\mathbf{T}}$$

Donde p_c es la probabilidad de que en el la bounding box haya un objeto, b_x y b_y son las coordenadas del centro de la bounding box, b_h y b_w son el alto y ancho de la caja en porcentaje respecto al ancho y alto de una celda. Mientras que c_1 y c_2 son las probabilidades de que en la celda se encuentre un objeto de dicha clase.

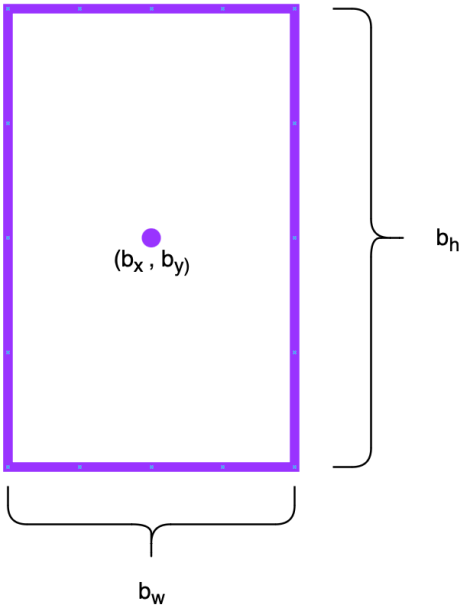


Figura 9 bounding box

En caso de existir más de una bounding box por celda, por ejemplo dos, el vector resultante seria tal que así:

$$\mathbf{y = [p_{c1}, b_{x1}, b_{y1}, b_{h1}, b_{w1}, p_{c2}, b_{x2}, b_{y2}, b_{h2}, b_{w2}, c_1, c_2]}^{\mathbf{T}}$$

Por lo que podemos deducir que el tamaño del vector será 5B+C donde C es el número de clases que es capaz de reconocer. Y dado que cada vector está presente en cada celda el resultado final tras el procesamiento en la red será un tensor multidimensional de SxSx5B+C.

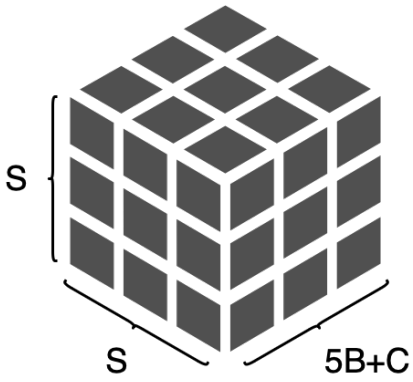


Figura 10 Tensor de salida YOLO

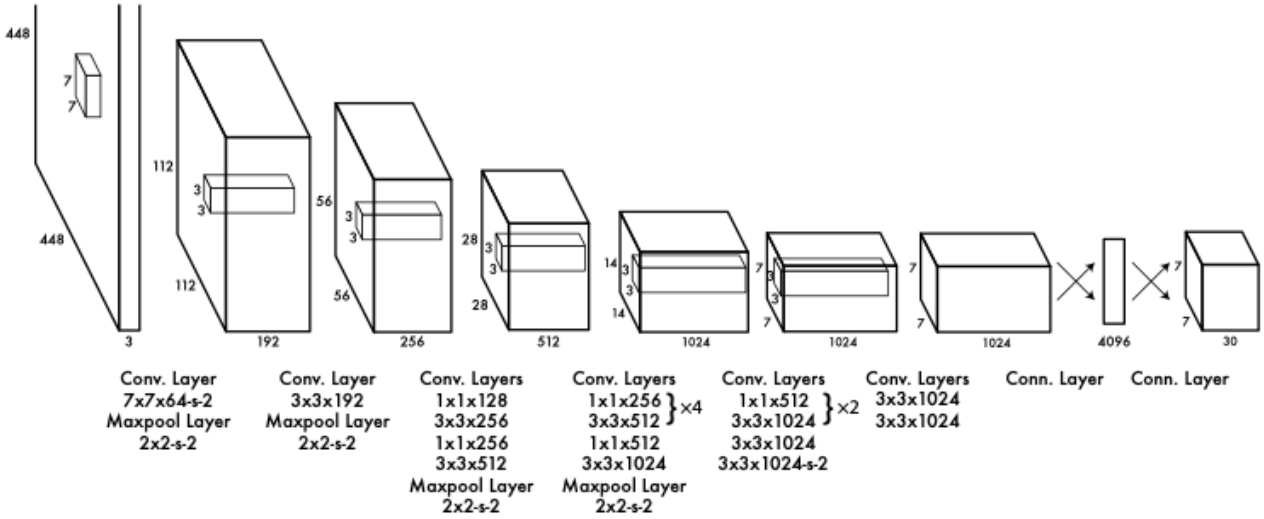


Figura 11 Arquitectura de YOLO (13)

Puede ocurrir que un mismo objeto sea detectado múltiples veces. Por lo que mediante el método de supresión de no máximos el cual eliminara la bounding box superpuesta con menor confianza de probabilidad haciendo uso de IOU (intersection over union).

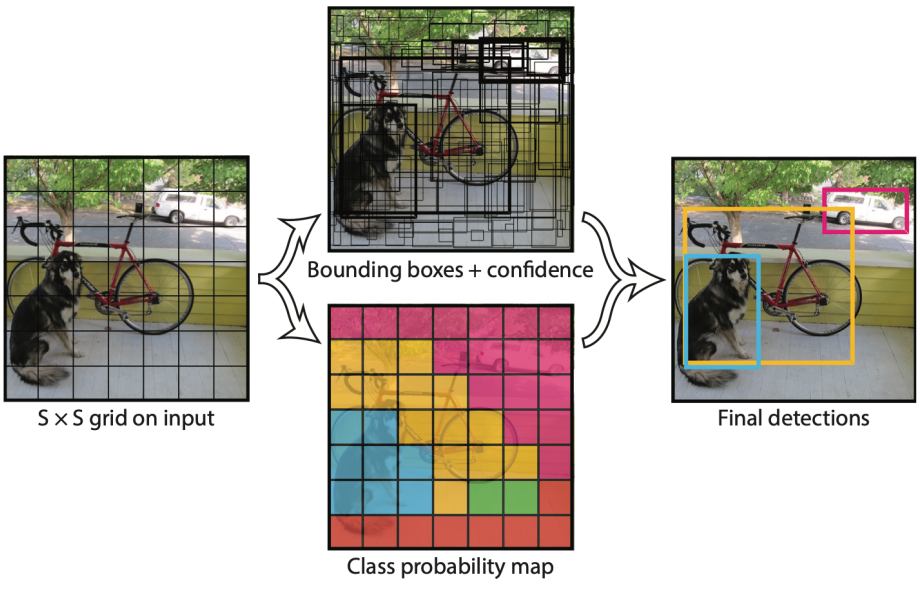


Figura 12 Procesamiento de imagen con YOLO (13)

Nosotros estaremos usando YOLOv7 el cual esta entrenado con el set de datos COCO (14) es decir que contaremos con hasta 80 clases distintas que podremos detectar. Entre ellas podemos encontrar personas, vehículos u objetos comunes. Por lo que es apto para nuestro trabajo.

La arquitectura que implementa YOLOv7 es una evolución de la versión 4. Presenta múltiples cambios y mejoras respecto a la versión inicial de YOLO. Entre lo más notorios que lo diferencian al resto de versiones es la introducción de una nueva arquitectura en el backbone llamada E-ELAN (Extended Efficient Layer Aggregation Network) lo que mejora el entrenamiento.

Otro punto clave es el escalado del modelo para modelos basados en concatenación (Model scaling for concatenation-based models). Esto permite ajustar los atributos del modelo para generar modelos a diferentes escalas para ajustarse a las necesidades de velocidad de inferencia en distintos tipos de aplicaciones.

Emplea la técnica denominada "Trainable bag-of-freebies". Son métodos que mejoran el rendimiento sin aumentar el costo de entrenamiento. Emplea los métodos "Planned re-parameterized convolution" y "Coarse for auxiliary and fine for lead loss".

En cuanto a rendimiento YOLOv7 podemos ver que sobrepasa a YOLOv4(predecesor) y YOLOR en parámetros necesarios para el entrenamiento y precisión según los datos obtenidos de su paper (12):

Tabla 3 Modelos de YOLOv7 comparados con modelos de YOLOv4 y YOLOR

Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLOv4 [3]	64.4M	142.8G	640	49.7%	68.2%	54.3%	32.9%	54.8%	63.7%
YOLOR-u5 (r6.1) [81]	46.5M	109.1G	640	50.2%	68.7%	54.6%	33.2%	55.5%	63.7%
YOLOv4-CSP [79]	52.9M	120.4G	640	50.3%	68.6%	54.9%	34.2%	55.6%	65.1%
YOLOR-CSP [81]	52.9M	120.4G	640	50.8%	69.5%	55.3%	33.7%	56.0%	65.4%
YOLOv7	36.9M	104.7G	640	**51.2%**	**69.7%**	**55.5%**	**35.2%**	**56.0%**	**66.7%**
improvement	-43%	-15%	-	+0.4	+0.2	+0.2	+1.5	=	+1.3
YOLOR-CSP-X [81]	96.9M	226.8G	640	52.7%	**71.3%**	57.4%	36.3%	57.5%	68.3%
YOLOv7-X	71.3M	189.9G	640	**52.9%**	71.1%	**57.5%**	**36.9%**	**57.7%**	**68.6%**
improvement	-36%	-19%	-	+0.2	-0.2	+0.1	+0.6	+0.2	+0.3
YOLOv4-tiny [79]	6.1	6.9	416	24.9%	42.1%	25.7%	8.7%	28.4%	39.2%
YOLOv7-tiny	6.2	5.8	416	**35.2%**	**52.8%**	**37.3%**	**15.7%**	**38.0%**	**53.4%**
improvement	+2%	-19%	-	+10.3	+10.7	+11.6	+7.0	+9.6	+14.2
YOLOv4-tiny-3l [79]	8.7	5.2	320	30.8%	47.3%	32.2%	**10.9%**	31.9%	51.5%
YOLOv7-tiny	6.2	3.5	320	**30.8%**	**47.3%**	**32.2%**	10.0%	**31.9%**	**52.2%**
improvement	-39%	-49%	-	=	=	=	-0.9	=	+0.7
YOLOR-E6 [81]	115.8M	683.2G	1280	55.7%	73.2%	60.7%	40.1%	**60.4%**	69.2%
YOLOv7-E6	97.2M	515.2G	1280	**55.9%**	**73.5%**	**61.1%**	**40.6%**	60.3%	**70.0%**
improvement	-19%	-33%	-	+0.2	+0.3	+0.4	+0.5	-0.1	+0.8
YOLOR-D6 [81]	151.7M	935.6G	1280	56.1%	73.9%	61.2%	**42.4%**	60.5%	69.9%
YOLOv7-D6	154.7M	806.8G	1280	56.3%	73.8%	61.4%	41.3%	60.6%	70.1%
YOLOv7-E6E	151.7M	843.2G	1280	**56.8%**	**74.4%**	**62.1%**	40.8%	**62.1%**	**70.6%**
improvement	=	-11%	-	+0.7	+0.5	+0.9	-1.6	+1.6	+0.7

Si lo comparamos con YOLOv5, YOLOX y PPYOLOE vemos que YOLOv7-X mejora a YOLOv5-L en + 15fps y + 3.9% de precisión. Frente a PPYOLE-L, YOLOv7 es 106% más rápido realizando inferencias.

Tabla 4 Modelos de YOLOv7 comparados con YOLOX, PPYOLE, YOLOR y YOLOv5 (12)

Model	#Param.	FLOPs	Size	FPS	AP^{test} / AP^{val}	AP^{test}_{50}	AP^{test}_{75}	AP^{test}_{S}	AP^{test}_{M}	AP^{test}_{L}
YOLOX-S [21]	9.0M	26.8G	640	102	40.5% / 40.5%	-	-	-	-	-
YOLOX-M [21]	25.3M	73.8G	640	81	47.2% / 46.9%	-	-	-	-	-
YOLOX-L [21]	54.2M	155.6G	640	69	50.1% / 49.7%	-	-	-	-	-
YOLOX-X [21]	99.1M	281.9G	640	58	51.5% / 51.1%	-	-	-	-	-
PPYOLOE-S [85]	7.9M	17.4G	640	208	43.1% / 42.7%	60.5%	46.6%	23.2%	46.4%	56.9%
PPYOLOE-M [85]	23.4M	49.9G	640	123	48.9% / 48.6%	66.5%	53.0%	28.6%	52.9%	63.8%
PPYOLOE-L [85]	52.2M	110.1G	640	78	51.4% / 50.9%	68.9%	55.6%	31.4%	55.3%	66.1%
PPYOLOE-X [85]	98.4M	206.6G	640	45	52.2% / 51.9%	69.9%	56.5%	33.3%	56.3%	66.4%
YOLOv5-N (r6.1) [23]	1.9M	4.5G	640	159	- / 28.0%	-	-	-	-	-
YOLOv5-S (r6.1) [23]	7.2M	16.5G	640	156	- / 37.4%	-	-	-	-	-
YOLOv5-M (r6.1) [23]	21.2M	49.0G	640	122	- / 45.4%	-	-	-	-	-
YOLOv5-L (r6.1) [23]	46.5M	109.1G	640	99	- / 49.0%	-	-	-	-	-
YOLOv5-X (r6.1) [23]	86.7M	205.7G	640	83	- / 50.7%	-	-	-	-	-
YOLOR-CSP [81]	52.9M	120.4G	640	106	51.1% / 50.8%	69.6%	55.7%	31.7%	55.3%	64.7%
YOLOR-CSP-X [81]	96.9M	226.8G	640	87	53.0% / 52.7%	71.4%	57.9%	33.7%	57.1%	66.8%
YOLOv7-tiny-SiLU	6.2M	13.8G	640	286	38.7% / 38.7%	56.7%	41.7%	18.8%	42.4%	51.9%
YOLOv7	36.9M	104.7G	640	161	51.4% / 51.2%	69.7%	55.9%	31.8%	55.5%	65.0%
YOLOv7-X	71.3M	189.9G	640	114	53.1% / 52.9%	71.2%	57.8%	33.8%	57.1%	67.4%
YOLOv5-N6 (r6.1) [23]	3.2M	18.4G	1280	123	- / 36.0%	-	-	-	-	-
YOLOv5-S6 (r6.1) [23]	12.6M	67.2G	1280	122	- / 44.8%	-	-	-	-	-
YOLOv5-M6 (r6.1) [23]	35.7M	200.0G	1280	90	- / 51.3%	-	-	-	-	-
YOLOv5-L6 (r6.1) [23]	76.8M	445.6G	1280	63	- / 53.7%	-	-	-	-	-
YOLOv5-X6 (r6.1) [23]	140.7M	839.2G	1280	38	- / 55.0%	-	-	-	-	-
YOLOR-P6 [81]	37.2M	325.6G	1280	76	53.9% / 53.5%	71.4%	58.9%	36.1%	57.7%	65.6%
YOLOR-W6 [81]	79.8G	453.2G	1280	66	55.2% / 54.8%	72.7%	60.5%	37.7%	59.1%	67.1%
YOLOR-E6 [81]	115.8M	683.2G	1280	45	55.8% / 55.7%	73.4%	61.1%	38.4%	59.7%	67.7%
YOLOR-D6 [81]	151.7M	935.6G	1280	34	56.5% / 56.1%	74.1%	61.9%	38.9%	60.4%	68.7%
YOLOv7-W6	70.4M	360.0G	1280	84	54.9% / 54.6%	72.6%	60.1%	37.3%	58.7%	67.1%
YOLOv7-E6	97.2M	515.2G	1280	56	56.0% / 55.9%	73.5%	61.2%	38.0%	59.9%	68.4%
YOLOv7-D6	154.7M	806.8G	1280	44	56.6% / 56.3%	74.0%	61.8%	38.8%	60.1%	69.5%
YOLOv7-E6E	151.7M	843.2G	1280	36	56.8% / 56.8%	74.4%	62.1%	39.3%	60.5%	69.0%

[1] Our FLOPs is calaculated by rectangle input resolution like 640 × 640 or 1280 × 1280.
[2] Our inference time is estimated by using letterbox resize input image to make its long side equals to 640 or 1280.

Con estos datos podemos concluir que YOLOv7 es un candidato apto para nuestra aplicación ya que ofrece un gran rendimiento con una tasa de precisión alta. En capítulos posteriores de este trabajo se realizará un estudio del rendimiento real y se comparará los diferentes modelos que ofrece pre-entrenados bajo nuestro hardware de pruebas para ver cual se adapta mejor a nuestra aplicación.

2.2 Servicios de computación en la nube

En la actualidad existen múltiples posibilidades a la hora elegir de servicios de computación en la nube. Los tres proveedores más populares e importantes son Amazon Web Services (15), Microsoft Azure (16) y Google Cloud (17). Cada uno de ellos ofrecen múltiples servicios como instancias virtuales, soluciones "serverless"[4], almacenamiento, APIs o redes virtuales.

Para este trabajo se hará uso de los servicios Web API, serverless y envío de correo y SMS. Por lo que estas son las alternativas que ofrece cada proveedor:

Tabla 5 Comparativa servicios AWS, Azure y Google Cloud

Servicio	AWS	Azure	Google Cloud
Web API	API Gateway	API Management	API Gateway
Envío de Correo	Simple Email Service (SES)	Communication Services	*
Envío de SMS	Simple Notification Service (SNS)	Communication Services	*
Serverless	Lambda Function	Azure Functions	Cloud Functions

*No disponible de forma nativa.

[4] Los servicios serverless permiten al desarrollador enfocarse al despliegue de software sin tener que lidiar con la configuración y mantenimiento del entorno de la aplicación ni servidor.

Como se puede observar cada proveedor ofrece un servicio equivalente a excepción de Google que no cuenta con servicios de envío de correo y SMS nativos. Aunque si ofrece herramientas y facilidades para usar servicios de terceros como SendGrid [15].

Otro factor a tener en cuenta es la facturación de cada servicio. Como vimos en 1.3 AWS cuenta con una capa gratuita bastante generosa, los costes de utilizar los servicios son:

- Lambda Function: 1 000 000 solicitudes gratuitas por mes y hasta 3,2 millones de segundos de tiempo de informática por mes.
- SES: 62 000 mensajes salientes por mes y 1000 mensajes entrantes por mes gratuitos.
- SNS: 1 000 000 de publicaciones gratuitas y 0,06087$ por SMS entregado.
- API Gateway: 1 millón de peticiones gratuitas al mes.

En el caso de Azure tenemos los siguientes costes:

- Communication services: Para el envío de SMS Azure se necesita arrendar un número de teléfono con un coste de 2$ al mes. Luego se aplica dos precios por uso, 0,0075$ por segmento de SMS enviado (un segmento ~ 150 caracteres) y suplemento del operador de 0,0025$. Estos precios son para EE. UU. por lo que se puede aplicar costes adicionales en España. Por cada correo electrónico se aplican los siguientes cargos; $0,00025/Correo electrónico enviado y $0,00012/MB transferidos.
- API Management: 1 millón de operaciones gratuitas.
- Azure Functions: 1 millones de solicitudes y 400.000 GB-segundos de consumo de recursos gratuitos mensuales.

En el caso de Google al no contar con servicios de mensajería ha sido descartada. Como parte de los requisitos establecidos es mantener los costes más bajos posibles se ha elegido a AWS como el proveedor de servicios para la aplicación ya que solo hay que pagar por SMS.

2.3. Hardware utilizado

Es necesario cierto equipamiento para ejecutar la aplicación. Primero se necesita una o varias fuentes de vídeo que sean compatibles con RTSP. Para esto puede hacerse uso de cámaras de videovigilancia domésticas. En este escenario se ha empleado el modelo Tapo C200 de TP-Link. Este equipo puede adquiriese por unos 30€. Entre sus características cuenta con almacenamiento interno en micro-sd, se controla y configura mediante su app para smartphone, es accesible desde cualquier lugar a través de la app entre otras.

Figura 13 Cámara Tapo C200

Por otro lado se ha utilizado un ordenador para realizar las funciones de servidor con las siguientes características:

- CPU: AMD Ryzen 2600, 6c/12t 3.4Ghz.
- GPU: Nvidia RTX 4070 12GB gddr6x. y Nvidia GTX 1070ti 8GB
- RAM: 16GB 3200Mhz
- SSD: 1TB SSD sata.
- SO: Ubuntu 22.04.2 LTS

También se ha empleado un MacBook Air con chip Apple M1 para el desarrollo.

El principal requisito que se ha de cumplir es contar con tarjeta de vídeo dedicada de la marca Nvidia ya que estas cuentan con núcleos Cuda. Esto es importante debido a que las librerías implementadas hacen uso de estos núcleos para acelerar el procesamiento de las imágenes. En caso de no contar con alguna tarjeta se puede realizar el procesamiento con la CPU del ordenador pero resultará ser mucho más lento.

En siguientes capítulos se estudiará el rendimiento de diferentes modelos de YOLOv7 corriendo en diferentes tarjetas y en CPU para comparar los resultados.

En este capítulo hemos introducido el hardware y las tecnologías empleadas para este trabajo junto a sus competidoras en el mercado. En el caso de Yolov7 la hemos seleccionado porque muestra una gran diferencia tanto con el resto de las versiones de Yolo como con el resto de modelos existentes (SSD y R-CNN) en la velocidad de inferencia manteniendo buena precisión. En el campo de servicios en la nube nos hemos quedado con AWS ya que cuenta con todos servicios que necesitamos para nuestro proyecto y los costes de uso son mínimos respecto al resto.

3 APLICACIÓN

En este capítulo se expondrá los diferentes componentes de la aplicación, sus dependencias, configuración, puesta en marcha y funcionamiento de la misma. Se mostrarán partes de archivos de configuración y código como apoyo en la representación. Se omitirán archivos de entorno que contienen claves de acceso, usuarios o información considerada sensible que supongan un riesgo, ya que se podrían utilizar para obtener acceso a los servicios de AWS, los cuales tienen coste de uso.

Como se ha comentado anteriormente, el sistema de videovigilancia consta de dos partes, la detección y la notificación. Se recomienda consultar la **Figura 4** en el apartado **1.3 Escenario propuesto** para una visión del esquema de la arquitectura.

La parte de detección se realiza en un entorno local, un servidor obtendrá las imágenes de las cámaras que están en la red y las analizará con Yolov7 para detectar alguna persona. Una vez realizada una detección se almacenará una grabación de todo lo que este ocurriendo mientras la detección dure y se realizará una petición de notificación al usuario utilizando los servicios de AWS mediante una llamada a una API HTTP.

La parte de notificación está implementada en AWS y pone a disposición de las aplicaciones de detección una API HTTP como punto de conexión y activación. Una vez llega una petición de notificación la cual contiene un número de teléfono, correo electrónico y captura de la imagen de la detección. Se activará una función lambda la cual coordinará y hará uso de los servicios SNS para enviar un mensaje SMS y SES para enviar un correo electrónico al usuario.

Dividiremos la explicación en dos partes, el entorno local donde se realiza la detección y el entorno de AWS donde se realiza la notificación. Ya que ambos elementos son independientes entre sí.

3.1 Entorno local

El entorno local consta del servidor y las cámaras como se puede ver en Figura 4, ambos componentes deben de encontrarse dentro de la misma red y el servidor debe contar con conexión a internet para alcanzar los servicios de AWS. Dentro del servidor se encuentra la aplicación la cual es la encargada de obtener los stream de vídeo de las cámaras, analizar las imágenes, almacenar localmente los clips de video de las grabaciones con detecciones y activar el sistema de notificación cuando se realice una detección.

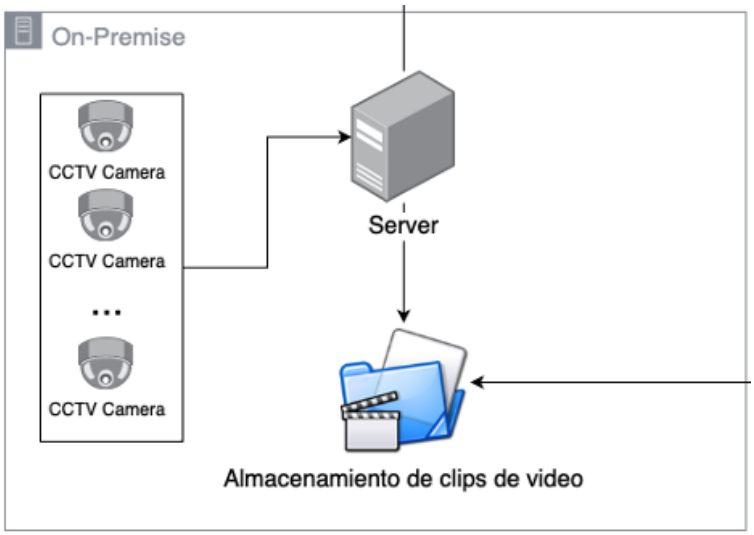


Figura 14 Fragmento del entorno local de la Figura 4

En los próximos apartados se explicarán la configuración y funcionamiento de los elementos que acabamos de ver y de la aplicación. Dentro de la aplicación se explicará el sistema de archivos el cual es importante conocer para su configuración e interacción con el usuario. Por último se explicará el proceso para instalar la aplicación y ponerla en marcha.

3.1.1 Cámaras CCTV

Para este trabajo se ha empleado la cámara Tapo C200 de TPLink por los motivos comentados en el apartado **2.3 Hardware utilizado**. La configuración de este elemento se realiza a través de la aplicación móvil Tapo, solo hay que activar RTSP y anotar la URL del stream. Si se emplea un dispositivo de otra marca la configuración podría ser distinta. El único requisito es tener activado RTSP en el dispositivo y conocer la URL. Por norma general suele tener el siguiente formato: `rtsp://usuario:contraseña@host:puerto/stream`. En nuestro caso se ha añadido una tarjeta de memoria microSD para que esté continuamente grabando en ella, activado algunas funcionalidades extras de este modelo y creado una reserva DHCP en el router para tener una IP estática de la cámara.

En caso de no contar con una cámara compatible con RTSP se puede emplear el programa VLC el cual permite realizar una transmisión RTSP de un archivo de video que tenga almacenado el equipo.

3.1.2 Servidor local

Como servidor se ha utilizado un ordenador con el hardware descrito en **2.3 Hardware utilizado**. Como sistema operativo se ha utilizado Ubuntu 22.04.2 LTS. La aplicación es compatible con el resto de los sistemas Linux y también se ha probado en sistemas Unix como MacOS Ventura 13.4. Para sacar partido de la aceleración por hardware de vídeo es necesario contar con una tarjeta Nvidia e instalar los drivers de la tarjeta y CUDA. Dependiendo de la versión, sistema operativo y método elegido los pasos son diferentes. En la documentación de Nvidia (19) puede consultarse los pasos a seguir. En este caso se descargó e instaló el archivo runfile `cuda_12.1.1_530.30.02_linux.run`.

Se puede comprobar las versiones de CUDA y drivers instalados mediante el comando `nvidia-smi`.

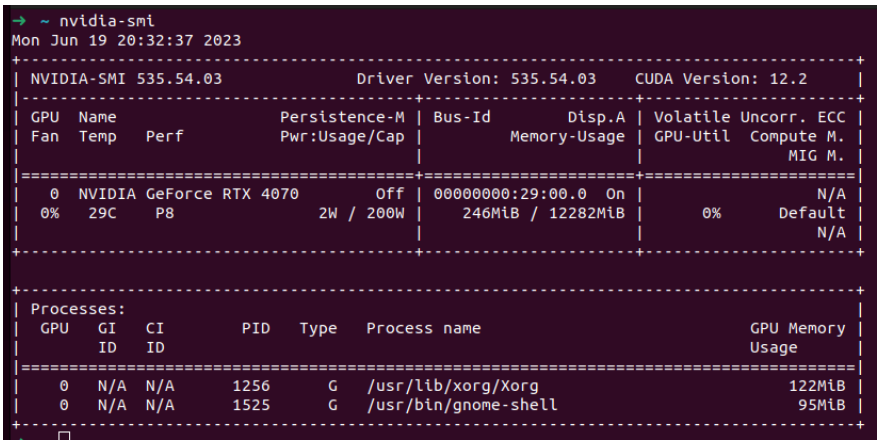


Figura 15 Drivers de Nvidia y CUDA instalados

Para poder interactuar directamente con los servicios de AWS es necesario tener instalado la herramienta AWS CLI y configurar el archivo `~/.aws/credentials` con una clave de acceso de tu perfil de AWS.

3.1.3 Aplicación

La aplicación se ejecuta mediante terminal y está desarrollada en Python 3.9. Se ha utilizado como punto de partida el proyecto original de Yolov7 (20) y sobre este se han modificado archivos, añadido nuevos y eliminado los no necesarios. Se ha adaptado el comportamiento del programa original adaptándolo a las necesidades del nuestro y añadido funcionalidades.

El funcionamiento y la interacción con la aplicación se ha querido mantener simple para facilitar la replicación y usabilidad. Para ello se hace uso de diferentes ficheros de configuración que pueden ser modificados con un simple editor de texto y todos los parámetros han sido detallados y dado ejemplos de las posibles opciones a utilizar.

```
[Configuration]
;Object to detect
object = person
;Model used for predictions. yolov7.pt, yolov7-tiny.pt, yolov7-e6e.pt
weights = yolov7.pt
;Inference size (pixels) Default 640
img_size = 640
;.txt file with video streams ulrs
source = streams.txt
;Minimun confidence threshold for detections
conf_thres = 0.65
;IOU threshold for NMS
iou_thres = 0.45
;Decive used for predictions if CPU use cpu if GPU use 0 or gpu number
device = cpu
;True if you want to display the stream on the screen
view_img = True
;Save images/videos
save_vid= True
;don`t trace model
no_trace = False

[Notification]
;phone number to send sms notifiations. Use E.164 format. Example: +34XXXXXXXXX
phone = +34123456789
;Mail address to send mail notifications.
mail = fraberare@alum.us.es
;Enable notifications
notifiations = True
;Time between alerts sent to user in minutes
countdown = 10
```

Figura 16 ejemplo de fichero de configuración config.ini

Una vez ejecutada la aplicación leerá los ficheros de configuración, obtendrá los stream de vídeo indicados y cargará el modelo de detección indicado junto al método de procesamiento (cpu o gpu). Tras esto empezará el bucle principal donde estará constantemente analizando fotograma a fotograma en busca de detecciones.

Cuando haya una coincidencia y se detecte uno de los objetos que se le ha indicado (cuenta con más de 80 objetos distintos ya que estamos haciendo uso de los modelos previamente entrenados con el dataset de COCO) se guardará la captura del fotograma y en caso de tener activas las notificaciones se llamará a la API web donde se pasará la captura, número de teléfono y correo electrónico para notificar al usuario. Una vez realizada la notificación habrá un periodo de tiempo de espera especificado en el fichero de configuración hasta enviar la siguiente notificación para evitar constantes notificaciones.

Del mismo modo si la grabación de vídeo está activa se grabará un clip de vídeo de lo que esté capturando la cámara y se guardará con la fecha y la cámara que realizó la grabación.

Mientras esté en ejecución se visualizará por pantalla todos los stream de vídeo que estén activos y finalizará cuando se presione la tecla “q” o se acabe los stream de vídeo. A lo largo de la ejecución se guardarán logs los cuales contienen información de los objetos detectados y eventos que han ido ocurriendo.

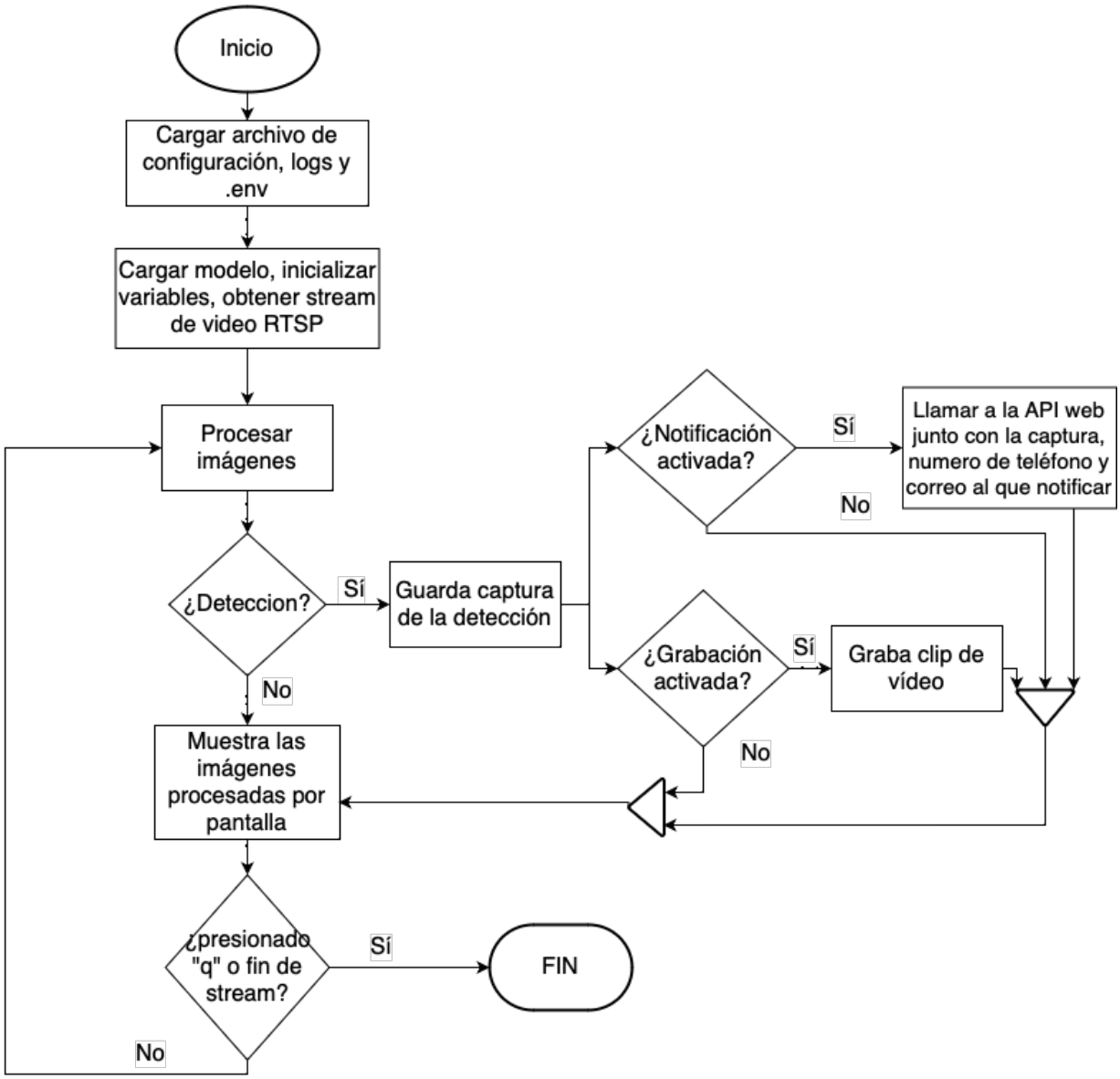


Figura 17 Diagrama de flujo Main.py

3.1.3.1 Sistema de archivos

Es importante conocer el sistema de archivos ya que para poner en marcha, configurar parámetros y revisar clips de video se necesitan hacer uso de diferentes archivos. Los directorios `models` y `utils` son dependencias de Yolo y contienen funciones que utiliza el modelo internamente para realizar inferencias. También podemos encontrar los directorios `tools` y `AWS` los cuales no son parte de la propia aplicación pero contienen código auxiliar para interactuar directamente con los servicios de AWS o el código fuente de la función lambda que está implementada. Para más información sobre estos ficheros se puede consultar el apartado **3.2**.

El sistema de archivos final es el siguiente:

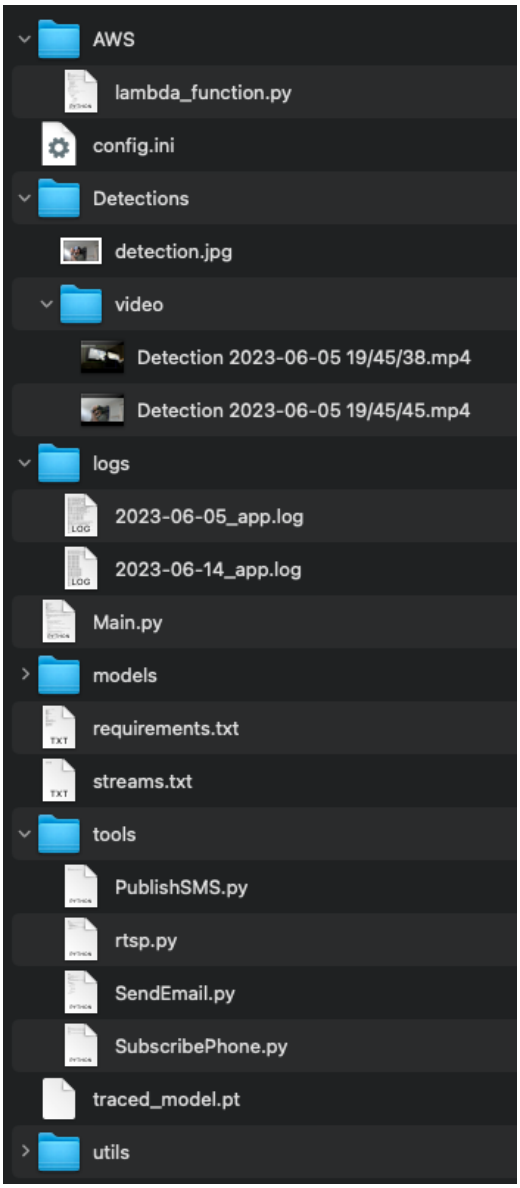


Figura 18 Sistema de archivos

- AWS: Directorio que contiene el código implementado en la función lambda de AWS.
- Config.ini: Archivo de configuración donde modificar los parámetros de ejecución.
- Detections: Directorio que contiene una captura de la última detección realizada y un sub-directorio con los clips de vídeo de las detecciones con sello de tiempo y cámara.
- Logs: directorio donde se guardan los logs de la ejecución.
- Main.py: Archivo principal de ejecución y modificación del archivo original detect.py de Yolov7.
- Models: Directorio que contiene los modelos de Yolov7 y funciones necesarias para su funcionamiento.
- Requirements.txt: Archivo que contiene una lista de las librerías de Python necesarias para el funcionamiento de la aplicación y que deben de ser instaladas.
- Streams.txt: Fichero donde se encuentran las URLs RTSP de las cámaras a las que conectarse.
- Tools: Directorio que contiene funciones de apoyo que interactúan con AWS para agilizar el registro de números de teléfono o interacción directa con los mismos.
- Utils: Directorio de apoyo de Yolov7 para su funcionamiento.

3.1.3.2 Puesta en marcha

Para instalar y configurar la aplicación en un equipo nuevo se han de seguir los siguientes pasos:

- Obtener el proyecto: El proyecto se encuentra en GitHub a través de la siguiente url https://github.com/franberdejo/tfg para obtenerlo se puede clonar el repositorio mediante. `git clone https://github.com/franberdejo/tfg`

- Instalar dependencias: Para instalar las dependencias necesarias se puede emplear PIP el sistema de gestión de paquetes de Python. Todas las dependencias están listadas en requirements.txt. Ejecuta `pip install -r requirements.txt`

- Crear archivo .env con la API key obtenida al crear la API web para autenticar las peticiones a la API. Y URL de del método post http. Dicha llave es única y secreta por eso no se ha añadido ninguna en el proyecto y está en un archivo .env.

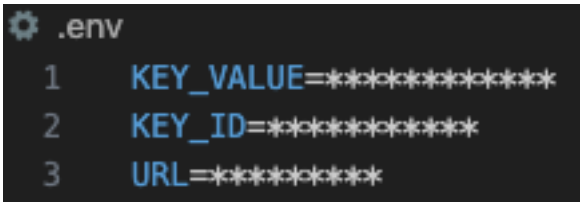


Figura 19 fichero .env

- Añadir las URLs de RTSP al archivo streams.txt (una URL por fila).

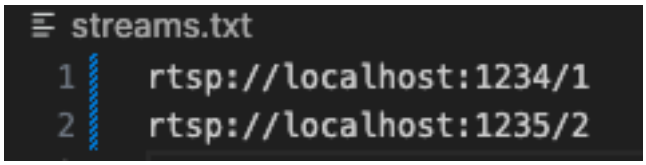


Figura 20 fichero streams.txt

- Modificar el archivo config.ini. Los principales campos a tener en cuenta son:
  - Device: dispositivo que realizará el procesamiento de imágenes. “0” en caso de contar con tarjeta gráfica o “cpu” si usamos el procesador.
  - Phone: número de teléfono donde enviar los SMS.
  - Mail: dirección de correo electrónico donde enviar el correo junto la imagen de detección.
  - Weights: modelo utilizado para la detección de objetos.
  - Object: objeto que queremos detectar. (person, dog, cell phone…)
- Ejecutar el archivo Main.py. `pyhon3 main.py`

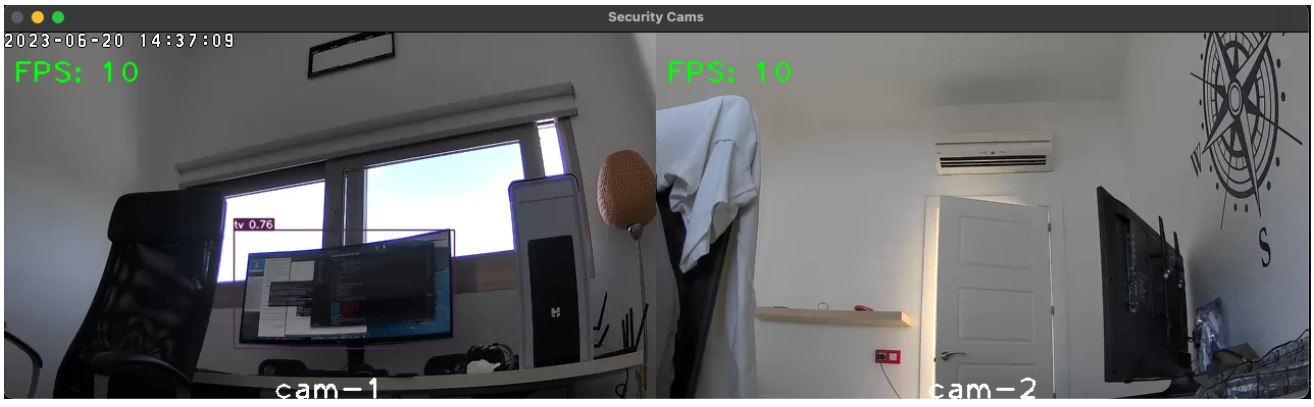


Figura 21 ejemplo de ejecución con dos cámaras

3.2 Amazon Web Services

Una vez visto el entorno local y cómo funcionan sus componentes pasaremos a ver el segundo bloque de nuestro proyecto encargado de realizar la notificación.

Como se comentó en apartados anteriores se ha utilizado AWS como proveedor de servicios ya que ofrece integración total con SMS, Correo, API y Serverless prácticamente a coste 0 en su capa gratuita teniendo que pagar solo por el precio de los SMS. Los servicios utilizados son API Gateway para hacer las funciones de API HTTP, Lambda Functions para realizar el procesamiento de la información y gestionar el resto de los servicios con tecnología serverless. Simple Notification Service para el envío de mensajes SMS y Simple Email Service para el envío de correos electrónicos.

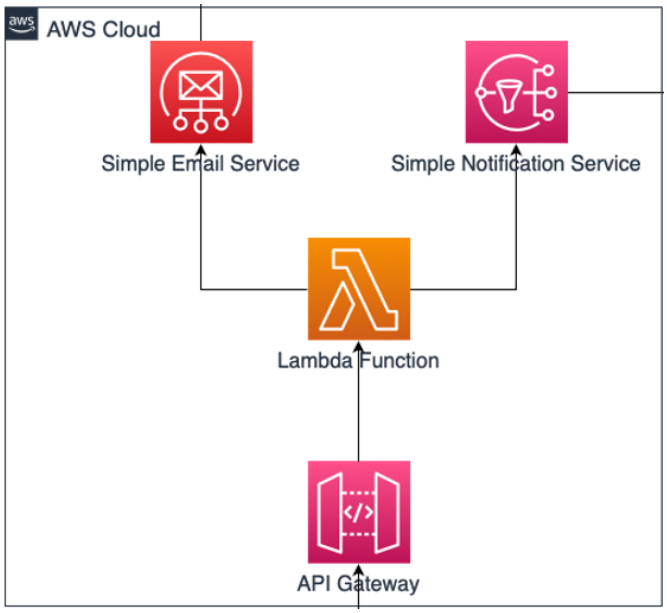


Figura 22 Fragmento AWS Cloud de la Figura 4

3.2.1 API Gateway

Este servicio ofrece una API REST como puerta de comunicación entre la aplicación y el resto del sistema de AWS. Para ello la API ofrece un método POST llamado /LambdaDetection. Dicho método está integrado directamente con la función Lambda y le proporciona los datos necesarios para la ejecución que recibe en el cuerpo del mensaje.

Figura 23 API Gateway

A parte de punto de entrada al sistema, también realiza las funciones de autenticación mediante la configuración de una llave que se ha de adjuntar en la llamada para que sea aceptada. Junto a un plan de uso de un máximo de 2 peticiones por segundo.

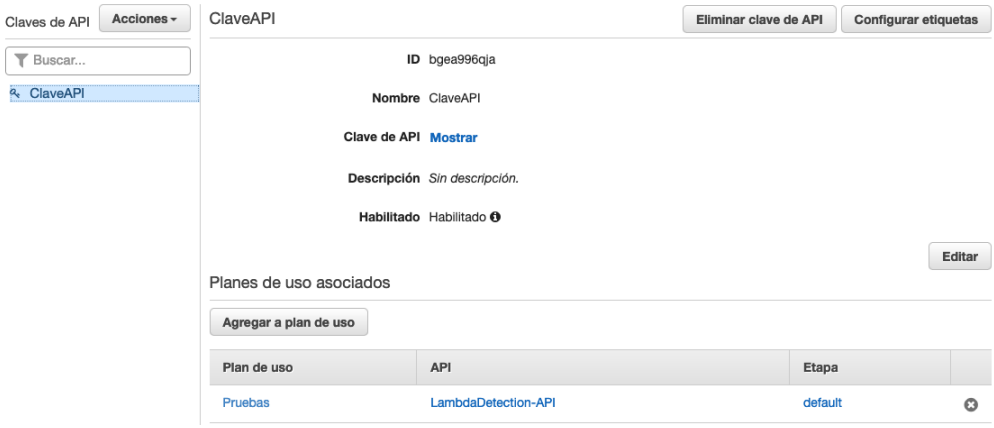


Figura 24 Clave API

Dicha llave es la que se ha de guardar en el archivo .env junto a la URL del método y adjuntarlas en la cabecera de la petición HTTP.

```
#initalice aws http api gateway variables
load_dotenv()
KEY_VALUE = os.getenv('KEY_VALUE')
KEY_ID = os.getenv('KEY_ID')
URL = os.getenv('URL')
headers = {'x-api-key': KEY_VALUE}
```

Figura 25 inicialización de variable API

3.2.2 Lambda Function

El servicio Lambda Function es un servicio serverless, es decir, no hay que gestionar ningún entorno o equipo. Simplemente desplegamos el código que queremos que sea ejecutado cuando se active. En nuestro caso tenemos una función en Python3.9. Es el punto central del entorno AWS ya que es el encargado de procesar la imagen y notificar por correo y SMS a los destinos que le han sido pasados. Recibe en la llamada a ejecución tres parámetros; el número telefónico, dirección de correo electrónico y la imagen de la detección obtenida de la aplicación.

Para la interacción con el resto de los servicios se ha empleado la librería boto3 la cual permite realizar peticiones a los servicios mediante el SDK de Python. Una vez finalizada la función se devolverá un mensaje de error en caso de algún fallo o un mensaje notificando que todo ha salido bien.

Para la comunicación con el resto de servicios han de configurarse los permisos pertinentes. Para ello se ha creado un rol de ejecución y una política basa en recursos.

Dentro del rol creado se han insertado dos políticas PublicarSNS para interactuar con SNS y SES_Detection para interactuar con SES.

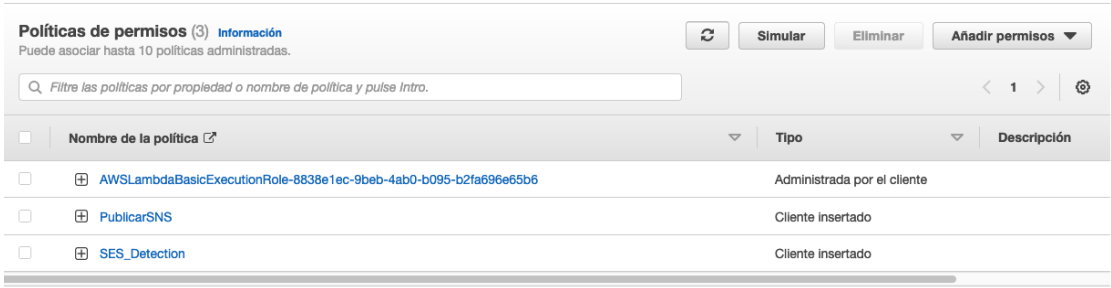


Figura 26 Politicas empleadas en el rol de Lambda

Para enviar mensajes SMS mediante el servicio SNS es necesario incluir los permisos de Publish en SNS y de SendMessages de Pinpoint en la política PublicarSNS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sns:Publish",
                "mobiletargeting:SendMessages"
            ],
            "Resource": "*"
        }
    ]
}
```

Del mismo modo para poder enviar correos electrónicos en la política SES_Detection se ha de añadir permiso para usar la acción SendEmail.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SESDetection",
            "Effect": "Allow",
            "Action": "ses:SendEmail",
            "Resource": "*"
        }
    ]
}
```

Por último en la política basada en recursos añadimos los permisos para que API Gateway pueda ejecutar nuestra función lambda.

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "******",
      "Effect": "Allow",
      "Principal": {
        "Service": "apigateway.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource":                                  "arn:aws:lambda:eu-west-
1:*****:function:LamdaDetection",
      "Condition": {
        "ArnLike": {
          "AWS:SourceArn":                "arn:aws:execute-api:eu-west-
1:****:***/*/POST/LamdaDetection"
        }
      }
    }
  ]
}
```

3.2.3 Simple Notification Service

Este servicio está centrado en enviar notificaciones entorno a topics o temas. Los temas son grupos de notificación donde los clientes que reciben notificaciones pueden suscribirse a los que estén interesados en recibir. Por otro lado están los publicadores donde pueden publicar un mensaje en un tema y todos los que estén suscritos a este recibirán el mensaje sin necesidad de tener que preocuparte del envío individual. Dichos mensajes pueden ser de diverso tipo; SMS, correo, notificaciones de inserción a aplicaciones o enlazarlos con otros servicios de AWS.

También permite el envío directo de mensajes SMS sin necesidad de suscribir los números a un tema. Aunque para esto se apoya en el servicio de Pinpoint, pero para nosotros no requiere de ninguna acción extra aparte de asegurarnos de tener los permisos requeridos que vimos en el apartado anterior. Se podría haber implementado las notificaciones por correo bajo este servicio pero no permite una personalización ni incrustar archivos dentro del correo, tiene unas finalidades para casos de uso más simples. Por lo que se utilizará el servicio Simple Email Service.

Para utilizar el servicio lo haremos desde la propia función lambda mediante la librería de boto3. Para ello utilizaremos el método de publicar y adjuntaremos el número de teléfono donde enviarlo que recibimos de la aplicación y un mensaje prestablecido avisando de la detección.

```
try:
    sns = boto3.client('sns')
    SNSResponse = sns.publish(
    PhoneNumber=event['phone'],
    Message= MESSAGE_SNS,
    )
except ClientError:
    response = "SMS couldn't be sent."
    return response
```

Figura 27 activación del servicio SNS

Como apunte a tener en cuenta para este proyecto se ha utilizado el modo sandbox de AWS, el cual nos permite utilizar el servicio bajo un entorno de pruebas donde limitan el envío de mensajes a números de teléfono previamente verificados en el sistema. Esto nos permite poder enviar mensajes sin que afecte a nuestra reputación como emisores. Como este proyecto tiene fines académicos no se ha solicitado el traspaso de la cuenta fuera del entorno ya que no afecta a la funcionalidad de este.

Para poder registrar números nuevos de manera rápida y sencilla y realizar pruebas de envío de SMS, se han creado los scripts `SubscribePhone.py` y `PublishSMS.py` respectivamente.

3.2.4 Simple Email Service

El motivo de utilizar este servicio se debe a que nos permite crear y enviar correos electrónicos con formato HTML y así poder incrustar una imagen de la detección realizada en Base64 y que al usuario le llegue la notificación con una imagen ya que la mayoría de los clientes de correos modernos soportan este formato.

Para ello y al igual que con SNS este servició se activará mediante la función lambda y la librería boto3. Como se comentó el apartado anterior donde se exponía la configuración de lambda hay que asegurarse que se tienen los permisos correctos para poder hacer el envío de mensajes.

De forma análoga al anterior se enviará un mensaje de correo electrónico a la dirección proporcionada en la llamada a la API junto a la captura en Base64 que será interpretada posteriormente por el cliente de correo del usuario.

```
ses = boto3.client('ses',region_name=AWS_REGION)
try:
    SESResponse = ses.send_email(
        Destination={
            'ToAddresses': [
                event['mail'],
            ],
        },
        Message={
            'Body': {
                'Html': {
                    'Charset': CHARSET,
                    'Data': BODY_HTML,
                },
                'Text': {
                    'Charset': CHARSET,
                    'Data': BODY_TEXT,
                },
            },
            'Subject': {
                'Charset': CHARSET,
                'Data': SUBJECT,
            },
        },
        Source=SENDER,
    )
except ClientError as e:
    response = "Email couldn't be sent"
    return response
```

Figura 28 Activación del servicio SES desde la función lambda

Para finalizar, al igual que ocurría con Simple Notification Service se está utilizando un entorno sandbox para el envío de correos y solo se podrán enviar a aquellos destinatarios que estén previamente verificados. Aunque en este caso aun no estando en el entorno de pruebas se ha de verificar un dominio o dirección de correo para usarla como emisor en los envíos. También se cuenta con un script llamado `SendEmail.py` para publicar correos directamente y probar el servicio independientemente.

3.3 Notificaciones recibidas

Tras simular un escenario donde se realiza una detección el usuario recibe dos tipos de notificaciones, la primera un SMS donde le indica que compruebe el correo electrónico para que vea una captura de la toma realizada.

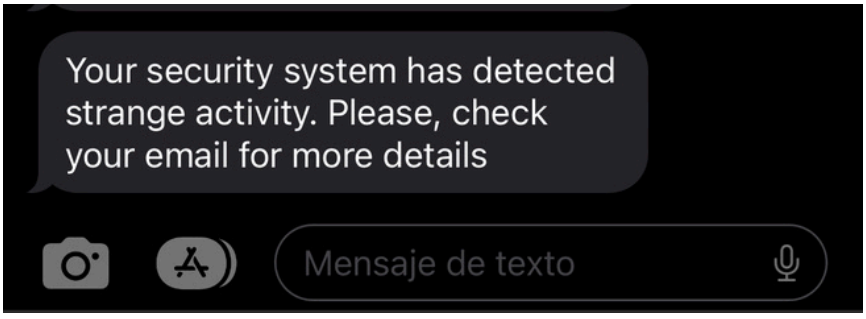


Figura 29 SMS recibido tras una detección

En la bandeja de entrada encontrará un correo bajo el asunto Security Alert con una imagen de la detección como la siguiente:

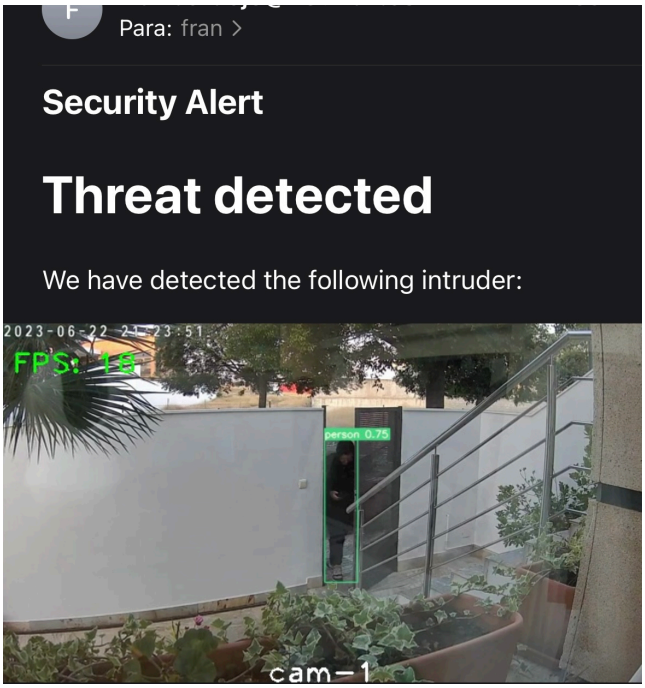


Figura 30 Correo recibido tras detección

Para finalizar podrá revisar los clips de video guardados con la detección en el servidor local bajo el directorio `Detections/video` dichos clips se guardan con el nombre de la cámara que realizó la detección y la fecha y hora.

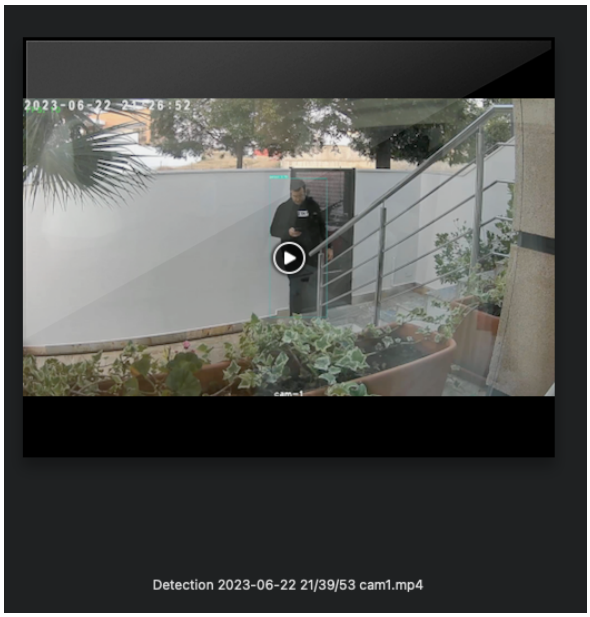


Figura 31 Clip de video guardado tras detección

En conclusión se ha realizado una explicación del funcionamiento, configuración y puesta en marcha del proyecto detallando sus componentes y que función cumplen. Al dividir las funciones de notificación y detección en dos bloques distintos e independientes entre si nos aporta mayor flexibilidad. Haciendo que se asemeje al comportamiento de un sistema comercial ya que se pueden desplegar distintos sistemas de detección pero mantener únicamente un servicio de notificación en la nube para todos.

4 PRUEBAS DE RENDIMIENTO

Se han realizado varias pruebas de rendimiento utilizando diferentes componentes hardware para estudiar la fluidez y viabilidad de un sistema de videovigilancia que utilice modelos de visión por ordenador para realizar detecciones específicas. Existen dos maneras distintas de realizar todo el cómputo del procesamiento de las imágenes. La primera es usando el procesador o CPU del propio dispositivo y la segunda es usar hardware específico como tarjetas gráficas que ayudan a acelerar el cómputo.

Para las pruebas contamos con dos tarjetas gráficas de la marca Nvidia, la GTX 1070ti de generación más antigua que sirve de gama de entrada y RTX 4070 de última generación de una gama más alta. Y para procesadores se ha utilizado una CPU de la marca AMD ryzen 5 2600 y un macbook con el chip M1. Aunque tanto la RTX 4070 y el chip M1 cuenta con hardware específico para uso de inteligencia artificial este no será empleado.

Como entorno de pruebas se ha utilizado un vídeo de resolución 1280x720p. Se han empleado los modelos yolov7 de 36,9M de parámetros, yolov7-tiny con 6,2M de parámetros y yolov7-e6e con 151,7M de parámetros. Como podemos observar en la Tabla 3 la diferencias entre estos modelos recaen en la precisión, el modelo más grande yolov7-e6e cuenta con una precisión mayor, incluso con archivos de mayor tamaño que el resto, pero a costa de la velocidad. En cambio yolov7-tiny es menos preciso pero cuenta con una velocidad mucho mayor. El modelo yolov7 se queda en un punto intermedio de equilibrio entre mayor precisión y rendimiento.

El método de medición de rendimiento empleado será obtener la media de fotogramas por segundo (fps) que obtenemos al utilizar un stream de video. Se incrementarán el número de streams para estudiar la caída de rendimiento a medida que aumenta los streams. Dichos streams representan una cámara de seguridad por lo que tras las pruebas podemos saber cuál es el número de cámaras que podemos agregar a nuestro sistema manteniendo una tasa de imágenes aceptables según el hardware que tengamos.

4.1 Pruebas realizadas sobre CPU

Como se ha comentado se ha realizado las pruebas de los tres modelos bajo las CPU M1 de Apple y Ryzen 5 2600. Se han tomado seis mediciones por CPU y modelo, incrementando el número de streams de uno en uno y obteniendo la media de fotogramas.

4.1.1 Ryzen 2600

Tabla 6 Rendimiento Ryzen 2600

Número de streams	yolov7-tiny	yolov7	yolov7-e6e
1	16,76 fps	2,88 fps	0,94 fps
2	7,49 fps	1,00 fps	0 fps
3	4,06 fps	0,18 fps	0 fps
4	3,27 fps	0 fps	0 fps
5	2,27 fps	0 fps	0 fps
6	1,76 fps	0 fps	0 fps

El mínimo recomendable para una imagen de seguridad con cierta fluidez son 15 fps por lo que solo tenemos la posibilidad de tener una cámara utilizando yolov7-tiny. En el caso de yolov7 y yolov7-e6e vemos que los resultados son muy pobres y la viabilidad de implementar un sistema de detección de objetos en tiempo real con CPU resulta, por el momento, no viable.

4.1.2 Apple M1

Tabla 7 Rendimiento M1

Número de streams	yolov7-tiny	yolov7	yolov7-e6e
1	20,64 fps	2,95 fps	1,98 fps
2	9,10 fps	1,00 fps	0,43 fps
3	5,97 fps	0,89 fps	0 fps
4	4,05 fps	0 fps	0 fps
5	3,63 fps	0 fps	0 fps
6	2,97 fps	0 fps	0 fps

Al igual que en el apartado anterior los resultados son demasiados pobres para aplicaciones en tipo real a excepción de 1 flujo de imágenes con yolov7-tiny. Aunque muestra buenos resultados tratándose de un equipo portátil el cual tiene grandes restricciones de temperatura y potencia.

4.2 Pruebas realizadas sobre GPU

A diferencia de los procesadores, las tarjetas de vídeo están orientadas a un tipo de cálculo especifico y son mucho más rápidas y eficaces en esta tarea. Las librerías con las que se construyen este tipo de modelos como TensorFlow o Pytorch tienen soporte para sacar provecho a este hardware. Hoy en día gran parte de las Inteligencias Artificiales utilizan este tipo de tarjetas e incluso existe hardware específico para el uso de IA.

Si observamos los grandes proveedores de servicios de computación en la nube como AWS, Google Cloud o Azure cada una ofrece este tipo de hardware para entrenar y ejecutar todo tipo de modelos. En este trabajo solo se ha podido emplear tarjetas de uso doméstico.

Para estas pruebas a diferencia de las anteriores, aumentaremos el número de streams de cinco en cinco.

4.2.1 GTX 1070ti

Tabla 8 Rendimiento GTX 1070ti

Número de streams	yolov7-tiny	yolov7	yolov7-e6e
1	115,37 fps	30,02 fps	17,92 fps
5	42,42 fps	10,16 fps	5,67 fps
10	19,70 fps	4,79 fps	2,43 fps
15	11,78 fps	2,97 fps	1,92 fps
20	7,19 fps	1,27 fps	0,98 fps
25	5,67 fps	1,01 fps	0,93 fps

Respecto a los resultados anteriores vemos multiplicado el rendimiento a más de 10 veces en la mayoría de los casos. Bajo este equipo podemos obtener entre 10-15 cámaras simultaneas funcionando al mismo tiempo con yolov7-tiny, entre 1-5 cámaras con yolov7 e incluso hasta una cámara con yolov7-6e6. Con estos datos en cuenta se podría tener un sistema económico que pudiera dar servicio a una casa o negocio pequeño/mediano.

4.2.2 RTX 4070

Tabla 9 Rendimiento RTX 4070

Número de streams	yolov7-tiny	yolov7	yolov7-e6e
1	198,6 fps	81,05 fps	27,47 fps
5	112,06 fps	29,08 fps	12,85 fps
10	51,93 fps	15,37 fps	6,98 fps
15	34,78 fps	8,73 fps	4,65 fps
20	25,02 fps	5,49 fps	2,79 fps
25	18,26 fps	3,89 fps	1,98 fps

Vemos un incremento sustancial respecto a la tarjeta anterior ya que esta es más moderna y cuenta con un poder de cómputo superior. Contamos con la posibilidad de usar más de 25 cámaras con yolov7-tiny de manera fluida, hasta 10 cámaras con yolov7 y entre 1 y 5 cámaras con yolov6-e6e.

Tras los resultados obtenidos podemos concluir que cualquier aplicación en tiempo real con más de una cámara que vaya a hacer uso de Yolov7 debe de ejecutarse en un hardware que cuente con tarjeta gráfica o emplear algún servicio de computación en la nube que ofrezca buen rendimiento. Sin embargo para tareas puntuales o que no requieran de un procesamiento en tiempo real las CPU pueden lograr buenos resultados como el procesamiento de una sola imagen o un vídeo.

Si tenemos en cuenta los datos obtenidos con GPU y el modelo yolov7-tiny podemos obtener escenarios con 10-15 cámaras en el caso de contar con una tarjeta de gama de entrada o de más de 25 con una tarjeta más nueva. Por lo que podrían ser opciones interesantes para dar cobertura a una vivienda, pequeño negocio u otras infraestructuras que no requieran un gran número de cámaras.

Gracias a los avances de nuevos modelos y evolución del hardware podría ser más común ver sistemas de videovigilancia que hagan uso de modelos de visión por ordenador para evitar tener falsas notificaciones ya que podrían asegurarse de detectar a una persona y no a algún animal o movimiento extraño.

5 CONCLUSIONES

Para finalizar se realizará una valoración sobre si los objetivos planteados inicialmente han podido ser realizados. Y se abrirá una venta a líneas futuras de avance sobre las que se pueden seguir trabajando, tomando como punto de partida este proyecto.

El objetivo principal era crear una aplicación de videovigilancia funcional. Con dos bloques independientes, responsables de realizar uno la función de detección y otro la función de notificación. Los cuales han sido cumplidos mediante la creación de una aplicación que se ejecuta en un servidor local y se encarga de detectar personas mediante Yolov7, guardar los clips de video de las detecciones y de activar el sistema de notificación. Y un sistema de notificación implementado en AWS, el cual recibe las peticiones de las aplicaciones y envía las notificaciones a los usuarios por correo y SMS.

Ambos bloques funcionan de forma independiente y se comunican mediante una API HTTP la cual solo acepta peticiones que lleven una llave concreta, que se utiliza como método de seguridad para que terceros no hagan uso de ella.

También se ha realizado un estudio para dimensionar la cantidad cámaras que puede soportar un equipo y poder comprobar en qué casos de uso sería viable implementar un sistema de estas características. Teniendo en el caso más favorable alrededor de 25 cámaras simultaneas.

Por lo que se puede concluir que la implementación de sistema de videovigilancia con Yolov7 es viable para un gran grupo de casos de uso como casas y pequeños negocios.

Por último, había dos requisitos. Disminuir al mínimo los costes de uso, los únicos costes de implementación y uso que hay que pagar al utilizar la aplicación son los SMS enviados a través de AWS, los cuales tienen un coste de 0,06087$ por cada uno. La capa gratuita nos permite utilizar el resto de los servicios con unos límites bastante amplios que no han sido excedidos durante el desarrollo del trabajo.

El segundo requisito, que sea fácilmente replicable bajo cualquier hardware, también ha sido cumplido ya que podemos contar con alternativas como VLC que simula transmisiones RTSP si no contamos con cámaras compatibles. Aunque no se cuente con un hardware potente o tarjeta gráfica para un uso básico se puede utilizar la aplicación con Yolov7-tiny bajo cpu y obtener unos resultados aceptables bajo un solo stream de vídeo.

5.1 Líneas futuras

Como posibles líneas de avance que resultan interesantes implementar y mejoras podemos destacar las siguientes:

- Implementación de interfaz gráfica donde se puedan ver y manejar los flujos de video de manera ordenada y mejorar la interacción con el usuario.
- Añadir soporte para que utilizando el mismo sistema en la nube y mediante la API soporte la integración de diferentes aplicaciones simultaneas y se establezca un sistema de cuotas y facturación de esta. Tal y como hacen los sistemas comerciales equivalentes.
- Estudiar la viabilidad de la implementación en un dispositivo como Jetson Nano o Raspberry Pi e integrarlo junto a una cámara para tener un sistema cerrado.

REFERENCIAS

1. **McCarthy, John.** WHAT IS ARTIFICIAL INTELLIGENCE? Stanford : Stanford University, 2007.

2. **ChatGPT.** openai. [Online] Mayo 2023. https://chat.openai.com.

3. **Caparrini, Fernando Sancho.** Redes Neuronales: una visión superficial. [Online] 2023. http://www.cs.us.es/~fsancho/?e=72.

4. **OpenAI.** openAI. [Online] https://openai.com.

5. **Google.** Bard. [Online] https://bard.google.com/?hl=en.

6. **Nvidia.** Nvidia DLSS. [Online] https://www.nvidia.com/es-es/geforce/technologies/dlss/.

7. **Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik.** Rich feature hierarchies for accurate object detection and semantic segmentation. [Online] https://doi.org/10.48550/arXiv.1311.2524.

8. **Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun.** Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. [Online] https://doi.org/10.48550/arXiv.1506.01497 .

9. **Joseph Redmon, Ali Farhadi.** YOLOv3: An Incremental Improvement. [Online] https://doi.org/10.48550/arXiv.1804.02767 .

10. **Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy.** Speed/accuracy trade-offs for modern convolutional object detectors. [Online] https://doi.org/10.48550/arXiv.1611.10012 .

11. **Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg.** SSD: Single Shot MultiBox Detector. [Online] https://doi.org/10.48550/arXiv.1512.02325.

12. **Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao.** YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. [Online] https://doi.org/10.48550/arXiv.2207.02696 .

13. **Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.** You Only Look Once: Unified, Real-Time Object Detection. [Online] https://doi.org/10.48550/arXiv.1506.02640.

14. **Common Objects in Context. [Online] https://cocodataset.org/#home.**

15. Amazon Web Services. [Online] https://aws.amazon.com/es/.

16. Azure. [Online] https://azure.microsoft.com/es-es/.

17. Google Cloud. [Online] https://cloud.google.com/?hl=es.

18. MORENO PRIETO, Á., SIERRA COLLADO, A.J. y MARTÍN RODRÍGUEZ, Á. ***Detección de Objetos con TinyYOLOv3 sobre Raspberry Pi 3 Trabajo Fin de Grado.*** **Sevilla : s.n., 2020.**

19. MUÑIZ GARCÍA, L. y SIERRA COLLADO, A.J. ***Viabilidad y rendimiento de YOLOv5 en Raspberry Pi 4 modelo B Trabajo Fin de Grado.*** **Sevilla : s.n., 2021.**

20. GARCÍA HERNÁNDEZ, A. y SIERRA COLLADO, A.J. ***Ajuste de la eficiencia en redes neuronales para la detección de señales de tráfico comparativa de rendimiento de Yolov-3 y EfficientDet : Trabajo Fin de Grado.*** **Sevilla : s.n., 2021.**

21. NVIDIA CUDA Installation Guide for Linux. [Online] https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#.

22. WongKinYiu. [Online] https://github.com/WongKinYiu/yolov7.

23. SenGrid. [Online] https://sendgrid.com.

24. NVIDIA CUDA Installation Guide for Linux. [Online] https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#.