



Escuela Técnica Superior de
Ingeniería Informática

TRABAJO FIN DE GRADO

LiveGuardian: Sistema de videovigilancia mediante el uso de DeepLearning

Realizado por
José Joaquín Virtudes Castro

Para la obtención del título de
Grado en Ingeniería Informática - Tecnologías Informáticas

Dirigido por
Dr. Juan Antonio Álvarez García

Realizado en el departamento de
Lenguajes y Sistemas Informáticos

Convocatoria de Junio, curso 2022/23

Agradecimientos

Tras cuatro años duros de carrera, termina aquí este viaje junto a este proyecto. Durante estos cuatro años, he aprendido mucho y conocido a personas que no sabía que acabarían siendo una parte tan importante de mi vida. Quiero agradecer a mi familia, mis amigos y compañeros por estar siempre, apoyarme y aconsejarme. A mi pareja, por su comprensión y esfuerzo en animarme en los peores momentos. Por último, pero no menos importante, deseo reconocer y agradecer a mi tutor, Juan Antonio, por guiarme durante el desarrollo de este proyecto.

Resumen

En cualquier sociedad es importante garantizar la seguridad y protección de los ciudadanos. Desafortunadamente, los recientes ataques armados en diversas partes del mundo han puesto en evidencia que este es un problema que se encuentra todavía por resolver. Una prevención y detección temprana pueden ser la clave para evitar consecuencias catastróficas y salvar vidas. En los últimos años, la utilización de cámaras de seguridad y detectores de objetos se ha popularizado para la detección de mascarillas en lugares públicos debido al COVID-19, y han demostrado todo el potencial de estas tecnologías.

La idea principal de este proyecto es desarrollar una aplicación desde la que poder monitorizar, detectar y alarma sobre actividades violentas observadas por cámaras de seguridad. De forma secundaria, otro de los propósitos de esta aplicación es agilizar el proceso de prueba de modelos de Visión por Computador, permitiendo desplegar en esta varios modelos a la vez en multiples o una única cámara, además de permitir almacenar datos sobre las detecciones con la intención de ser utilizados en un futuro para mejorar los mismos.

Abstract

In any society, it's crucial to ensure the safety and security of its citizens. Sadly, recent armed attacks in different parts of the world have exposed the ongoing challenge in achieving this. The key to avoiding catastrophic consequences and saving lives lies in early prevention and detection. As a response, the use of security cameras and object detectors has become increasingly prevalent, such as in the case of detecting masks in public places during the COVID-19 pandemic, demonstrating the vast potential of such technologies.

The primary objective of this project is to develop an application that can monitor, detect, and alert authorities to violent activities observed by security cameras. In addition, the application seeks to streamline the process of testing Computer Vision models by enabling multiple models to be deployed on one or multiple cameras simultaneously, and by storing detection data for future improvements.

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Contexto en detección de objetos	2
1.4. Aplicaciones similares	2
1.4.1. Blue Iris	2
1.4.2. ZoneMinder	3
1.4.3. iSpy	4
1.4.4. Xeoma	6
1.4.5. Comparativa final	6
1.5. Metodología a seguir	7
1.5.1. Metodologías ágiles	7
1.5.2. Jira en el proyecto	7
1.5.3. Gitflow	10
1.5.4. Gitflow en el proyecto	12
2. Planificación y costes	14
2.1. Introducción	14
2.2. Planificación temporal	14
2.2.1. Estimación de la duración	15
2.2.2. Sprints durante el desarrollo	15
2.3. Coste de personal	18
2.4. Coste de hardware	19
2.5. Coste total	20
2.6. Errores de planificación	21
3. Análisis de requisitos	22
3.1. Organizaciones y participantes	22
3.2. Objetivos	24
3.3. Requisitos de información	25
3.4. Requisitos funcionales	27
3.5. Requisitos no funcionales	29
3.6. Casos de uso	31
4. Tecnologías utilizadas	39
4.1. Primer bloque: Gestión web	39
4.1.1. Python	39
4.1.2. Django	39
4.1.3. JavaScript	40
4.1.4. Boostrap	41
4.1.5. SQLite	41

4.1.6. JSON	42
4.2. Segundo bloque: Envío de alertas	42
4.2.1. Telegram API	42
4.2.2. pyTelegramBotAPI	43
4.3. Tercer bloque: Detección de objetos	43
4.3.1. OpenCV	43
4.3.2. YoloV5	44
4.3.3. IceVision	47
5. Diseño del sistema	48
5.1. Primer bloque: Gestión web	49
5.2. Segundo bloque: Envío de alertas	55
5.3. Tercer bloque: Detección de objetos	56
6. Detalles de implementación	61
6.1. Primer bloque: Gestión web	61
6.1.1. Extracción del vídeo en tiempo real	61
6.1.2. Gestión de instancias VideoCamera	62
6.1.3. Gestión de modelos y detecciones	62
6.2. Segundo bloque: Envío de alertas	63
6.2.1. Bot de Telegram	63
6.3. Tercer bloque: Detección de objetos	65
6.3.1. Integración de Icevision con Django	65
6.3.2. Adaptaciones a la arquitectura de inferencia	65
6.3.3. Paciencia en las detecciones	66
7. Pruebas	68
7.1. Pruebas funcionales	68
7.2. Pruebas no funcionales	69
7.2.1. Interfaz	70
7.2.2. Inferencia	71
7.3. Pruebas de validación del usuario	71
7.4. Conclusiones	71
8. Conclusiones y trabajo futuro	73
8.1. Conclusiones	73
8.2. Trabajo futuro	73
9. Bibliografía	75
A. Manual de usuario	77
A.1. Interfaz de usuario	77
A.1.1. Dashboard	77
A.1.2. Detalles de una cámara	79
A.1.3. Creación/Edición de una cámara	80
A.1.4. Configuración	82
A.1.5. Detectores	83
A.1.6. Crear detección	84

A.1.7. Detecciones	84
A.1.8. Detección detallada	85
A.1.9. Telegram	86
A.1.10. Crear usuario de Telegram	86
A.1.11. Crear grupo de Telegram	87
A.1.12. Mensaje de Telegram	88
B. Instalación	89
B.0.1. Versiones	89
B.0.2. Python	89
B.0.3. CUDA	90
B.0.4. IceVision	91
B.0.5. Django	92
B.0.6. Pillow	92
B.0.7. OpenCV	92
B.0.8. pyTelegramBotApi	92
B.1. Puesta en marcha	93

Índice de figuras

1.1.	Dashboard Blue Iris	3
1.2.	Dashboard web Blue Iris	3
1.3.	Logo Blue Iris	3
1.4.	Dashboard ZoneMinder	4
1.5.	Instalación plugin IA ZoneMinder	4
1.6.	Logo ZoneMinder	4
1.7.	Configuración de una cámara iSpy	5
1.8.	Configuración general iSpy	5
1.9.	Logo iSpy	5
1.10.	Dashboard Xeoma	6
1.11.	Detección Xeoma	6
1.12.	Logo Xeoma	6
1.13.	Epics del proyecto	8
1.14.	Backlogs Jira	8
1.15.	Tarea de configuración	9
1.16.	Tablero Jira para el último sprint	10
1.17.	Gitflow ramas principales	11
1.18.	Gitflow ramas de función	11
1.19.	Gitflow ramas de corrección	12
1.20.	Ramas activas	12
1.21.	Commits de merges a Develop	13
1.22.	Grafo Git	13
1.23.	Release con tags	13
2.1.	Diagrama de Gantt	15
2.2.	Ejemplo de dataset con bounding boxes en Roboflow	16
2.3.	Falso positivo, dispositivo móvil	17
4.1.	Modelo-Vista-Controlador	40
4.2.	Diagrama API Telegram	43
4.3.	Ejemplo recuadro detección OpenCV	44
4.4.	Funcionamiento YoloV5	45
4.5.	Arquitectura YoloV5	46
4.6.	Comparación YoloV5	46
4.7.	Leaky ReLU	47
4.8.	Sigmoide	47
5.1.	Conexiones entre bloques	48
5.2.	Árbol de ficheros del proyecto	49
5.3.	Panel de administración Django	50
5.4.	VideoCamera	51
5.5.	CamCache	52

5.6.	AppConfig	53
5.7.	ERD DB	54
5.8.	Views	55
5.9.	Telegram	56
5.10.	Train dataset	57
5.11.	Objetos en COCO	57
5.12.	Dataset Error01	58
5.13.	Dataset Error02	58
5.14.	Split conjunto de datos	58
5.15.	Imágenes de armas etiquetadas	58
5.16.	Parser personalizado	59
5.17.	Split conjunto de datos negativo	59
5.18.	Imágenes negativas	59
5.19.	Learning Rate	60
5.20.	Guardar modelo entrenado	60
6.1.	Crear bot de Telegram	64
6.2.	Editar foto de perfil	64
6.3.	Sistema de paciencia. Umbral: 6. Historial: 12	67
7.1.	Prueba visual 1	68
7.2.	Prueba visual 2	68
7.3.	Prueba visual 3	69
7.4.	Prueba visual 4	69
7.5.	Recepción de alerta en Telegram	69
7.6.	Interfaz adaptada a dispositivo móvil	70
7.7.	CSS validado por W3C	71
A.1.	Mapa conceptual de la interfaz	77
A.2.	Dashboard sin cámaras	78
A.3.	Dashboard con cámaras	78
A.4.	Previsualización cámara	79
A.5.	Animación borrar	79
A.6.	Cámara fallando	79
A.7.	Detalles cámara	80
A.8.	Panel desplegado	80
A.9.	Panel oculto	80
A.10.	Creación de una cámara	81
A.11.	Editar una cámara	81
A.12.	Selector de detector en formulario cámara	82
A.13.	Configuración de la aplicación	83
A.14.	Lista de detectores	83
A.15.	Subir un detector	84
A.16.	Lista de detecciones	85
A.17.	Detalles de una detección	85
A.18.	Usuarios y grupos de Telegram	86
A.19.	Crear usuario de Telegram	87
A.20.	Crear grupo de Telegram	87

A.21.Chat Telegram	88
------------------------------	----

Índice de cuadros

1.1. Comparativa de aplicaciones	7
2.1. Estimación de la duración del TFG	15
2.2. Estimación de la duración de los sprints	16
2.3. Salarios medios en España	19
2.4. Estimación del coste del personal	19
2.5. Costes totales del proyecto	20
2.6. Desviación total	21
3.1. ACT-0001 Usuario	22
3.2. Organización: Departamento de Lenguajes y Sistemas Informáticos	22
3.3. Organización: DeepKnowledge	22
3.4. Participante: Juan Antonio Álvarez García	23
3.5. Participante: José Joaquín Virtudes Castro	23
3.6. Participante: José Luis Salazar González	23
3.7. Participante: M ^a de Lourdes Linares Barrera	23
3.8. OBJ-0001 Detectar objetos	24
3.9. OBJ-0002 Envío de alerta	24
3.10. OBJ-0003 Visualización de cámaras en tiempo real	24
3.11. OBJ-0004 Visualización de detecciones pasadas	24
3.12. IRQ-0001 Cámara	25
3.13. IRQ-0002 Detector	25
3.14. IRQ-0003 Detección	26
3.15. IRQ-0004 Usuario	26
3.16. IRQ-0005 Grupo	26
3.17. FRQ-0001 Manipulación de cámaras	27
3.18. FRQ-0002 Cargar modelos	27
3.19. FRQ-0003 Gestionar modelos	27
3.20. FRQ-0004 Añadir cámaras	27
3.21. FRQ-0005 Eliminar detecciones	28
3.22. FRQ-0006 Crear usuarios y grupos	28
3.23. FRQ-0007 Asignar grupos	28
3.24. FRQ-0008 Almacenar detección	28
3.25. FRQ-0009 Gestión de errores	29
3.26. FRQ-0010 Aplicar configuraciones	29
3.27. NFR-0001 Compatibilidad host	29
3.28. NFR-0002 Interfaz multiplataforma	30
3.29. NFR-0003 Tiempo de inferencia	30
3.30. NFR-0004 Falsos positivos	30
3.31. UC-0001 Cargar un modelo	31
3.32. UC-0002 Crear una cámara	32

3.33. UC-0003 Visualizar una detección	33
3.34. UC-0004 Crear un usuario	34
3.35. UC-0005 Crear un grupo	35
3.36. UC-0006 Eliminar una cámara	36
3.37. UC-0007 Eliminar usuario o grupo de Telegram	37
3.38. UC-0008 Cambiar la configuración	38

Índice de extractos de código

4.1. Menú dinámico JavaScript	40
4.2. Ejemplo Boostrap en HTML	41
4.3. JSON configuración	42
4.4. Encuadre de objetos OpenCV	44
5.1. Formulario Django	52
5.2. Modelo Django	53
5.3. Vista para generar video en tiempo real	55
6.1. Creación hilo VideoCamara	61
6.2. Extracción de fotograma sin detección.	61
6.3. Generar streaming.	62
6.4. Receiver para la eliminación de detecciones.	63
6.5. Carga del modelo Icevision.	65
B.1. Crear entorno virtual	90
B.2. Instalar drivers NVIDIA	90
B.3. Instalar CUDA Toolkit	90
B.4. Comprobar versión instalada	90
B.5. Instalar torch y torchvision	91
B.6. Instalación opcional Icevision	91
B.7. Instalación Icevision	91
B.8. Instalar Django	92
B.9. Instalar Pillow	92
B.10. Instalar openCV	92
B.11. Instalar pyTelegramBotAPI	92
B.12. Inicializar base de datos	93
B.13. Editar settings.py	93
B.14. Inicializar base de datos	93

1. Introducción

Este capítulo tiene como propósito exponer las motivaciones y objetivos generales del proyecto, comparando la aplicación en cuestión con otras similares, y explicando la metodología de desarrollo y flujo de trabajo que se utilizará.

1.1. Motivación

Las cámaras de seguridad proporcionan una sensación de seguridad y protección a las personas y sus hogares, ya que pueden ayudar a disuadir a los delincuentes, a prevenir robos y a evitar intrusiones no deseadas.

Las técnicas de Visión por Computador son muy útiles para la implementación de sistemas de cámaras de seguridad, ya que permiten el procesamiento automático de las imágenes y la extracción de información relevante, abriendo esto un abanico de posibilidades. Estas técnicas permiten la detección y análisis automático de objetos y personas en las imágenes, incluso permitiendo poder generar alertas automáticas en caso de actividad sospechosa.

Este proyecto apunta a que cualquiera, de forma sencilla pueda desplegar un sistema de seguridad potenciado por inteligencia artificial mediante el uso de modelos entrenados personalmente, además de poder automatizar alertas basadas en la detección de objetos.

1.2. Objetivos

1. **Formarse en Deep Learning.** Estudiar el desarrollo de redes neuronales en el campo de Visión por Computador para la detección de objetos. Para ello nos centraremos en utilizar los frameworks Icevision [2] y FastAI [9].
2. **Desarrollar la aplicación.** Desarrollar una plataforma de dashboard que permita visualizar en tiempo real múltiples cámaras de seguridad. Esta aplicación se construirá utilizando Django, un framework de desarrollo web, y se integrará con SQLite para la gestión de la base de datos. Además, se utilizará la biblioteca OpenCV para el procesamiento de imágenes en tiempo real.
3. **Recopilar datos.** Búsqueda, selección y etiquetado de imágenes para construir un dataset relacionado con la detección de armas.
4. **Entrenar un modelo.** Entrenar un modelo de detección de objetos, enfocado específicamente en la detección de armas. Utilizaremos la arquitectura YoloV5, que será entrenada con el dataset previamente recopilado.

5. **Desarrollar alertas.** Implementar alertas en tiempo real utilizando la API de Telegram. Estas alertas nos notificarán sobre cualquier actividad inusual sobre las cámaras que estemos monitorizando.

1.3. Contexto en detección de objetos

El auge del **Deep Learning** en el ámbito de la *Visión por Computador* ha logrado aumentar de forma totalmente sorprendente el rendimiento y la eficacia de la detección de objetos. Hoy en día existen diversas arquitecturas de *redes neuronales convolucionales* (CNN) especializadas y pre-entrenadas para abordar este desafío. Entre ellas se encuentran *YOLO* (You Only Look Once) [25], *R-CNN* (Region-based Convolutional Neural Network), *Fast R-CNN*, *Faster R-CNN* y *SSD* (Single Shot MultiBox Detector), entre otros.

Cada una de estas arquitecturas presenta sus propias ventajas y desventajas, por ejemplo, YOLO es conocida por su alta velocidad de inferencia, provocando que sea de las mejores para detección en tiempo real. El proyecto tiene como objetivo desarrollar una aplicación de videovigilancia basada en la detección de objetos en tiempo real, tomando en consideración las ventajas y desafíos asociados a la arquitectura elegida.

Para lograr un alto rendimiento y precisión en la detección de objetos, el proyecto también contempla la implementación de técnicas de **Transfer Learning**, donde se aprovechan los modelos pre-entrenados para mejorar la capacidad de generalización y reducir el tiempo de entrenamiento. Además, se investigarán posibles optimizaciones para mejorar aún más la eficacia y el tiempo de inferencia del modelo seleccionado.

1.4. Aplicaciones similares

Como trabajo previo, se han analizado las siguientes aplicaciones son softwares de videovigilancia y seguridad que permiten la gestión de cámaras desde una interfaz, del mismo modo que plantea este TFG. Ofrecen una amplia variedad de funciones para la vigilancia en tiempo real, grabación y análisis de vídeo. Incluso algunas de ellas permiten integración con herramientas relacionadas con IA como la aplicación desarrollada en este proyecto.

1.4.1. Blue Iris

Fortalezas

1. Altamente configurable y personalizable, ofreciendo una alta flexibilidad al usuario.
2. Soporta hasta 64 cámaras simultáneamente (aunque el hardware del usuario puede ser un factor clave).

3. Permite acceso remoto.

Debilidades

1. No es gratuito. La versión básica carece de potencia y funcionalidad, además de tener un coste alto.
2. Interfaz algo difícil de utilizar para usuarios poco experimentados en este tipo de aplicaciones.
3. No cuenta con tecnologías relacionadas con la detección de objetos.

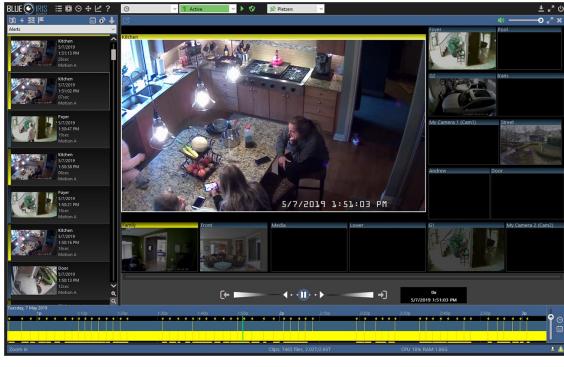


Figura 1.1: Dashboard Blue Iris

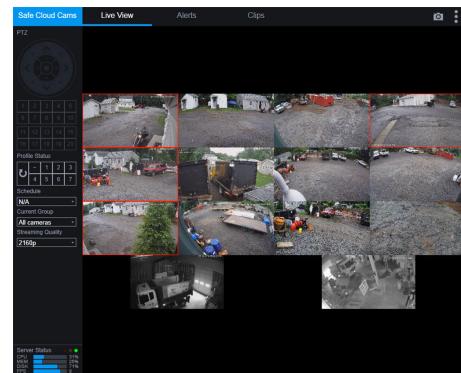


Figura 1.2: Dashboard web Blue Iris



Figura 1.3: Logo Blue Iris

1.4.2. ZoneMinder

Fortalezas

1. Software de código abierto.
2. Alta compatibilidad con distintas cámaras y servidores (Windows y distribuciones Linux).
3. Es posible la instalación de plugins que añaden funcionalidades AI desarrolladas por la comunidad.

Debilidades

1. No es fácil de configurar, requiere de tiempo y conocimiento técnico.
2. Los plugins requieren de conocimiento técnico para ser instalados, ya que es necesario manejar el código fuente de la aplicación.

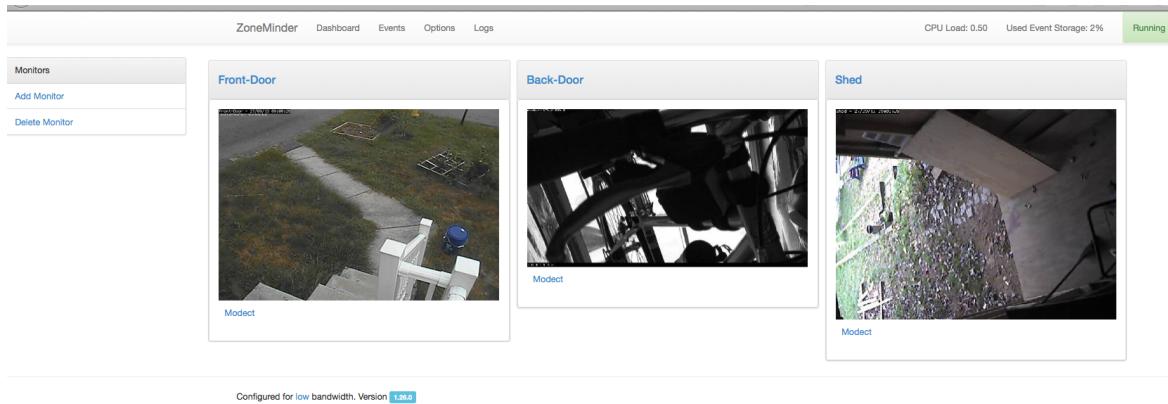


Figura 1.4: Dashboard ZoneMinder

Installation

```
# needs python3, so you may need to use pip3 if you have 2.x as well
git clone https://github.com/pliablepixels/zmMagik
cd zmMagik
# you may need to do sudo -H pip3 instead for below, if you get permission errors
pip3 install -r requirements.txt
```

Note that this package also needs OpenCV which is not installed by the above step by default. This is because you may have a GPU and may want to use GPU support. If not, pip is fine. See [this page](#) on how to install OpenCV

If you are using yolo extraction, you also need these files and make sure your config variables point to them

```
wget https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg
wget https://pjreddie.com/media/files/yolov3.weights
wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names
```

Figura 1.5: Instalación plugin IA ZoneMinder



Figura 1.6: Logo ZoneMinder

1.4.3. iSpy

Fortalezas

1. Posee una interfaz limpia y sencilla de utilizar, facilitando el uso de usuarios inexpertos.
2. Alta compatibilidad con cámaras y sistemas operativos.
3. Software de código abierto.
4. Permite acceso remoto.
5. Permite incorporar funcionalidades relacionadas con IA.

Debilidades

1. Las funcionalidades relacionadas con IA son integradas mediante proveedores externos.
2. Aunque su interfaz es clara, su configuración es compleja.
3. En caso de no realizar una configuración correcta del acceso remoto es posible tener problemas de seguridad.

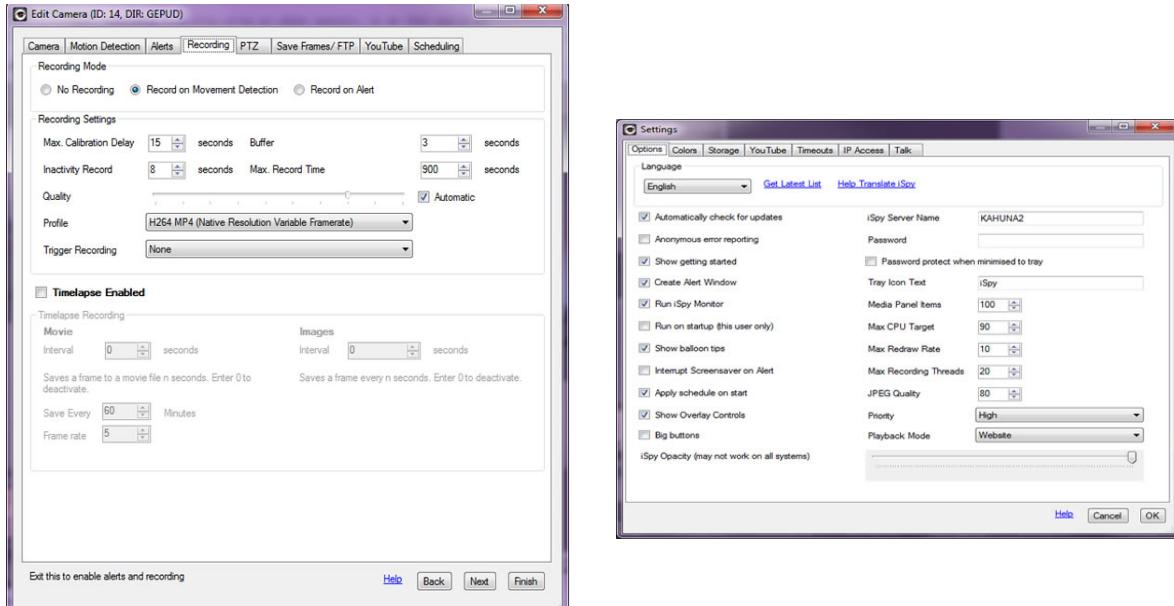


Figura 1.8: Configuración general iSpy

Figura 1.7: Configuración de una cámara iSpy



Figura 1.9: Logo iSpy

1.4.4. Xeoma

Fortalezas

1. Ofrece servicio Cloud, sin necesidad de instalación.
2. Alta integración con Smart Homes.
3. Alta variedad de funciones como detección de vehículos, detección facial, monitorización de material, etc.

Debilidades

1. Tiene un coste muy elevado.
2. Aunque tiene muchas funcionalidades, la mayor parte de estas requieren de licencias especiales o soporte personalizado.



Figura 1.10: Dashboard Xeoma



Figura 1.11: Detección Xeoma



Figura 1.12: Logo Xeoma

1.4.5. Comparativa final

A continuación se resumen las diferencias entre las aplicaciones estudiadas, y la propuesta en este TFG (Liveguardian)

Blue Iris **BI**¹, ZoneMinder **ZM**², iSpy **IS**³, Xeoma **XE**⁴ y LiveGuardian **LG**:

¹<https://blueirissoftware.com>

²<https://zoneminder.com>

³<https://www.ispyconnect.com>

⁴<https://felenasoft.com/xeoma/es/>

Característica	BI	ZM	IS	XE	LG
Configuración sencilla	✓		✓	✓	
Sin límite de cámaras		✓	✓		✓
Permite acceso remoto	✓		✓	✓	✓
Gratis		✓	✓		✓
Código abierto	✓		✓		✓
Interfaz sencilla			✓		✓
Funcionalidades IA		✓	✓	✓	✓
Compatible Linux/Windows/Mac	✓		✓		✓
Sin necesidad de conocimiento técnico	✓			✓	✓
Permite detectores propios					✓

Cuadro 1.1: Comparativa de aplicaciones

Aunque nuestra opinión no es imparcial, ya que somos los autores de LiveGuardian, nuestra aplicación es totalmente comparable e incluso superior a algunas las alternativas existentes en el mercado.

1.5. Metodología a seguir

Gestionar proyectos que incluyen tecnologías novedosas para los implicados en el desarrollo siempre es complejo, por esto es necesario tener a nuestra disposición herramientas que nos permitan adaptarnos a los cambios. Debido a esto para el desarrollo de LiveGuardian entran en escena las metodologías ágiles y Gitflow.

1.5.1. Metodologías ágiles

En el desarrollo de software, las metodologías ágiles son un conjunto de técnicas y métodos que buscan generar software funcional en una cantidad de tiempo reducida, adaptándose a los cambios, además de enfocarse a las necesidades del cliente.

Existen varias metodologías ágiles entre las que podemos elegir, las principales son **Kanban**, **Scrum**, **Lean** y **Programación extrema (XP)**. Para este proyecto se ha decidido seguir una metodología Scrum adaptada a un solo usuario. Durante el desarrollo del mismo se realizarán reuniones periódicas con el cliente, cada dos semanas, apoyadas en el desarrollo continuo basado en un Backlog y tablero Kanban en **Jira**⁵.

1.5.2. Jira en el proyecto

Jira es una herramienta de gestión de proyectos ágil ampliamente utilizada en la industria del software. Utilizaremos Jira como herramienta principal para administrar nuestra metodología Scrum adaptada. En primer lugar, creamos un proyecto en Jira y

⁵Jira Software: <https://www.atlassian.com/es/software/jira>

se configuran las opciones de administración del proyecto, incluyendo los permisos de acceso para el cliente. Seguidamente se crean Epics para diferenciar las tareas de las distintas fases del proyecto (éstas fases son explicadas en el capítulo de planificación).



Figura 1.13: Epics del proyecto

Con Jira, podemos crear y gestionar fácilmente nuestros backlogs y realizar un seguimiento del progreso del proyecto en tiempo real. En la Figura 1.14 podemos observar el Sprint Backlog para nuestro último sprint y el Product Backlog:

The screenshot displays the Jira Backlog interface. At the top, it shows the 'Backlog' section for the '4: Interfaz y backend' sprint, spanning from '24 ene – 14 mar' with 9 incidents. Below this, the 'Sprint Backlog' table lists tasks such as 'Crear interfaz creación de una cámara' (DESEARROLLO), 'Integración Bootstrap' (DESEARROLLO), and 'Crear interfaz base' (DESEARROLLO). To the right, a 'TAREAS POR HACER' column shows task status. At the bottom, the 'Product Backlog' table lists tasks like 'Memoria TFG' (DOCUMENTACIÓN) and 'Optimización detectores' (DESEARROLLO) across various sprints, with a 'TAREAS POR HACER' column on the right.

Figura 1.14: Backlogs Jira

Podemos configurar el seguimiento de tiempo para todas las tareas de nuestro proyecto. En nuestro caso, será la forma en la que faremos seguimiento del tiempo invertido en cada tarea y sprint. En la Figura 1.5.2 podemos observar como en los detalles

de una tarea tenemos información detallada sobre tiempo, sprint al que pertenece, Epic al que pertenece e incluso información detallada sobre la implementación.

The screenshot shows two Jira interface sections. On the left, under 'Crear panel de configuración', there's a description of the task: 'Este panel tendrá:' followed by a bulleted list of requirements. On the right, the task details are shown in a card format:

Detalles	
Responsable	JC José Joaquín Virtudes Castro
Seguimiento de tiempo	Registrado: 4h
Etiquetas	Ninguno
Sprint	4:Interfaz y backend
Informador	JC José Joaquín Virtudes Castro

At the bottom, there are status logs: 'Creado 30 de enero de 2023, 21:40', 'Actualizado hace 3 minutos', and 'Resuelto hace 3 minutos'. A 'Configurar' button is also present.

Figura 1.15: Tarea de configuración

Flujo de trabajo

Una vez se define e inicia un sprint, se dispone de un tablero Kanban desde el que gestionar la ejecución de las tareas del sprint. Nuestro flujo de trabajo será el siguiente:

1. Se crea una tarea y se le asigna un Epic correspondiente.
2. Si se comienza a trabajar en la tarea, se mueve de la columna “Por Hacer” a la columna “En Curso”.
3. Se registra el tiempo dedicado a la tarea en cada sesión de trabajo.
4. Una vez finalizada, se mueve la tarea a la columna “Listo”.

4:Interfaz y backend

Durante el último sprint, nos enfocaremos en el desarrollo final de la interfaz de usuario y del backend de nuestra aplicación.

The screenshot shows a Jira board with three columns:

- POR HACER 1 INCIDENCIA** (1 Incident):
 - Separar el envío de alertas de Telegram en una clase distinta. **DESARROLLO**: TFG-54. Status: JC.
- EN CURSO 3 INCIDENCIAS** (3 Incidents):
 - Cambiar estilos app **DISEÑO**: TFG-47. Status: JC.
 - Crear página de Ayuda **DISEÑO**: TFG-44. Status: JC.
 - Crear panel de configuración **DESARROLLO**: TFG-36. Status: JC.
- LISTO 5 INCIDENCIAS** (5 Incidents):
 - Crear interfaz creación de una cámara **DESARROLLO**: TFG-27. Status: ✓ JC.
 - Integración Bootstrap **DESARROLLO**: TFG-13. Status: ✓ JC.
 - Integración con Telegram **DESARROLLO**: TFG-6. Status: ✓ JC.
 - Crear interfaz base **DESARROLLO**: TFG-2. Status: ✓ JC.
 - Crear modelos Django **DESARROLLO**: TFG-1. Status: ✓ JC.

Figura 1.16: Tablero Jira para el último sprint

1.5.3. Gitflow

Gitflow^[4] es un modelo alternativo de desarrollo basado en ramas de Git. En Gitflow existen 4 tipos de ramas, la rama principal, la rama de desarrollo, las ramas de función y las ramas de corrección.

Ramas principal y de desarrollo

En lugar de existir únicamente la rama **Main**, nos encontramos también con la rama **Develop**. La rama Develop será utilizada para el desarrollo continuo de las nuevas funciones, mientras que la rama Main tan solo será utilizada como un historial de las publicaciones oficiales del proyecto.

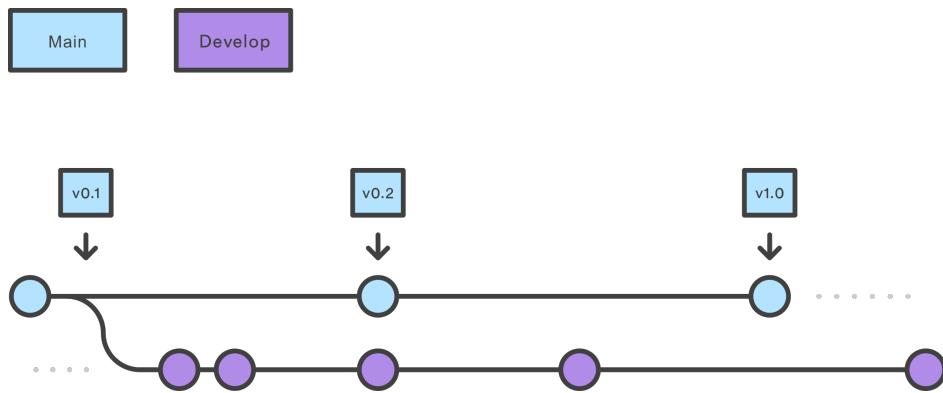


Figura 1.17: Gitflow ramas principales

Ramas de funciones

Según este modelo, cada vez que sea necesario implementar una nueva funcionalidad al proyecto se hará mediante una rama de función, la cual solo será fusionada con la rama de desarrollo cuando se haya terminado su implementación en su totalidad.

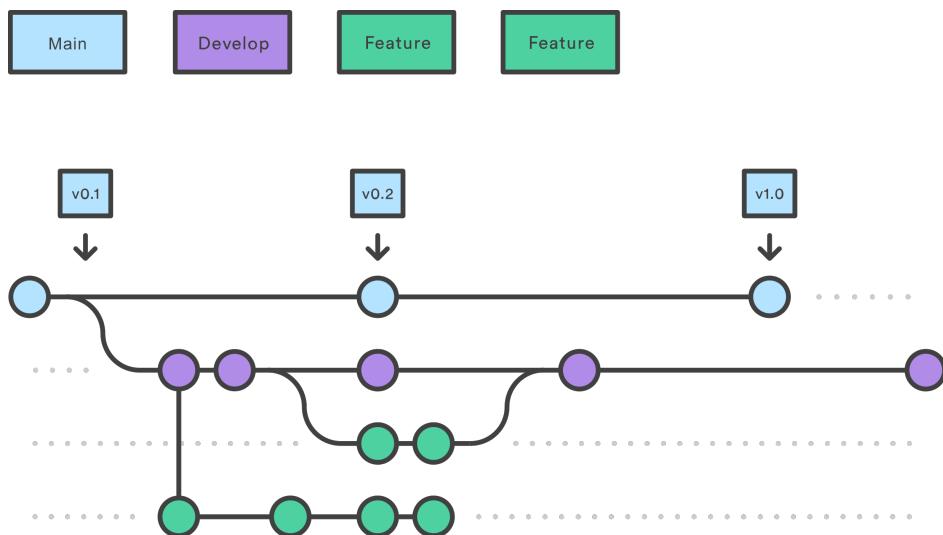


Figura 1.18: Gitflow ramas de función

Ramas de corrección

Finalmente las ramas de corrección con aquellas que sirven para arreglar de forma rápida y directa cambios críticos que se encuentran en las versiones públicas del proyecto.

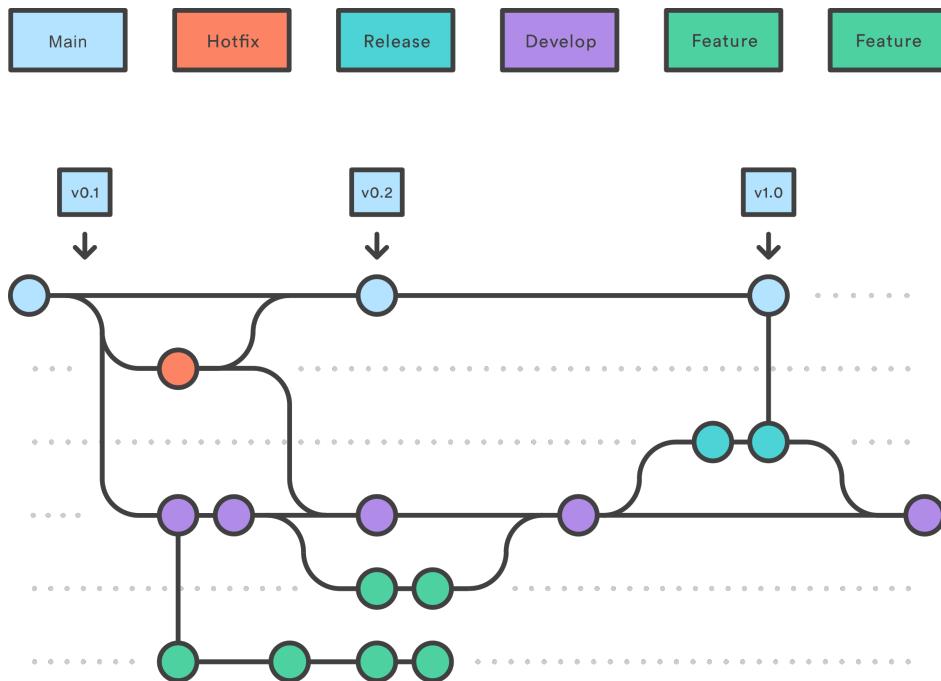


Figura 1.19: Gitflow rutas de corrección

1.5.4. Gitflow en el proyecto

En la Figura 1.5.4, se pueden apreciar las ramas activas durante la última semana del desarrollo del proyecto. Se observa la estructura previamente mencionada, compuesta por la rama Main, Develop y en este caso, una rama Feature dedicada a la refactorización de algunos componentes del proyecto.



Figura 1.20: Ramas activas

En este proyecto, el uso de este enfoque alternativo ha sido muy beneficioso para acelerar el proceso de desarrollo. El uso de ramas Feature ha permitido que la implementación del sistema de detección de objetos, que requiere obligatoriamente un equipo con tarjeta gráfica NVIDIA, se desarrolle de manera paralela e independiente al resto de la lógica de la aplicación. La Figura 1.22 ilustra claramente este desarrollo paralelo,

donde se puede observar dos ramas surgidas desde Develop, una para el desarrollo de la interfaz (rama azul) y otra para la integración de los detectores en la aplicación (rama morada).

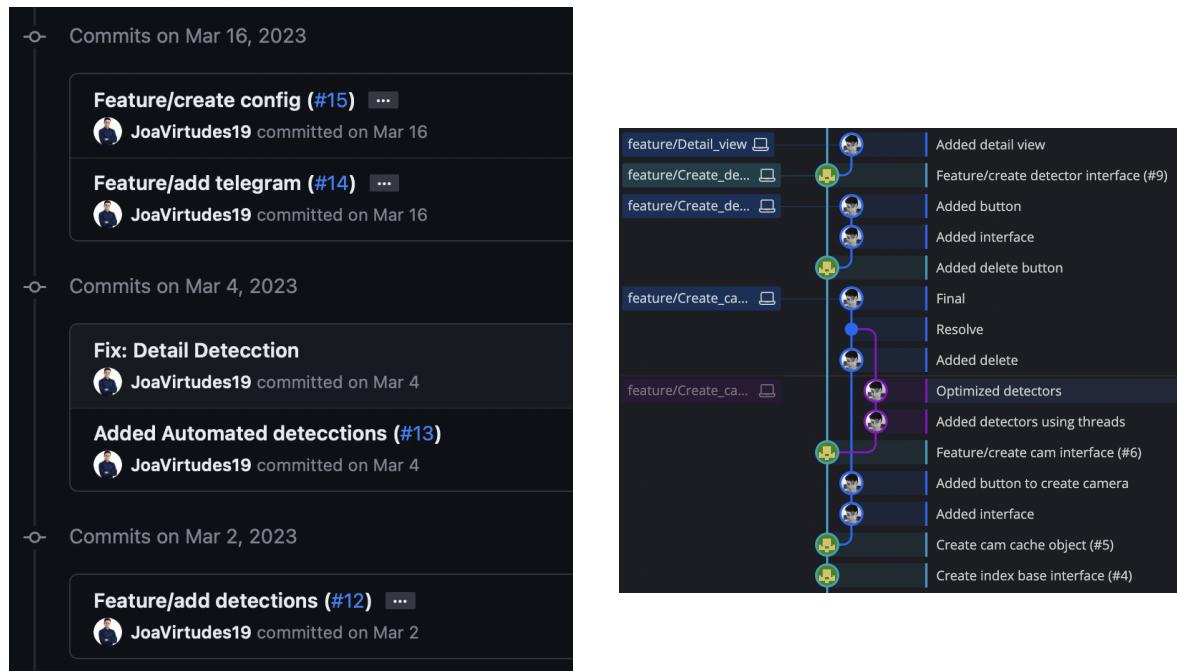


Figura 1.22: Grafo Git

Figura 1.21: Commits de merges a Develop

Finalmente, generamos versiones del proyecto a partir de la rama Main con tags.

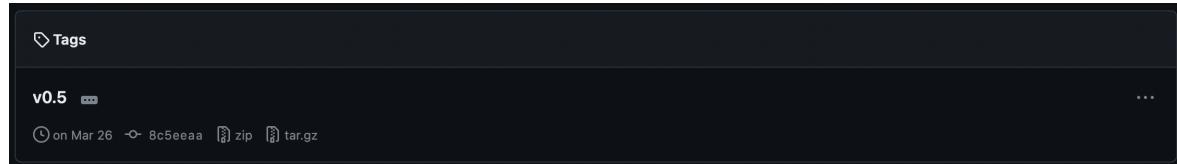


Figura 1.23: Release con tags

2. Planificación y costes

Durante este capítulo, se mostrara la planificación realizada para la realización del proyecto. Además se profundizará en costes, hitos y tiempos estimados.

2.1. Introducción

La planificación divide el proyecto en cinco fases:

- **Análisis:** esta fase se divide en dos partes, primero se realizará un estudio del problema a resolver y una formación previa [10] relacionada con las tecnologías utilizadas en el mismo (fastAI [9] y Icevision[2]). Seguidamente se determina que solución será la más adecuada y se definirán los requisitos necesarios para cumplir el proyecto.
- **Diseño:** considerando los requisitos definidos en la fase de análisis, se procede a realizar el diseño de la arquitectura del software. En esta etapa, se diseñará la estructura del sistema, se definirá la interfaz de la aplicación y el modelado de la base de datos.
- **Desarrollo:** en esta fase se realiza la implementación del software en su totalidad.
- **Pruebas:** en la fase de pruebas se comprueba el correcto funcionamiento del software en busca de posibles errores.
- **Documentación:** se documenta el software desarrollado, esta fase se realiza de forma paralela a todas las anteriores y es una de las más importantes, ya que facilita la comprensión del software y su mantenimiento,

2.2. Planificación temporal

La planificación temporal es una parte fundamental en la gestión de cualquier proyecto. Durante el proceso de desarrollo, se llevarán a cabo los sprints de manera lineal, dado que aunque existen diferentes roles en el equipo, cada tarea será realizada por una única persona.

2.2.1. Estimación de la duración

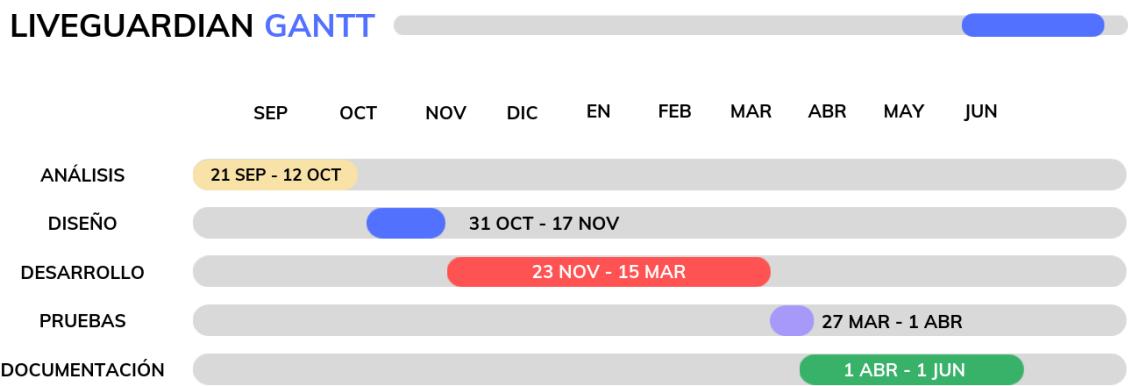


Figura 2.1: Diagrama de Gantt

Fase	Inicio/Fin	Horas
Análisis	21/09/22 a 12/10/22	25h
Diseño	31/10/22 a 17/11/22	10h
Desarrollo	23/11/22 a 15/03/23	200h
Pruebas	27/03/23 a 01/04/23	5h
Documentación	01/04/23 a 01/05/23	60h

Cuadro 2.1: Estimación de la duración del TFG

2.2.2. Sprints durante el desarrollo

En general, se plantea abordar la implementación de cada funcionalidad en diferentes sprints y desarrollarlas de manera incremental, con el objetivo de lograr una base sólida y añadir detalles posteriormente. En el proceso de desarrollo del proyecto, se han planificado cuatro sprints principales para abordar diferentes aspectos del sistema.

Sprint	Inicio	Fin	Horas
Sprint 1: Entrenamiento del modelo	23/11/22	07/12/22	30h
Sprint 2: Desarrollo de video en tiempo real	08/12/22	22/12/22	30h
Sprint 3: Detectar objetos en cámaras	09/01/23	23/01/23	60h
Sprint 4: Desarrollo de la interfaz y backend	24/01/23	15/03/23	80h
Total:			200h

Cuadro 2.2: Estimación de la duración de los sprints

Procedemos a describir cada sprint en detalle:

Sprint 1: Entrenamiento del modelo

Durante este primer sprint se pretende conseguir un modelo que permita detectar armas en las cámaras en tiempo real.

Utilizaremos el enfoque de aprendizaje supervisado, lo que implica que necesitaremos un dataset etiquetado. Cada muestra del dataset contendrá una imagen y las correspondientes etiquetas que delimitan las áreas donde se encuentran las armas (bounding boxes).

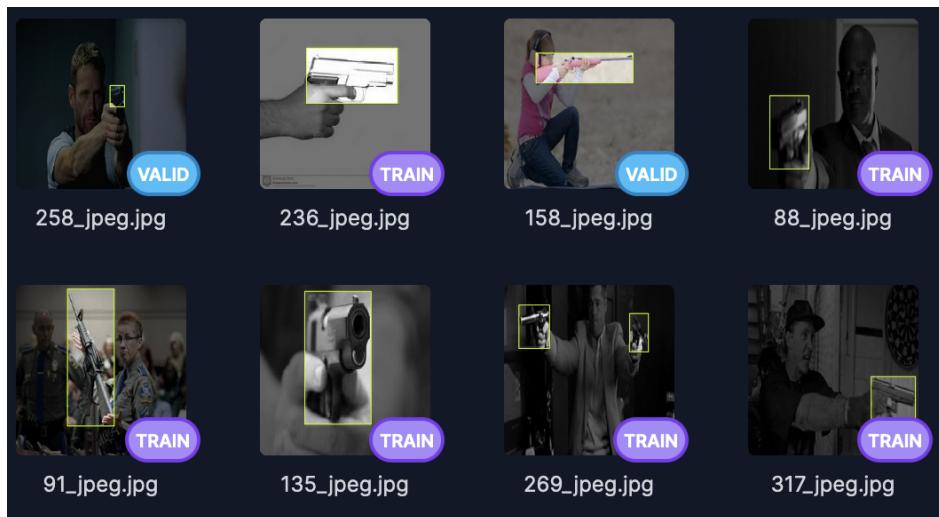


Figura 2.2: Ejemplo de dataset con bounding boxes en Roboflow

Los pasos durante este sprint son:

1. Entrenamiento inicial: Comenzaremos con un dataset pequeño para comprender el funcionamiento de la arquitectura del modelo.
2. Recopilación de datos: A continuación, recopilaremos un dataset más amplio para mejorar la precisión del modelo.

3. Construcción del modelo: Con el dataset recopilado, construiremos el modelo que se ajuste mejor a nuestras necesidades. Como veremos más adelante, el elegido finalmente es YoloV5 small.
4. Mejoras en el modelo: Por último, mejoraremos la precisión del modelo añadiendo más casos negativos, principalmente objetos de color negro que puedan ser sujetados con una mano, como dispositivos móviles o mandos.

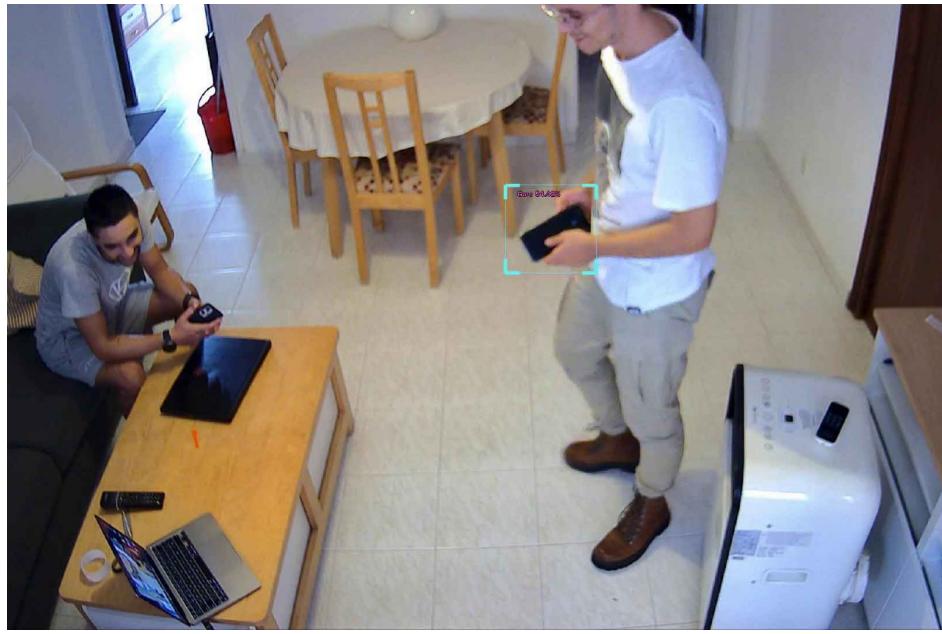


Figura 2.3: Falso positivo, dispositivo móvil

Siguiendo estos pasos, esperamos lograr un modelo preciso y veloz que nos permita detectar armas en tiempo real.

Sprint 2: Visualización de vídeo en tiempo real

Durante el segundo sprint, nos enfocaremos en desarrollar la funcionalidad base de la aplicación, que permita visualizar cámaras en tiempo real.

1. Visualización de una sola cámara: Comenzaremos por implementar la funcionalidad que permita visualizar una única cámara en tiempo real desde la aplicación.
2. Visualización de varias cámaras: A continuación, nos centraremos en permitir la visualización de varias cámaras simultáneamente desde la aplicación.
3. Funcionamiento dinámico: Finalmente, buscaremos lograr un funcionamiento dinámico en la aplicación, lo que significa que la cantidad de cámaras que se pueden reproducir en tiempo real dependerá de la cantidad de registros que tengamos en la base de datos.

Con estos pasos, esperamos lograr una aplicación que permita visualizar cámaras en tiempo real de manera eficiente y dinámica, lo que será fundamental para la implementación de nuestro dashboard de videovigilancia.

Sprint 3: Detectar objetos en cámaras

Durante el tercer sprint, nos enfocaremos en habilitar la detección de objetos en las cámaras en tiempo real.

1. Carga del modelo: Comenzaremos permitiendo la carga de un modelo entrenado desde un fichero que suba el usuario.
2. Implementación en una sola cámara: A continuación, nos enfocaremos permitir la detección de objetos en una única cámara simultanea.
3. Implementación en múltiples cámaras: Posteriormente, extenderemos la detección de objetos a varias cámaras simultaneas.
4. Optimizaciones: Finalmente, añadiremos optimizaciones que aumenten el rendimiento de las detecciones, permitiendo maximizar la cantidad de cámaras que pueden utilizar la detección de objetos en tiempo real.

Tras estos pasos, esperamos lograr una implementación con un buen rendimiento para nuestra detección de objetos en tiempo real en las cámaras.

Sprint 4: Desarrollo de la interfaz y backend

Durante el último sprint, nos enfocaremos en el desarrollo final de la interfaz de usuario y del backend de nuestra aplicación. En este sprint, ya contamos con todas las funcionalidades necesarias para la detección de armas y la visualización de cámaras en tiempo real, por lo que nuestro objetivo será completar la aplicación para su implementación y uso.

1. Interfaz de usuario final: Nos asegurándonos de que sea fácil de usar, intuitiva y atractiva visualmente. Además, integraremos todas las funcionalidades desarrolladas en sprints anteriores, para que sean accesibles desde la interfaz de usuario.
2. Backend completo: Continuaremos desarrollando el backend de la aplicación, completando todas las funcionalidades que aún no hayan sido implementadas, que en este caso, serán mayormente relacionadas con gestión de la base de datos y gestión de las cámaras.
3. Pruebas y correcciones de errores: Realizaremos pruebas en la aplicación para identificar posibles errores o fallos en el funcionamiento.

Tras este último sprint, pretendemos tener la aplicación completa y testeada, lista para ser utilizada por usuarios finales.

2.3. Coste de personal

Este proyecto involucra la participación de 2 roles, el cliente y el autor del proyecto. Aunque será realizado únicamente por el autor, la complejidad es suficiente como

invertido en los mismos:

- **PAPERSPACE:** para poder realizar un entrenamiento en tiempo y coste reducido, se ha utilizado el servicio de Cloud Computing de Paperspace. Para esto se ha utilizado la versión Gradient Pro, con un coste de 8\$ mes, durante un total de 3 meses, lo que supone un coste de 21,63€, en total.[18]
- **ORDENADOR SOBREMESA:** para las pruebas de la aplicación será necesario un ordenador con tarjeta gráfica NVIDIA, se ha utilizado un PC de sobremesa con un procesador Intel® Core™ i7-6700, una tarjeta gráfica NVIDIA GEFORCE GTX 1060 de 6GB de memoria de video, 16 GB de RAM y 1TB de almacenamiento SSD. El coste de este sobremesa es de 1.200€. Si se utiliza durante el proyecto, que tiene una duración de 8 meses y un período de amortización de 4 años, se le sumará un coste adicional de 200€.
- **ORDENADOR PORTATIL:** para el desarrollo de la aplicación sera necesario un ordenador portatil, se ha utilizado un MacBook Air, con un procesador Chip M1 de Apple, 8GB de RAM y 256GB de almacenamiento. El coste de este portátil es de 1.219€. Si se utiliza durante el proyecto, que tiene una duración de 8 meses y un período de amortización de 4 años, se le sumará un coste adicional de 203,17€.
- **WEBCAM:** para poder hacer pruebas de video en tiempo real se ha utilizado una cámara webcam Logitech C920 HD Pro Webcam, con un precio de 71€. Si se utiliza durante el proyecto, que tiene una duración de 8 meses y un período de amortización de 2 años, se le sumará un coste adicional de 23,67€.

En total acumulamos un coste de hardware de 448,47€.

2.5. Coste total

Siendo los costes indirectos aquellos que no están directamente relacionados con la producción del proyecto, como lo son luz, alquiler, servicios de internet y mantenimiento de equipos, pero que son necesarios para llevar a cabo el proyecto. Añadimos estos a los anteriormente mencionados y obtenemos un total de 7,930.39€:

Coste total	
Coste de personal	6.875,16€
Coste de hardware	448,47€
Coste indirecto	500€
Total	7.823,63€

Cuadro 2.5: Costes totales del proyecto

2.6. Errores de planificación

En la tabla 2.6 podemos observar los errores cometidos respecto a las horas estimadas en la planificación. En este caso, podemos observar que el proyecto tuvo una desviación total de 42 horas, además, la mayor desviación se produjo en el Diseño y Documentación. Las horas añadidas a Diseño se deben a un rediseño del modelo de datos relacionado con las detecciones.

Sprint	Horas estimadas	Horas reales	Desviación
Análisis	25h	20h	-5h
Diseño	10h	30h	20h
Desarrollo	200h	190h	-10h
Pruebas	5h	12h	7h
Documentación	60h	80h	20h
Total	300h	342h	42h

Cuadro 2.6: Desviación total

3. Análisis de requisitos

En este capítulo se presentarán detalladamente los requisitos distintivos de la aplicación, incluyendo tanto los requisitos de implementación como los casos de uso. Específicamente, se identificará al usuario final del sistema como ACT-0001 para la definición de los requisitos.

ACT-0001	Usuario
Descripción	Este actor representa al usuario final de la aplicación.

Cuadro 3.1: ACT-0001 Usuario

3.1. Organizaciones y participantes

Organización	Departamento de Lenguajes y Sistemas Informáticos
Dirección	Escuela Técnica Superior de Ingeniería Informática. Avenida Reina Mercedes s/n 41012 Sevilla
Telefono	(+34) 954 556 817

Cuadro 3.2: Organización: Departamento de Lenguajes y Sistemas Informáticos

Organización	Grupo de investigación DeepKnowledge
Dirección	Escuela Técnica Superior de Ingeniería Informática. Avenida Reina Mercedes s/n 41012 Sevilla
Telefono	-

Cuadro 3.3: Organización: DeepKnowledge

3.3. Requisitos de información

IRQ-0001	Cámara
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Descripción	El sistema deberá almacenar la información correspondiente a las cámaras. En concreto:
Datos específicos	<ul style="list-style-type: none">• name• detector• url

Cuadro 3.12: IRQ-0001 Cámara

IRQ-0002	Detector
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Descripción	El sistema deberá almacenar la información correspondiente a los detectores. En concreto:
Datos específicos	<ul style="list-style-type: none">• name• model

Cuadro 3.13: IRQ-0002 Detector

IRQ-0003	Detección
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Descripción	El sistema deberá almacenar la información correspondiente a las detecciones. En concreto:
Datos específicos	<ul style="list-style-type: none"> • cam • date • img • items • pred • detector

Cuadro 3.14: IRQ-0003 Detección

IRQ-0004	Usuario
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Descripción	El sistema deberá almacenar la información correspondiente a los usuarios de Telegram. En concreto:
Datos específicos	<ul style="list-style-type: none"> • name • chat_id • group

Cuadro 3.15: IRQ-0004 Usuario

IRQ-0005	Grupo
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Descripción	El sistema deberá almacenar la información correspondiente a los grupos de Telegram. En concreto:
Datos específicos	<ul style="list-style-type: none"> • name

Cuadro 3.16: IRQ-0005 Grupo

FRQ-0009	Gestión de errores
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Importancia	Vital
Descripción	El sistema deberá ser capaz de recuperarse de errores relacionados con la conexión y desconexión de cámaras.

Cuadro 3.25: FRQ-0009 Gestión de errores

FRQ-0010	Aplicar configuraciones
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Importancia	Vital
Descripción	El sistema deberá aplicar configuraciones del usuario sin necesidad de un reinicio.

Cuadro 3.26: FRQ-0010 Aplicar configuraciones

3.5. Requisitos no funcionales

NFR-0001	Compatibilidad host
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Importancia	Vital
Descripción	El sistema deberá poder ejecutarse en cualquier entorno Linux/-Windows/Mac mientras se disponga de al menos una tarjeta gráfica NVIDIA.

Cuadro 3.27: NFR-0001 Compatibilidad host

NFR-0002	Interfaz multiplataforma
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Importancia	Vital
Descripción	El sistema deberá poder utilizarse con un navegador web desde cualquier dispositivo, independientemente del tamaño o forma de su pantalla.

Cuadro 3.28: NFR-0002 Interfaz multiplataforma

NFR-0003	Tiempo de inferencia
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Importancia	Vital
Descripción	El tiempo de inferencia en cámaras que utilicen detector deberá permitir visualizar una imagen fluida en tiempo real.

Cuadro 3.29: NFR-0003 Tiempo de inferencia

NFR-0004	Falsos positivos
Versión	v1.0 (7/10/2022)
Autores	José Joaquín Virtudes Castro
Importancia	Vital
Descripción	El sistema deberá de evitar enviar y guardar alertas sobre posibles falsos positivos.

Cuadro 3.30: NFR-0004 Falsos positivos

3.6. Casos de uso

UC-0001	Cargar un modelo	
Versión	v1.0 (7/10/2022)	
Autores	José Joaquín Virtudes Castro	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera subir un modelo al mismo.	
Precondición	No existe precondición.	
Secuencia normal	Pasos	Acción
	1	El usuario (ACT-0001) selecciona el menú desplegable <i>Detectar</i> en la barra de navegación superior.
	2	El sistema muestra los elementos <i>Detectores</i> y <i>Detecciones</i> .
	3	El usuario (ACT-0001) selecciona <i>Detectores</i> .
	4	El usuario (ACT-0001) es llevado a una nueva pantalla y pulsa el botón <i>Crear detector</i> , situado en la esquina superior derecha.
	5	El sistema muestra un formulario para la creación de un detector.
	6	El usuario (ACT-0001) indica el nombre para el detector.
	7	El usuario (ACT-0001) sube el archivo del modelo.
	8	El usuario (ACT-0001) selecciona Crear.
	9	El sistema crea el detector y muestra al usuario la pantalla <i>Detectores</i> .
Postcondición	Un nuevo modelo debe de aparecer en la lista de detectores listo para usarse.	
Excepciones	Pasos	Acción
	6,7	Si el usuario (ACT-0001) no rellena los campos obligatorios, el sistema mostrará un mensaje y no creará el detector.

Cuadro 3.31: UC-0001 Cargar un modelo

UC-0002	Crear una cámara	
Versión	v1.0 (7/10/2022)	
Autores	José Joaquín Virtudes Castro	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera crear una cámara.	
Precondición	En caso de querer asignar un detector o grupo, es necesario que estos estén creados.	
Secuencia normal	Pasos	Acción
	1	El usuario (ACT-0001) selecciona <i>Dashboard</i> en la barra de navegación superior.
	2	El sistema muestra el vídeo en tiempo real de todas las cámaras creadas.
	3	El usuario (ACT-0001) selecciona el botón <i>Crear cámara</i> , situado en la esquina superior derecha.
	4	El sistema muestra un formulario para la creación de una cámara
	6	El usuario (ACT-0001) indica el nombre.
	7	El usuario (ACT-0001) selecciona un detector.
	8	El usuario (ACT-0001) indica el URL de la cámara deseada.
	9	El usuario (ACT-0001) selecciona un grupo de Telegram.
	10	El usuario (ACT-0001) selecciona el botón <i>Crear</i> .
	11	El sistema crea la cámara y muestra al usuario la pantalla <i>Dashboard</i> .
Postcondición	Una nueva cámara comienza a procesar video en tiempo real.	
Excepciones	Pasos	Acción
	7	Si el usuario (ACT-0001) no selecciona ningún detector la cámara no detectará objetos, por lo que podrá ser utilizada en sistemas sin tarjeta gráfica.
	10	Si el usuario (ACT-0001) no selecciona ningún grupo de Telegram la cámara creada no enviará alertas en caso de detectarse un objeto.
	6,8	Si el usuario (ACT-0001) no rellena los campos obligatorios, el sistema mostrará un mensaje y no creará la cámara.

Cuadro 3.32: UC-0002 Crear una cámara

UC-0003	Visualizar una detección	
Versión	v1.0 (7/10/2022)	
Autores	José Joaquín Virtudes Castro	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera visualizar una detección pasada.	
Precondición	Es necesario que se haya producido alguna detección en una cámara con un detector asignado.	
Secuencia normal	Pasos	Acción
	1	El usuario (ACT-0001) selecciona el menú desplegable <i>Detectar</i> en la barra de navegación superior.
	2	El sistema muestra los elementos <i>Detectores</i> y <i>Detecciones</i> .
	3	El usuario (ACT-0001) selecciona <i>Detecciones</i> .
	4	El sistema muestra una lista con detecciones pasadas.
	5	El usuario (ACT-0001) selecciona la palabra Ver en una de las detecciones.
	6	El sistema muestra todos los datos relacionados con la detección.
Postcondición	No existe postcondición.	
Excepciones	No se esperan excepciones.	

Cuadro 3.33: UC-0003 Visualizar una detección

UC-0004	Crear un usuario	
Versión	v1.0 (7/10/2022)	
Autores	José Joaquín Virtudes Castro	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario crea un usuario de Telegram.	
Precondición	No existe precondición.	
Secuencia normal	Pasos	Acción
	1	El usuario (ACT-0001) selecciona <i>Telegram</i> en la barra de navegación superior.
	2	El sistema muestra una lista con los usuarios y grupos de Telegram.
	3	El usuario (ACT-0001) selecciona el botón <i>Crear usuario</i> .
	4	El sistema muestra un formulario para la creación de un usuario.
	5	El usuario (ACT-0001) indica el nombre del usuario del Telegram.
	6	El usuario (ACT-0001) indica el id del usuario de Telegram.
	7	El usuario (ACT-0001) asigna un grupo de Telegram al usuario.
	8	El usuario (ACT-0001) selecciona <i>Crear</i>
	9	El sistema crea el usuario y muestra al usuario (ACT-0001) la página <i>Telegram</i> .
Postcondición	Un nuevo usuario debe de aparecer en la lista de usuarios de Telegram.	
Excepciones	Pasos	Acción
	5,6	Si el usuario no rellena los campos obligatorios, el sistema mostrará un mensaje y no creará el usuario de Telegram.

Cuadro 3.34: UC-0004 Crear un usuario

UC-0005	Crear un grupo	
Versión	v1.0 (7/10/2022)	
Autores	José Joaquín Virtudes Castro	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario crea un grupo de Telegram.	
Precondición	No existe precondición.	
Secuencia normal	Pasos	Acción
	1	El usuario (ACT-0001) selecciona <i>Telegram</i> en la barra de navegación superior.
	2	El sistema muestra una lista con los usuarios y grupos de Telegram.
	3	El usuario (ACT-0001) selecciona el botón <i>Crear grupo</i> .
	4	El sistema muestra un formulario para la creación de un grupo.
	5	El usuario (ACT-0001) indica el nombre del grupo de Telegram.
	6	El usuario (ACT-0001) selecciona <i>Crear</i>
	7	El sistema crea el grupo y muestra al usuario (ACT-0001) la página <i>Telegram</i> .
Postcondición	Un nuevo grupo debe de aparecer en la lista de grupos de Telegram.	
Excepciones	Pasos	Acción
	5	Si el usuario no rellena los campos obligatorios, el sistema mostrará un mensaje y no creará el grupo de Telegram.

Cuadro 3.35: UC-0005 Crear un grupo

UC-0006	Eliminar una cámara	
Versión	v1.0 (7/10/2022)	
Autores	José Joaquín Virtudes Castro	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario elimina una cámara.	
Precondición	Debe existir una cámara en el sistema.	
Secuencia normal	Pasos	Acción
	1	El usuario (ACT-0001) selecciona <i>Dashboard</i> en la barra de navegación superior.
	2	El sistema muestra el vídeo en tiempo real de todas las cámaras creadas.
	3	El usuario (ACT-0001) selecciona el ícono con forma de papelera de la cámara que desea eliminar.
	4	El sistema elimina la cámara seleccionada.
Postcondición	La cámara seleccionada es eliminada del sistema.	
Excepciones	No se esperan excepciones.	

Cuadro 3.36: UC-0006 Eliminar una cámara

UC-0007	Eliminar un usuario o grupo de Telegram	
Versión	v1.0 (7/10/2022)	
Autores	José Joaquín Virtudes Castro	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario elimina una cámara.	
Precondición	Debe existir un usuario/grupo en el sistema.	
Secuencia normal	Pasos	Acción
	1	El usuario (ACT-0001) selecciona <i>Telegram</i> en la barra de navegación superior.
	2	El sistema muestra una lista con los usuarios y grupos de Telegram.
	3	El usuario (ACT-0001) selecciona el ícono con forma de papelera usuario/grupo que desea eliminar.
	4	El sistema elimina el usuario/grupo seleccionado.
Postcondición	El usuario/grupo seleccionado es eliminado del sistema.	
Excepciones	No se esperan excepciones.	

Cuadro 3.37: UC-0007 Eliminar usuario o grupo de Telegram

UC-0008	Cambiar la configuración	
Versión	v1.0 (7/10/2022)	
Autores	José Joaquín Virtudes Castro	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario cambia la configuración.	
Precondición	No existe precondición.	
Secuencia normal	Pasos	Acción
	1	El usuario (ACT-0001) selecciona <i>Configuración</i> en la barra de navegación superior.
	2	El sistema muestra un formulario con la configuración y sus valores actuales
	3	El usuario (ACT-0001) modifica la configuración deseada.
	4	El usuario (ACT-0001) selecciona Guardar.
	5	El Sistema aplica los cambios en la configuración.
Postcondición	La configuración es aplicada al sistema.	
Excepciones	No se esperan excepciones.	

Cuadro 3.38: UC-0008 Cambiar la configuración

4. Tecnologías utilizadas

En este capítulo se explorarán las tecnologías utilizadas en el desarrollo de la aplicación, la cual se divide en tres bloques con funcionalidades distintas:

- El primer bloque se encarga de la gestión de la aplicación web y los datos.
- El segundo bloque está encargado de la gestión de las alertas del sistema.
- Finalmente, el tercer bloque se encarga de todo lo relacionado con la detección de objetos en las imágenes.

Para algunas de estas tecnologías, mostraremos pequeños fragmentos de código representativo. En cuanto a las demás tecnologías, se mostrará y explicará su código en capítulos posteriores.

4.1. Primer bloque: Gestión web

En esta sección nombraremos las tecnologías utilizadas para el desarrollo de la interfaz (front-end) y de la lógica de la aplicación (back-end).

4.1.1. Python

Python [19] es un lenguaje de programación de alto nivel, interpretado y con una sintaxis clara y sencilla. Es un lenguaje multiparadigma, permitiendo programación imperativa, funcional y orientada a objetos. Este será el lenguaje de programación core de nuestro proyecto, tanto para el back-end como para el envío de alertas y la detección de objetos.

4.1.2. Django

Django [8] es un framework de desarrollo web de código abierto, escrito en el Python. Su objetivo es proporcionar un conjunto de herramientas y características para ayudar a los desarrolladores a construir aplicaciones web de calidad de manera más rápida y sencilla. Django se compone de varias partes, cada una con un propósito específico en el desarrollo de aplicaciones web:

1. **Modelo:** los modelos en Django proporcionan una capa de abstracción entre la base de datos y el código proporcionando un ORM (Object-Relational Mapping) para representar las tablas y relaciones de la base de datos como clases de Python.
2. **Vista:** las vistas en Django son una capa que se encarga de procesar las peticiones de los usuarios y generar una respuesta. En general, son funciones que toman una

solicitud HTTP y devuelven una respuesta HTTP. Normalmente, interactúan con el ORM y generan fichero HTML basándose en una plantilla.

3. **Plantillas:** las plantillas en Django, también llamadas templates, son la capa de presentación de la aplicación web. Los templates son archivos HTML que contienen código que permite insertar dinámicamente datos y elementos de la aplicación.
4. **URL:** los URLs en Django son los responsables de mapear las peticiones de los usuarios a las vistas correspondientes. Estas se definen en un archivo centralizado y son resueltas en función de patrones de expresiones regulares.

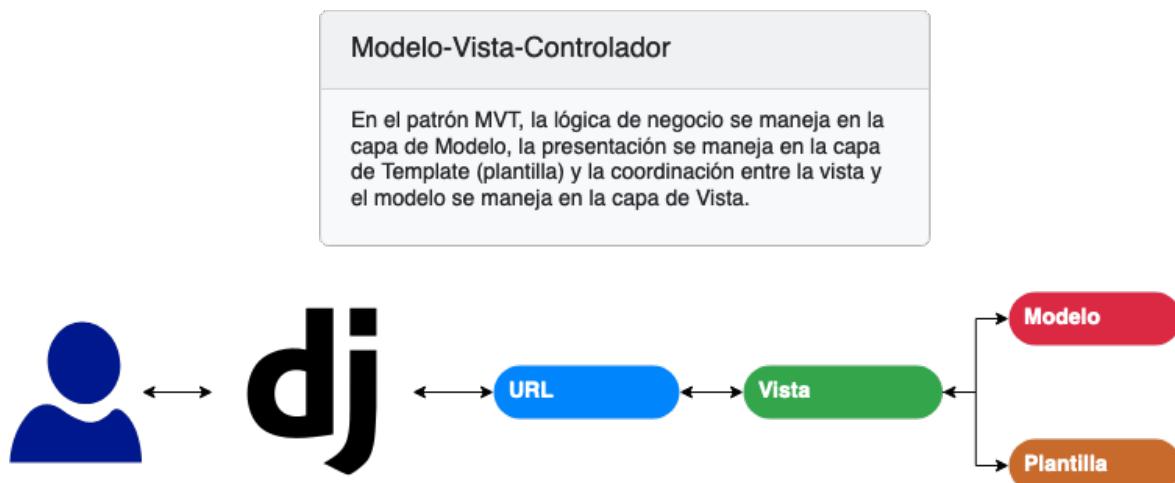


Figura 4.1: Modelo-Vista-Controlador

En nuestra aplicación, Django será el encargado de gestionar toda la lógica de negocio.

4.1.3. JavaScript

JavaScript es un lenguaje de programación interpretado que se utiliza principalmente para el desarrollo de aplicaciones web interactivas y dinámicas, tanto en el lado del cliente como del servidor.

En nuestra aplicación será utilizado mayormente para añadir menús dinámicos en la pantalla de detalles de una cámara:

```
1 const toggleOverlay = document.getElementById("toggleOverlay");
2 const ocultables = document.querySelectorAll(".ocultable");
3
4 toggleOverlay.addEventListener("click", function () {
5     ocultables.forEach(function (ocultable) {
6         if (ocultable.style.display === "none") {
7             ocultable.style.display = "";
```

```

8         toggleOverlay.innerHTML = '<i class="fa-solid fa-eye
9     "></i> Ocultar';
10    } else {
11        ocultable.style.display = "none";
12        toggleOverlay.innerHTML = '<i class="fa-solid fa-eye
13     "></i> Mostrar';
14    }
});
```

Extracto de código 4.1: Menú dinámico JavaScript

En resumen, este código se encarga de manejar los elementos ocultables de un menú. Si los elementos están ocultos, el botón muestra el texto “Mostrar” y oculta los demás elementos del menú. Por otro lado, si los elementos están visibles, el botón muestra el texto “Ocultar” y muestra el resto de elementos del menú. Podemos ver este comportamiento en el capítulo “Manual de usuario” en la Figuras A.1.2 y A.9

4.1.4. Bootstrap

Bootstrap [5] es una herramienta de código abierto para diseño web que ofrece una gran variedad de estilos y recursos predefinidos. Estos recursos incluyen plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de otros frameworks web, Bootstrap se enfoca exclusivamente en el desarrollo front-end, es decir, en la parte visible de la aplicación web.

En el siguiente código podemos ver un ejemplo del uso de las clases de Bootstrap para realizar la barra de navegación de nuestra aplicación:

```

1 ...
2 <div class="collapse navbar-collapse" id="navbarNavDropdown">
3     <ul class="navbar-nav">
4         <li class="nav-item">
5             <a class="nav-link active" aria-current="page" href="/
6 Dashboard"><i class="fa-solid fa-video"></i> Dashboard</a>
7         </li>
8     ...
```

Extracto de código 4.2: Ejemplo Bootstrap en HTML

4.1.5. SQLite

SQLite [23] es una base de datos relacional de código abierto, sin servidor y altamente escalable, que se integra directamente en la aplicación que lo utiliza. La ventaja principal de SQLite es que utiliza un formato de archivo único para almacenar toda la base de datos, lo que lo hace fácil de transportar y compartir entre diferentes sistemas.

En nuestra aplicación, SQLite será gestionada desde el ORM de Django.

4.1.6. JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y fácil de leer y escribir. Es ampliamente utilizado para intercambiar información entre distintos tipos de sistemas.

En nuestra aplicación, utilizaremos el formato JSON para guardar las configuraciones de la aplicación dadas por el usuario:

```
1 {
2   "framesToDetector": 3,
3   "token": "5282233910:AAG_mddkn8zdw_Iip-n1zQX_gSURiBLomC0",
4   "historySize": 40,
5   "historySizeToDetect": 10
6 }
```

Extracto de código 4.3: JSON configuración

4.2. Segundo bloque: Envío de alertas

En esta sección nombraremos las tecnologías utilizadas para permitir el envío de alertas a los usuario. Hemos decidido utilizar el servicio de mensajería Telegram debido a su API, popularidad, seguridad y capacidad de enviar archivos y mensajes de texto.

4.2.1. Telegram API

La API de Telegram es una interfaz de programación de aplicaciones basada en el protocolo MTProto¹ de Telegram, que utiliza una combinación de cifrado simétrico y asimétrico para asegurar la privacidad y seguridad de los datos. La API utiliza una arquitectura cliente-servidor, donde el cliente envía solicitudes a través de HTTP o HTTPS y el servidor responde con los resultados en formato JSON.

La idea principal, es que nuestra aplicación se encargará de enviar peticiones HTTP al servidor API de Telegram, que se encargará de enviar nuestros mensajes a los usuarios indicados.

¹<https://core.telegram.org/mtproto>

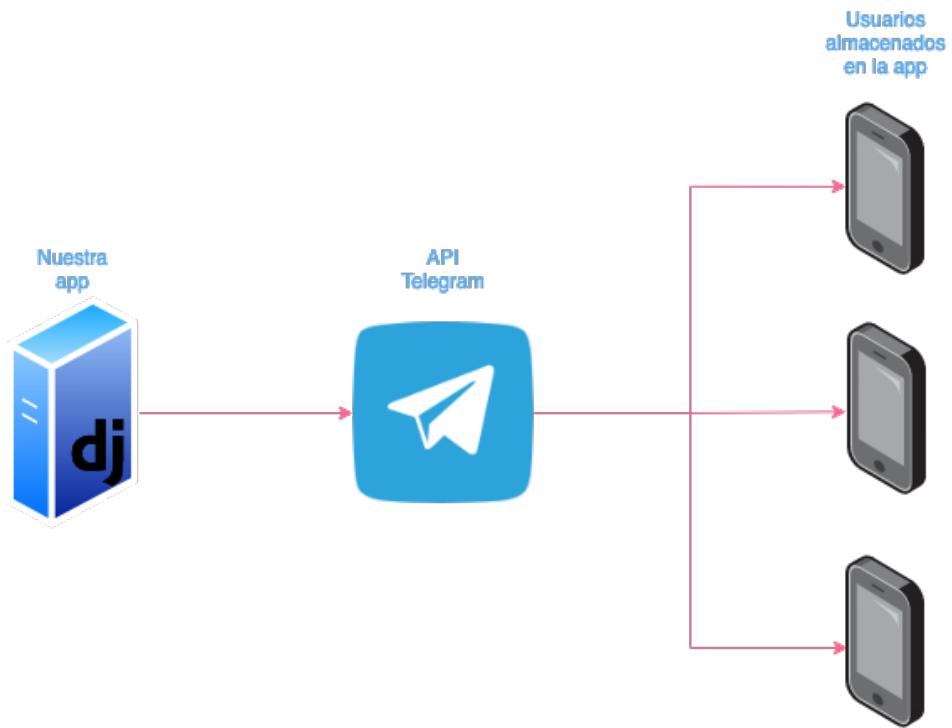


Figura 4.2: Diagrama API Telegram

4.2.2. pyTelegramBotAPI

pyTelegramBotAPI es una librería de Python que facilita el desarrollo de aplicaciones para la API de Telegram. Esta ofrece una gran cantidad de métodos y herramientas para interactuar con la API, permitiendo gestionar múltiples tipos de eventos, como mensajes de texto, vídeo y la interacción con elementos del chat.

Mostraremos la implementación de esta librería en el capítulo 6 Detalles de implementación.

4.3. Tercer bloque: Detección de objetos

En esta sección nombraremos las tecnologías utilizadas para el desarrollo del vídeo en tiempo real y la detección de objetos.

4.3.1. OpenCV

OpenCV [17] (Open Source Computer Vision Library) es una librería orientada al procesamiento y análisis de imágenes. Esta proporciona una amplia variedad de algoritmos de visión por computador predefinidos que pueden ser utilizados para procesar imágenes y vídeos en tiempo real. También incluye herramientas para la captura de vídeo y la lectura, escritura y edición de imágenes en varios formatos.

Durante nuestro proyecto, la utilizaremos para cargar el vídeo en tiempo real de las cámaras añadidas a la aplicación, procesar el vídeo, cambiar formatos y resoluciones (para insertarlo en la red neuronal), añadir recuadros de detecciones y para guardar capturas en los momentos de detecciones.



Figura 4.3: Ejemplo recuadro detección OpenCV

Un ejemplo de uso de OpenCV, como podemos ver en la Figura 4.3.1 anterior, es para el dibujado de los bounding boxes (recuadros de detección) de las detecciones de objetos:

```
1 for i in range(len(self.recent_bboxes)):  
2     bbox = self.recent_bboxes[i]  
3     label = self.recent_labels[i]  
4     cv2.rectangle(frame, (bbox.xmin, bbox.ymin), (bbox.xmax, bbox.ymax),  
5                   (255, 0, 0), 2)  
6     cv2.putText(frame, label, (bbox.xmin, bbox.ymin - 10), cv2.  
7                 FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
```

Extracto de código 4.4: Encuadre de objetos OpenCV

4.3.2. YoloV5

YOLO [25] (You Only Look Once) es un modelo de detección de objetos altamente eficiente y preciso que ha demostrado grandes capacidades para detectar objetos en tiempo real. A lo largo de su evolución desde la primera versión hasta YoloV5 [25], ha experimentado mejoras significativas en precisión, velocidad y eficiencia.

YOLO reutiliza las técnicas utilizadas por otros modelos de su clase, pero además parte de un concepto algo diferente al del resto de modelos, realiza sus predicciones

basadas en la imagen completa en lugar de trabajar por regiones de interés. Durante su entrenamiento, divide las imágenes en una cuadrícula y se encarga de aprender las probabilidades de encontrar los objetos en las distintas celdas de ésta.

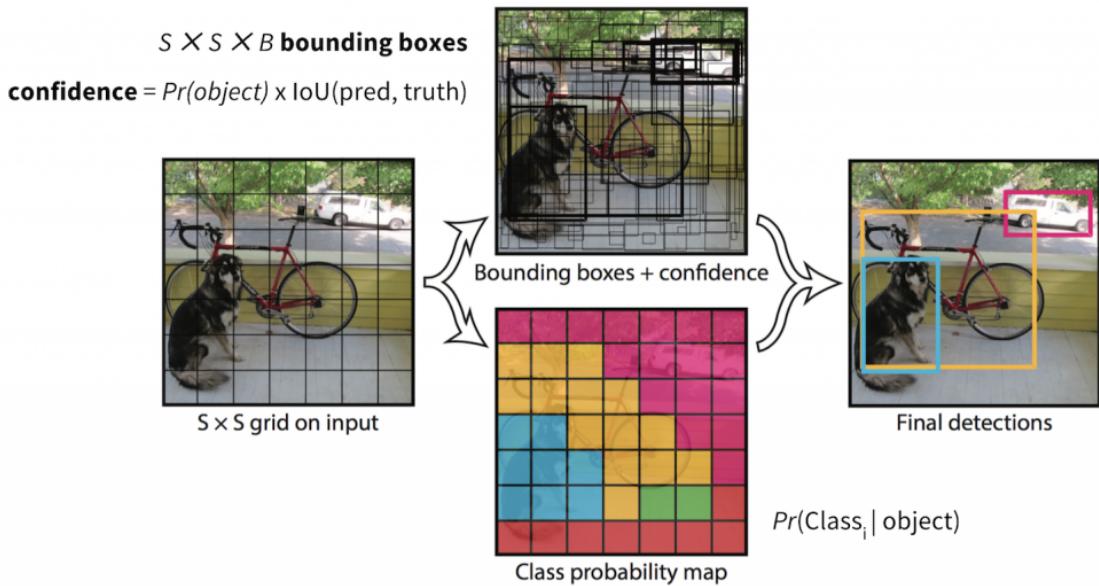


Figura 4.4: Funcionamiento YoloV5

Conceptos básicos: Detectores de objetos CNN

Las arquitecturas de estos modelos suelen estar compuestas por 3 componentes diferenciados:

- **Backbone:** es una red pre-entrenada que se utiliza para extraer características representativas de las imágenes. Es responsable de identificar bordes, formas y texturas en la imagen que pueden ser útiles para detectar objetos. Puede llegar a ser determinante para la eficiencia y rendimiento del modelo.
- **Neck:** son una serie de capas que mezclan y combinan las características de la imagen, con la intención de mejorar la generalización de distintos objetos a distintas escalas.
- **Head:** es la última parte de la arquitectura de la red neuronal, esta se utiliza para realizar la detección de objetos y la clasificación de imágenes. Dependiendo del tipo de detector, estará formada por una o dos redes (detectores en uno o dos pasos)

Arquitectura de YoloV5

La arquitectura de YoloV5 se basa en CSP-Darknet53 implementado en Pytorch (Backbone), a diferencia de las versiones anteriores que utilizan Darknet53 implementada en Caffe. YoloV5 es un detector de una etapa (one-stage), lo que significa que utiliza

una única red neuronal para realizar la detección de objetos y generar propuestas de regiones simultáneamente.

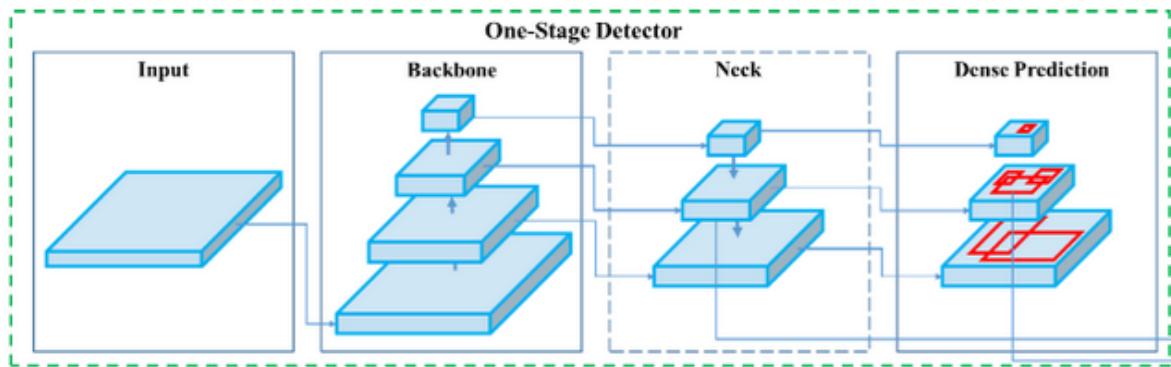


Figura 4.5: Arquitectura YoloV5

YoloV5 está disponible en varios tamaños. Para este proyecto, se eligió YoloV5s, debido a que presenta un menor tiempo de inferencia, lo que beneficia el procesamiento de imágenes en tiempo real y ocupa menos espacio en disco.

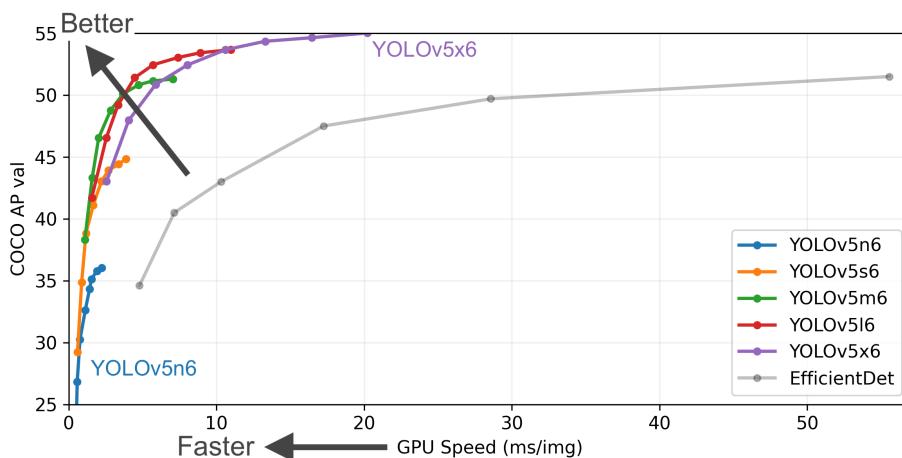


Figura 4.6: Comparación YoloV5

En cuanto a las funciones de activación, YoloV5 utiliza Leaky ReLU en todas las capas de la red neuronal, excepto en la última capa, donde se emplea la función Sigmoid. Estas funciones de activación ayudan a evitar la desaparición del gradiente durante el entrenamiento, mejoran la precisión de la red y generan los valores de confianza.

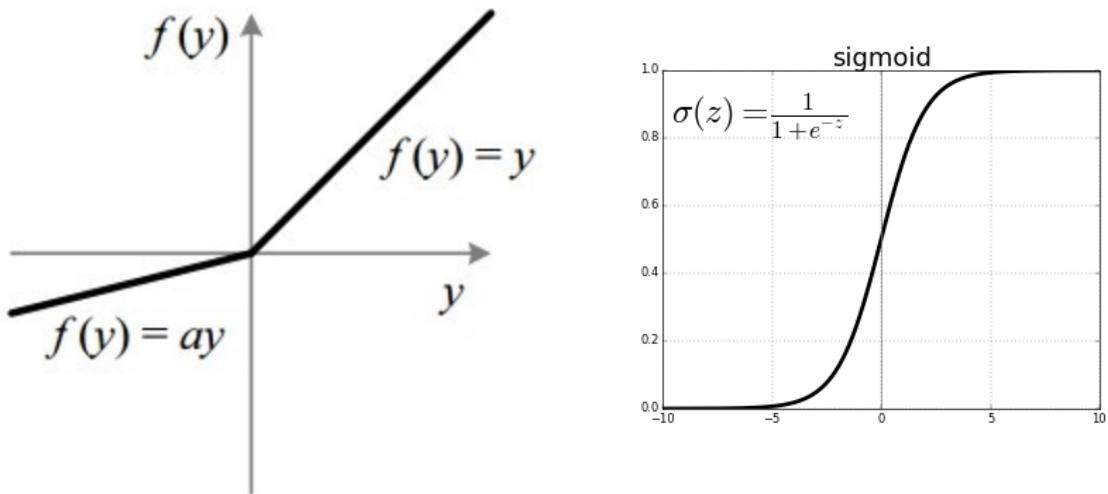


Figura 4.8: Sigmoid

Figura 4.7: Leaky ReLU

4.3.3. IceVision

IceVision [2] es un framework de código abierto para el desarrollo de algoritmos de Visión por Computador. Su enfoque está en la creación de modelos para la detección de objetos, segmentación de imágenes y clasificación de imágenes y se integra con otras bibliotecas populares de Python como PyTorch, FastAI y OpenCV. Ofrece varios modelos pre-entrenados con diferentes backbones para la extracción de características. Además, permite cargar y exportar modelos, así como realizar inferencia en ellos.

En nuestro proyecto, IceVision será el núcleo de inferencia para el vídeo en tiempo real de las cámaras. Los modelos subidos por los usuarios a LiveGuardian tendrán que ser entrenados con IceVision o compatibles con este.

5. Diseño del sistema

En este capítulo se presenta la arquitectura del sistema, se proporciona una descripción detallada de las tres partes mencionadas previamente y se explica cómo están interconectadas.

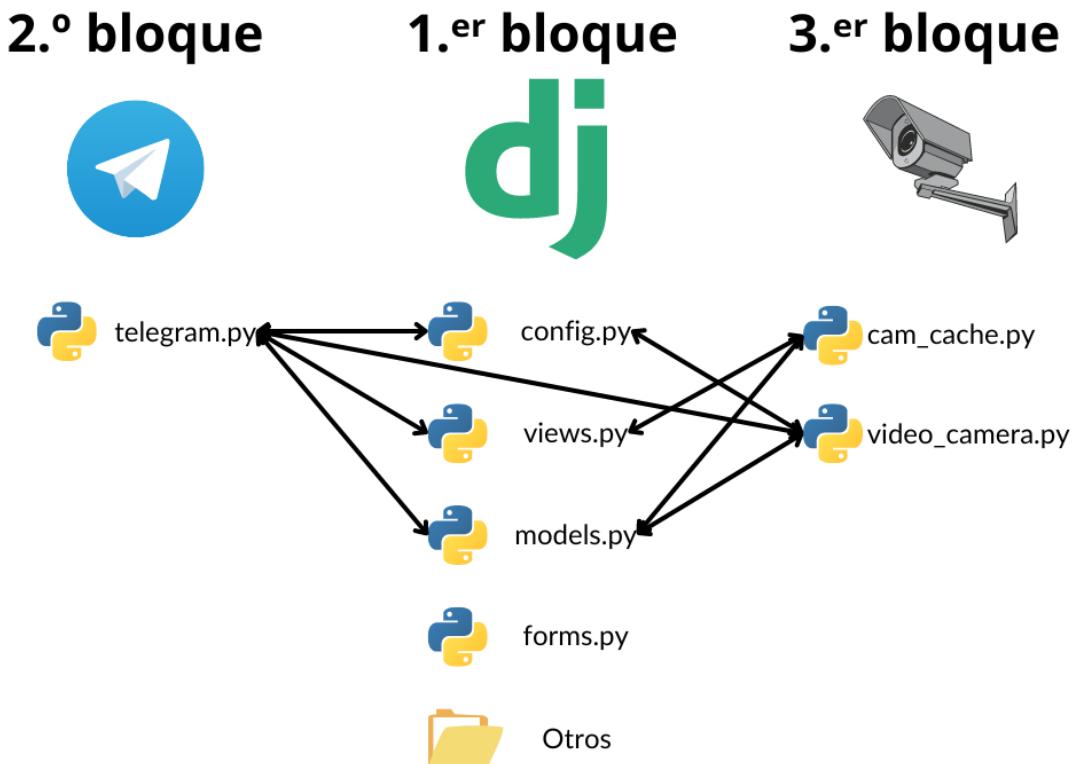


Figura 5.1: Conexiones entre bloques

En la siguiente Figura 5 se muestra la estructura de archivos del proyecto. Esta está fuertemente relacionada con la del framework Django, ya que es una de las tecnologías principales del proyecto. En las siguientes secciones se hablará de cada uno de estos archivos y su contenido.

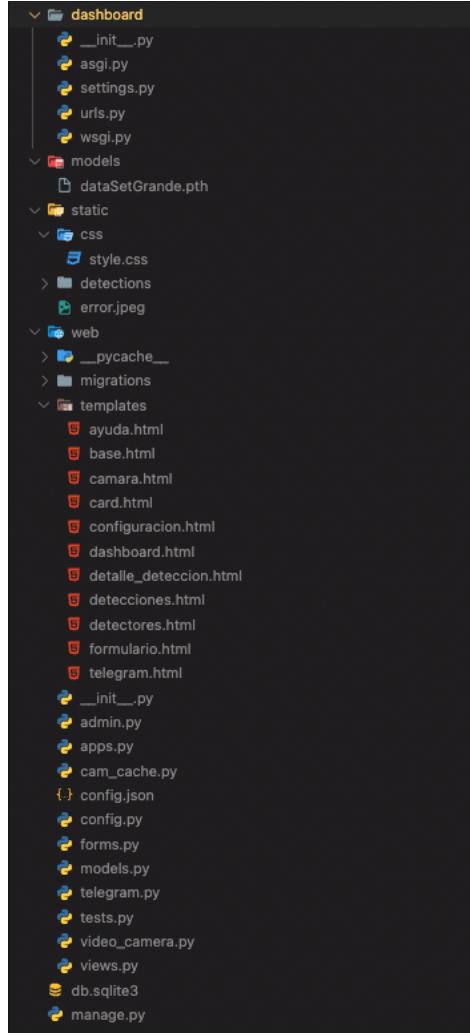


Figura 5.2: Árbol de ficheros del proyecto

5.1. Primer bloque: Gestión web

En esta sección hablaremos de los archivos destacados para la gestión web. Explicaremos de forma general los distintos archivos y directorios de Django.

- **dashboard:** contiene los archivos principales de configuración del proyecto relacionados con Django.
 - **settings.py:** en este archivo se gestiona toda la configuración relacionada con Django, incluyendo el idioma, los hosts permitidos, las aplicaciones creadas en el proyecto y la ruta de los archivos estáticos.
 - **urls.py:** este archivo contiene las URL de la aplicación y cómo se gestionan. Desde este punto, se establecen las relaciones entre las URLs que tendrá la aplicación y las vistas que manejarán las peticiones para dichas URLs.
- **models:** directorio donde se almacenarán los modelos subidos por los usuarios. En general en formato .pth.

- **static:** ruta de la aplicación donde se almacenan ficheros estáticos usados por la interfaz de la aplicación, incluyendo imágenes, ficheros CSS y código JavaScript. Además, en esta ruta nos encontramos con la carpeta detections, el lugar donde se almacenan las imágenes relacionadas con las detecciones de objetos.
- **web:** en este directorio se encuentran todos los archivos que modelarán la lógica de nuestra aplicación. Los explicamos más detalladamente a continuación.
- **manage.py:** archivo principal utilizado para realizar operaciones sobre el proyecto, como crear usuarios, borrar la base de datos e iniciar el servidor.
- **db.sqlite3:** archivo local donde se almacenan los datos de SQLite.

Procedemos a explicar los archivos del directorio web que nos interesan para este primer bloque:

admin.py

En este archivo se definen los modelos que podrán ser vistos desde el panel de administración de Django. En este panel los usuarios administradores pueden crear, leer, actualizar y eliminar registros de la base de datos sin escribir código. Para este proyecto se ha decidido hacer visibles todos los modelos para facilitar la gestión de los mismos.

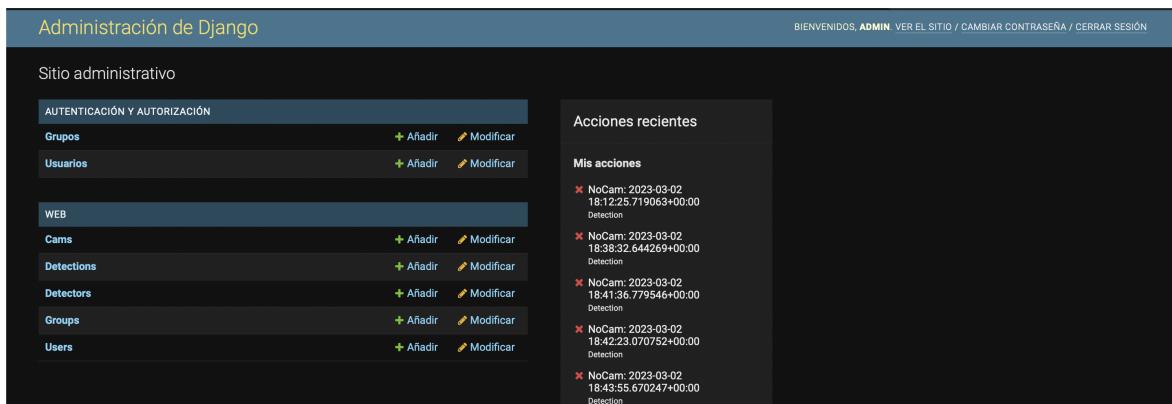


Figura 5.3: Panel de administración Django

video_camera.py

La clase **VideoCamera** implementa funcionalidad para los registros de cámaras en la base de datos. Esta clase se encarga de procesar el vídeo en tiempo real en un hilo e implementar detección de objetos en caso de ser necesaria. En el capítulo 5 Detalles de implementación, se abordará en profundidad esta clase y su funcionamiento.



Figura 5.4: VideoCamera

cam_cache.py

CamCache es responsable de la gestión de todos los objetos **VideoCamera** que procesan vídeo en tiempo real. El método “add” se encarga de crear una cámara utilizando registros de la base de datos. Por otro lado, el método “delete” se encarga de detener el hilo de la cámara, lo que resulta en la interrupción del procesamiento de dicha cámara en tiempo real.

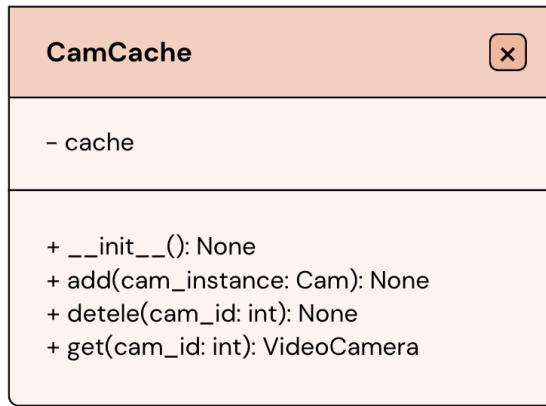


Figura 5.5: CamCache

forms.py

El archivo forms.py se utiliza para definir formularios con **ModelForm** en Django. Estos formularios se generan automáticamente a partir de modelos definidos en Django, lo que evita la necesidad de escribir código repetitivo. Además, proporcionan validación directa del formulario de manera conveniente.

```

1 class CrearCamara(ModelForm):
2     class Meta:
3         model = Cam
4         fields = '__all__'
5         labels = {
6             'name': 'Nombre de la camara',
7             'url': 'Url de la camara',
8             'detector': 'Seleccione un detector',
9             'groups': 'Seleccionar grupos'
10        }
11        widgets = {
12            'name': forms.TextInput(attrs={'size': '10', 'class': 'form-
control'}),
13            'url': forms.TextInput(attrs={'size': '40', 'class': 'form-
control'}),
14            'detector': forms.Select(attrs={'class': 'form-control'}),
15            'groups': forms.SelectMultiple(attrs={'class': 'form-control'
})
16        }

```

Extracto de código 5.1: Formulario Django

config.py

AppConfig se encarga de gestionar la configuración utilizada en las cámaras. Desde esta clase se puede modificar y obtener la configuración desde cualquier parte

de la aplicación.

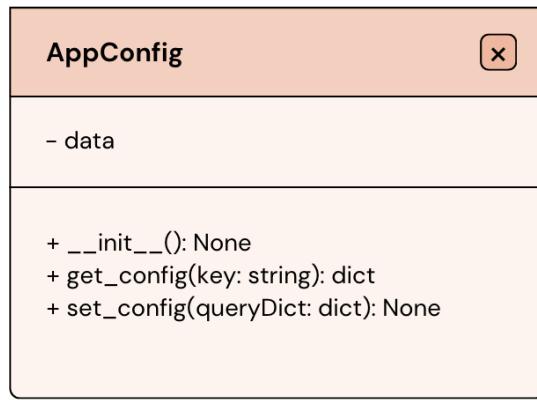


Figura 5.6: AppConfig

models.py

En este fichero se definen los modelos Django que constituirán las tablas finales en la base de datos. Los modelos siguen la siguiente estructura:

```
1 class Detection(models.Model):
2     cam = models.CharField(max_length=100, null=True, blank=True)
3     date = models.DateTimeField(null=False, blank=False)
4     img = models.ImageField(upload_to='static/detections')
5     items = models.CharField(max_length=100, null=True, blank=True)
6     pred = models.CharField(max_length=100, null=True, blank=True)
7     detector = models.CharField(max_length=100, null=True, blank=True)
8
9     def __str__(self) -> str:
10         if self.cam != None:
11             return str(self.cam.name) + ":" + str(self.date)
12         else:
13             return "NoCam:" + str(self.date)
```

Extracto de código 5.2: Modelo Django

Finalmente, nuestra base de datos SQLite acaba modelada con la siguiente estructura:

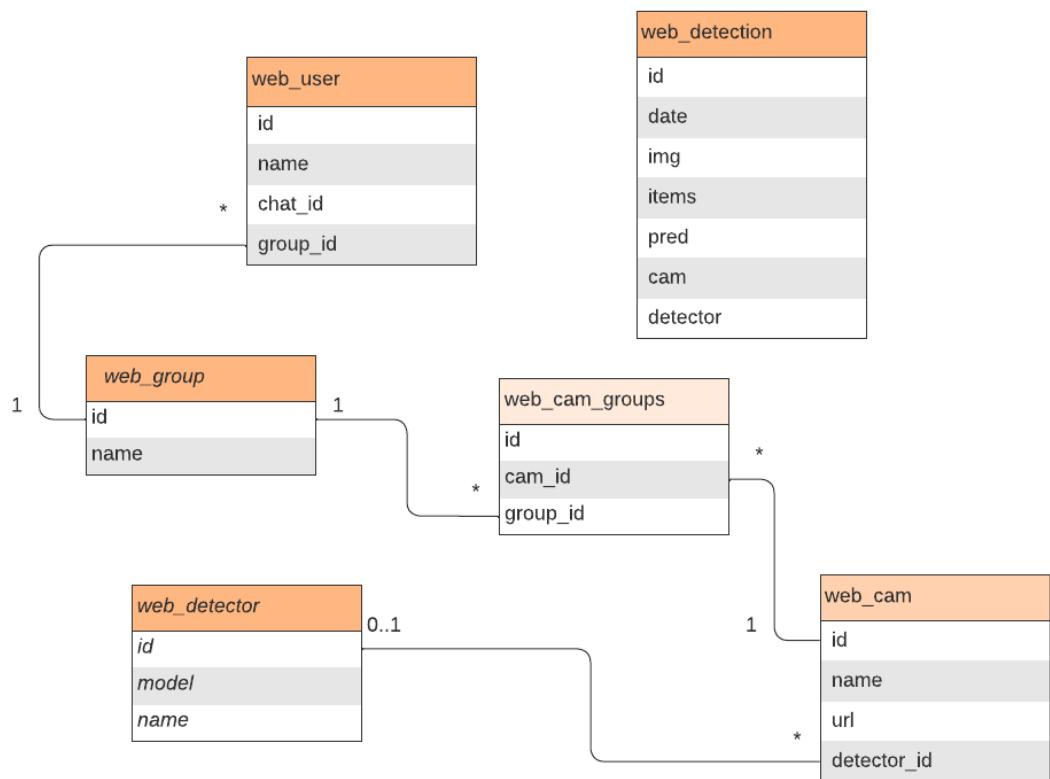


Figura 5.7: ERD DB

views.py

En este fichero se definen las funciones o clases de vista que son llamadas cuando se accede a una URL específica. Estas funciones o clases se encargan de interactuar con la base de datos, procesar formularios y renderizar plantillas HTML o devolver respuestas en otros formatos, como JSON.



Figura 5.8: Views

En nuestra aplicación, además de tener una vista para cada pantalla y para eliminar registros de la base de datos, tendremos una vista para generar vídeo en streaming dado un id de una cámara.

```

1 def video(request, id_cam):
2     CAMERA_INPUT = cam_cache.get(id_cam)
3     try:
4         return StreamingHttpResponse(gen(CAMERA_INPUT), content_type=""
5             multipart/x-mixed-replace;boundary=frame")
6     except:
7         render(request, 'dashboard.html')

```

Extracto de código 5.3: Vista para generar video en tiempo real

5.2. Segundo bloque: Envío de alertas

En esta sección hablaremos de la interacción del envío de alertas de Telegram con el resto de la aplicación.

Desarrollar la comunicación con Telegram no ha sido una tarea complicada. Telegram proporciona una documentación ¹ completa y sencilla con muchos ejemplos de código que facilitan la comprensión, además se encuentra respaldada por una amplia comunidad de desarrolladores que proporcionan muchas herramientas que facilitan muchas labores de desarrollo.

Gracias a esto, podemos introducir todo nuestro código relacionado con la conexión con Telegram en un solo archivo. Dentro del archivo **telegram.py**, encontramos la clase **Telegram**. Esta clase se utiliza para enviar mensajes a los usuarios relacionados con una instancia de VideoCamera, que se pasa como parámetro en la función `send_detection`.

Al crear una instancia de **Telegram**, se inicializa un bot de Telegram con el token almacenado en la configuración de la aplicación. Sin embargo, es posible cambiar el token del bot en cualquier momento utilizando la función `set_bot`. Además, cuando se cambia la configuración de la aplicación, el bot se actualiza automáticamente con el nuevo token.

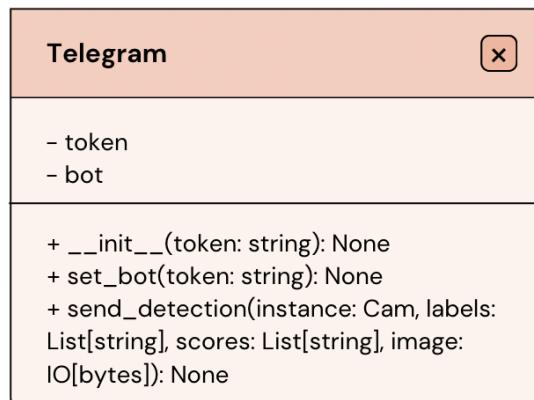


Figura 5.9: Telegram

5.3. Tercer bloque: Detección de objetos

En esta sección hablaremos del entrenamiento del detector de objetos usado en la aplicación. La implementación del mismo se explicará más adelante en el capítulo 6 Detalles de implementación.

El entrenamiento se ha realizado en el entorno Jupyter. El notebook utilizado realiza el entrenamiento utilizando IceVision y la arquitectura YOLOv5.

Durante el proceso de entrenamiento, se han empleado dos conjuntos de datos:

¹<https://core.telegram.org/bots/api>

El primero está formado por imágenes con armas etiquetadas, este ha sido extraído del artículo *A Dataset and System for Real-Time Gun Detection in Surveillance Video Using Deep Learning* [21] publicado en 2021. El dataset ² ya se encuentra en formato Pascal VOC (en XML), en ficheros .tar:

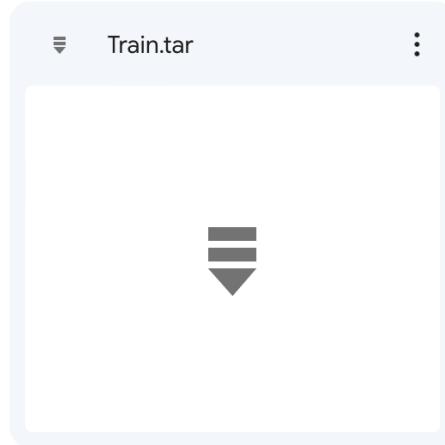


Figura 5.10: Train dataset

Nuestro segundo conjunto de datos está formado con imágenes sin presencia de armas (conjunto negativo) mayormente de imágenes de cámaras de seguridad en entornos cotidianos. Esta estrategia garantiza que el sistema sea más robusto y evite etiquetar erróneamente objetos comunes como armas, como por ejemplo teléfonos móviles o carteras sostenidas en las manos. Estas imágenes las extraemos de dos fuentes, el dataset COCO (Common objects in context) que no contiene armas y utilizaremos sin sus etiquetas:

COCO Explorer

COCO 2017 train/val browser (123,287 images, 886,284 instances). Crowd labels not shown.

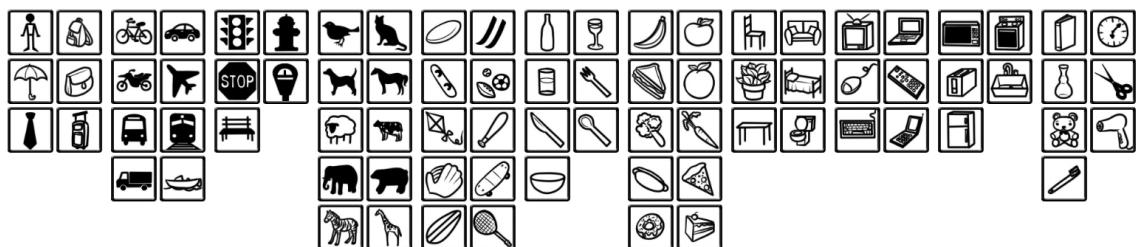


Figura 5.11: Objetos en COCO

Y otros 2 pequeños datasets con ejemplos negativos brindados también por el artículo anteriormente mencionado:

²https://drive.google.com/drive/folders/179q_MNjx0ipzybhdjpQTxVu3IbI-5lWl

Archivos

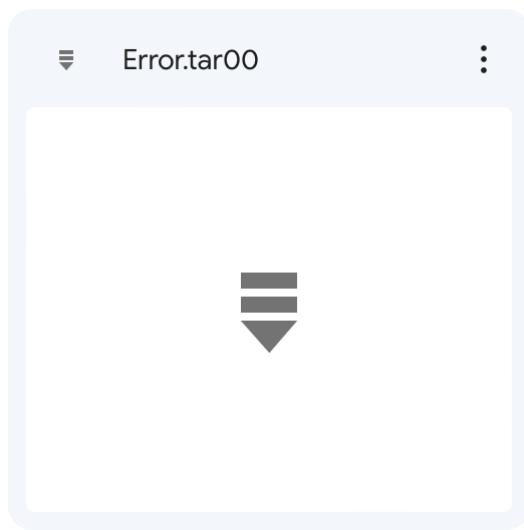


Figura 5.12: Dataset Error01

Archivos

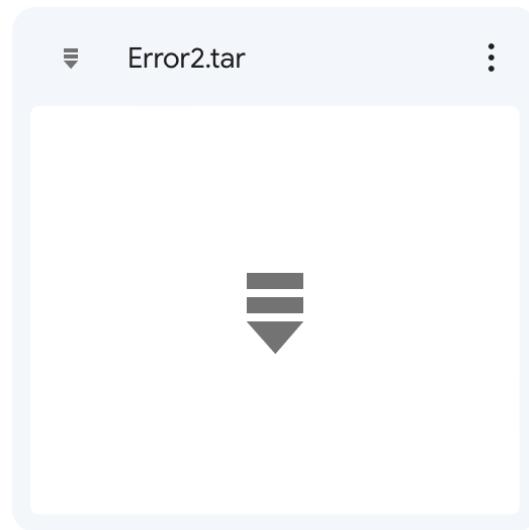


Figura 5.13: Dataset Error02

Los dos conjuntos de datos han sido divididos en 80 % imágenes para entrenamiento y un 20 % imágenes para validación.

```
In [ ]: # Default 80% 20%
train_records, valid_records = parser.parse()
```

Figura 5.14: Split conjunto de datos

```
In [41]: %matplotlib inline
show_records(random.choices(train_records, k=10), ncols=5)
```



Figura 5.15: Imágenes de armas etiquetadas

Para poder cargar las imágenes del conjunto negativo, es necesario crear un parser personalizado. En este parser no es necesario realizar ninguna transformación con las

etiquetas de los objetos, ya que como hemos mencionado antes, estas imágenes con contienen etiquetas:

```
In [44]: ### No necesitamos los métodos para detectar, por lo que no añademos bboxes
class NegativeImageParser(Parser):
    def __init__(self, template_record, data_dir):
        super().__init__(template_record=template_record)
        self.image_filepaths = get_image_files(data_dir) ### Para que el parser obtenga las imágenes

    def __iter__(self) -> Any:
        yield from self.image_filepaths

    def __len__(self) -> int:
        return len(self.image_filepaths)

    def record_id(self, o) -> Hashable:
        return o.stem

    def parse_fields(self, o, record, is_new):
        if is_new:
            record.set_img_size(get_img_size(o))
            record.set_filepath(o)
```

Figura 5.16: Parser personalizado

Una vez tenemos nuestro Parser definido, podemos cargar el conjunto de datos y añadirlo al conjunto de entrenamiento y validación:

```
In [45]: #Definimos los path de los negativos
negative_1_path = Path('./Descargas/Error/JPEGImages')
negative_2_path = Path('./Descargas/Error2')

In [46]: negative_1_parser = NegativeImageParser(negative_template_record, negative_1_path)
negative_2_parser = NegativeImageParser(negative_template_record, negative_2_path)

In [ ]: train_negative1_records, valid_negative1_records = negative_1_parser.parse()

In [ ]: train_negative2_records, valid_negative2_records = negative_2_parser.parse()

In [49]: train_negative_records = train_negative1_records + train_negative2_records

In [50]: valid_negative_records = valid_negative1_records + valid_negative2_records
```

Figura 5.17: Split conjunto de datos negativo

```
In [52]: %matplotlib inline
show_records(random.choices(valid_negative_records, k=10), ncols=5)
```

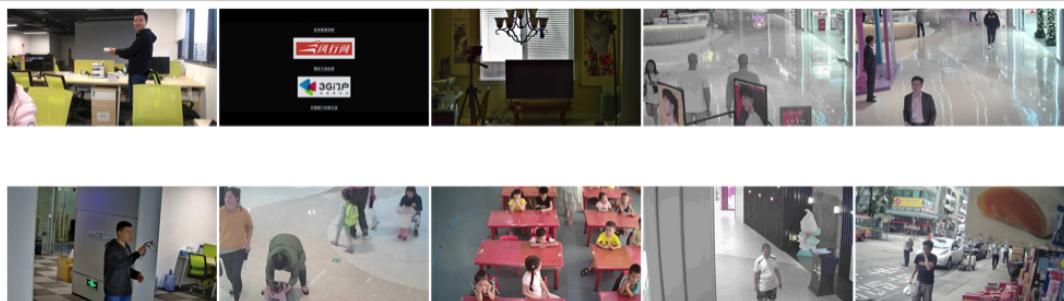


Figura 5.18: Imágenes negativas

Finalmente, entrenamos nuestro modelo con un Learning Rate del 0.003 y obtenemos un valid_loss del 0.038242 utilizando fine tuning.

SuggestedLRs(valley=0.0030199517495930195)

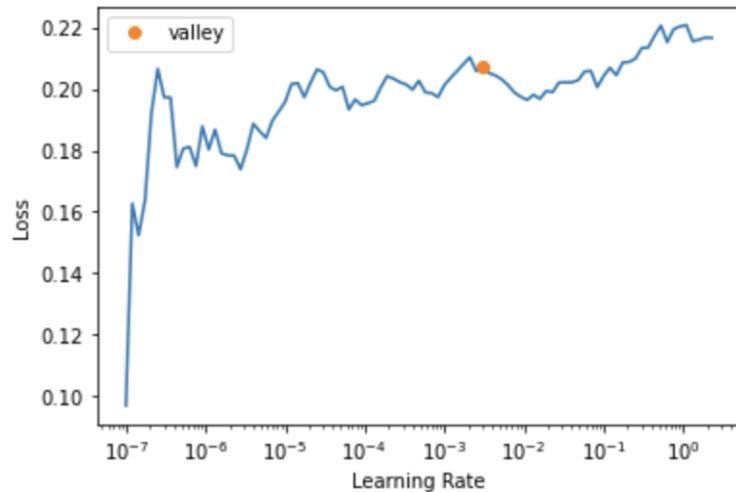


Figura 5.19: Learning Rate

Una vez entrenado nuestro modelo, procedemos a guardarla en un archivo .pth para poder usarlo en nuestra aplicación:

```
In [67]: checkpoint_path = 'dataSetGrande.pth'
save_icevision_checkpoint(model,
    model_name='ultralytics.yolov5',
    backbone_name='small',
    classes = parser.class_map.get_classes(),
    img_size=image_size,
    filename=checkpoint_path,
    meta={'icevision_version': '0.12.0'})
```

Figura 5.20: Guardar modelo entrenado

6. Detalles de implementación

En este capítulo hablaremos sobre los detalles fundamentales y complejos de cada una de las 3 partes diferenciadas de la aplicación.

6.1. Primer bloque: Gestión web

En este bloque tenemos varias implementaciones destacadas, mayormente relacionadas con la integración del vídeo en tiempo real, los detectores de objetos y modelos.

6.1.1. Extracción del vídeo en tiempo real

La extracción de vídeo en tiempo real de las diferentes cámaras se realiza en hilos independientes. Cada vez que se crea una cámara en la base de datos, se instancia un objeto **VideoCamera**, que inicializa un hilo encargado de extraer los fotogramas de la fuente y asignarlos como un atributo de la clase.

```
1 class VideoCamera(object):
2     def __init__(self, instance):
3         ...
4         self.thread = threading.Thread(target=self.update, args=())
5         self.thread.start()
6         ...
```

Extracto de código 6.1: Creación hilo VideoCamara

El hilo tiene como objetivo ejecutar la función **update()**, la cual se encarga de determinar cómo se procesarán los fotogramas, ya sea con detección de objetos o sin ella. Si se inicializa una cámara sin detección de objetos, se extraen los fotogramas y se asignan al atributo `self.frame`. En caso de no obtener correctamente el fotograma debido a errores relacionados con la cámara, se muestra un mensaje de error junto junto al tiempo transcurrido en segundos desde que comenzó a fallar.

```
1 ### Se encarga de extraer los fotogramas sin detector de objetos.
2 def no_detection(self):
3     while self.live:
4         (self.grabbed, frame) = self.video.read()
5         if self.grabbed:
6             self.retry = 0
7             self.frame = frame
8         else:
9             self.retry += 1
10            self.set_error_frame()
```

```
11     self.video = cv2.VideoCapture(self.instance.url)
12     time.sleep(1)
```

Extracto de código 6.2: Extracción de fotograma sin detección.

Gracias a esta implementación, cuando un usuario desea ver una cámara, Django puede obtener de forma asíncrona las imágenes de este atributo. Podemos entenderlo como un problema de productor-consumidor, donde el productor es el objeto VideoCamera y el consumidor es la vista de Django que se encarga de generar el streaming al usuario.

6.1.2. Gestión de instancias VideoCamera

Como mencionamos en la sección anterior, obtener imágenes de una cámara es tan sencillo como acceder a su atributo self.frame. Sin embargo ¿cómo se gestionan estos objetos?

Para gestionar estos objetos se ha creado una cache de cámaras. Esta cache se encarga de replicar las operaciones realizadas a la base de datos a las cámaras activas. Además, al proporcionar un id, nos permite obtener la instancia de **VideoCamera**, lo que nos facilita acceder a los fotogramas producidos por una cámara desde cualquier parte de la aplicación. Un ejemplo claro de su uso es cuando necesitamos generar streaming para una cámara. Dado un id, obtenemos la instancia de la cámara para obtener los fotogramas y generar un streaming:

```
1 def video(request, id_cam):
2     CAMERA_INPUT = cam_cache.get(id_cam)
3     try:
4         return StreamingHttpResponse(gen(CAMERA_INPUT), content_type=""
5             multipart/x-mixed-replace;boundary=frame")
6     except:
7         render(request, 'dashboard.html')
8
9 def gen(camera):
10     while True:
11         frame = camera.get_frame()
12         yield (b"--frame\r\n"
13             b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
```

Extracto de código 6.3: Generar streaming.

Por último, cabe destacar que cuando una cámara es utilizada, se libera el hilo y la instancia VideoCamara asignada es eliminada, liberando de procesamiento a la aplicación.

6.1.3. Gestión de modelos y detecciones

Cada vez que un usuario sube un modelo o se genera una detección, tenemos un archivo asociado a un registro en la base de datos. En caso de un modelo un

archivo .pth que contiene los pesos de la red. En caso de una detección, una imagen del momento de la detección. Nuestro objetivo principal es asegurarnos de que cuando se elimine un registro de la base de datos que tenga un archivo asociado, también se elimine dicho archivo. Para lograr esto, utilizamos el decorador `@receiver` de Django, el cual se encarga de eliminar automáticamente el archivo cuando se recibe un evento de eliminación:

```
1 @receiver(models.signals.post_delete, sender=Detection)
2 def auto_delete_file_on_delete(sender, instance, **kwargs):
3     try:
4         if instance.img:
5             if os.path.isfile(instance.img.path):
6                 os.remove(instance.img.path)
7                 print("Archivo " + str(instance.img.path) + " eliminado")
8             else:
9                 print("Archivo " + str(instance.img.path) + " no existe")
10    except:
11        print("Archivo " + str(instance.img.path) + " no ha podido ser
eliminado")
```

Extracto de código 6.4: Receiver para la eliminación de detecciones.

6.2. Segundo bloque: Envío de alertas

En esta segunda parte, abordaremos la creación y administración del bot de Telegram.

6.2.1. Bot de Telegram

Para poder enviar alertas a nuestros usuarios, necesitamos un bot de Telegram con el que poder comunicarnos desde la API. Para crear un bot es necesario tener cuenta en Telegram, ya que necesitaremos comunicarnos con el bot “BotFather”.

Para crear un bot, mandamos el comando /newbot, le asignamos un nombre y un usuario. De esta forma, BotFather nos crea nuestro bot y nos brinda su token:

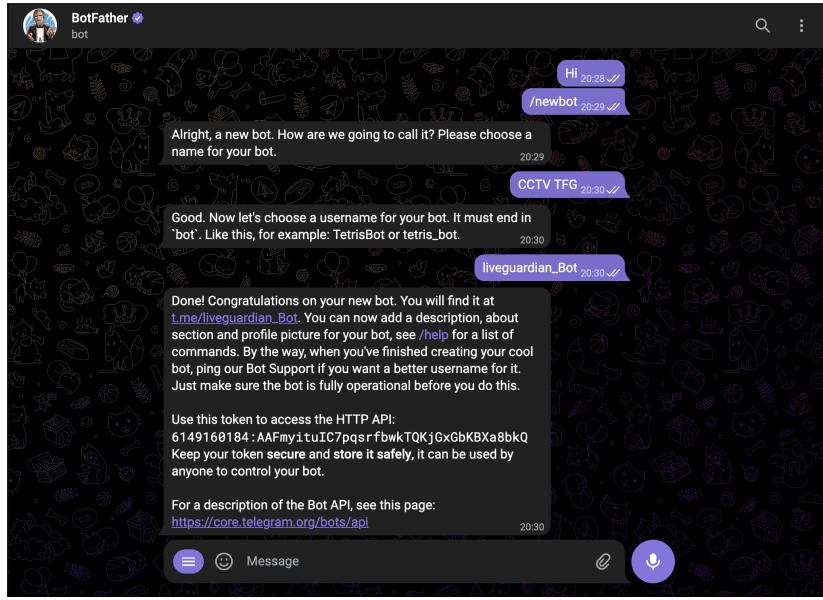


Figura 6.1: Crear bot de Telegram

Una vez creado, finalizamos añadiéndole una foto de perfil. Usamos el comando /mybots y cliclamos en Edit Botpic:

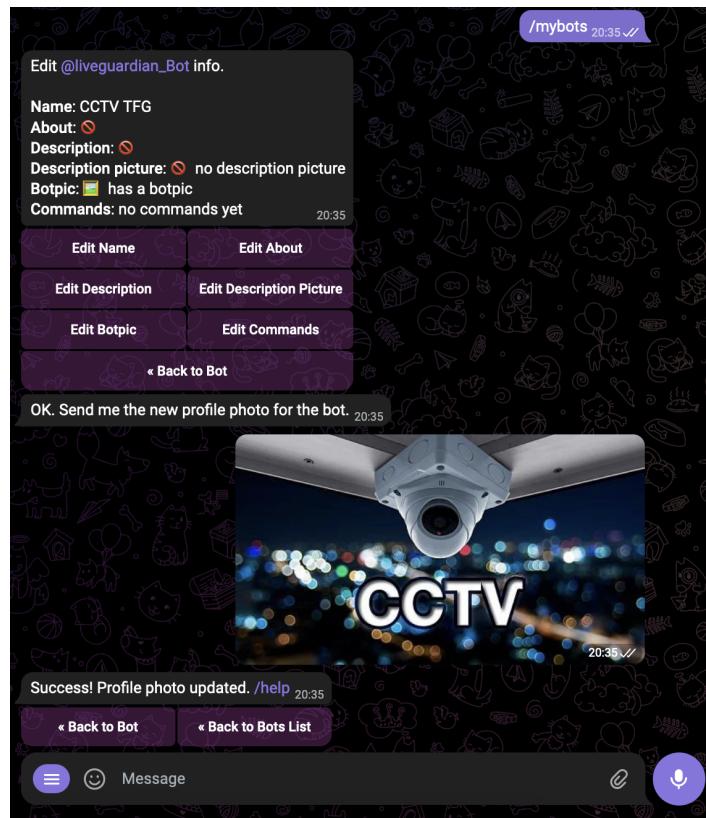


Figura 6.2: Editar foto de perfil

6.3. Tercer bloque: Detección de objetos

En este tercer apartado, hablaremos sobre la integración de Icevision con Django, adaptaciones a la arquitectura de inferencia y el concepto de “pacienza” en detecciones.

6.3.1. Integración de Icevision con Django

Para utilizar los modelos de Icevision para la inferencia, aprovechamos el sistema de instancias de VideoCamera mencionado previamente. Cuando se crea una instancia desde la base de datos y se le asigna un detector de objetos, cargamos el modelo junto con sus metadatos antes de comenzar a generar los fotogramas:

```
1 def detection(self):
2     checkpoint_and_model = model_from_checkpoint(self.instance.detector.
3         model.path)
4     model_type = checkpoint_and_model["model_type"]
5     class_map = checkpoint_and_model["class_map"]
6     img_size = checkpoint_and_model["img_size"]
7     model = checkpoint_and_model["model"]
8
9     valid_transforms = tfms.A.Adapter([*tfms.A.resize_and_pad(img_size),
10                                         tfms.A.Normalize()])
11
12     while self.live:
13         (self.grabbed, frame) = self.video.read()
14         ...
```

Extracto de código 6.5: Carga del modelo Icevision.

Una vez se encuentra el modelo cargado, podemos empezar: entramos en el bucle while, donde generamos fotogramas mientras la cámara esté activa.

6.3.2. Adaptaciones a la arquitectura de inferencia

Inicialmente, procesábamos todos los frames provenientes de las cámaras a una velocidad de 30 frames por segundo (fps). Sin embargo, al añadir una segunda cámara, el detector introducía un retardo que podía incluso llegar a paralizar la visualización de las cámaras con dicho detector. Para resolver este problema, se propuso reducir la cantidad de fotogramas que el detector procesa por cámara para saturar menos nuestra GPU y el I/O con la web. Durante el desarrollo, se implementaron optimizaciones para permitir el uso de un mayor número de cámaras con detectores, sin importar el hardware utilizado por el host de la aplicación. Cuando se emplea un modelo en una cámara, no todos los fotogramas pasan a través de la arquitectura de inferencia. Esta estrategia libera los recursos del sistema y facilita la escalabilidad horizontal al utilizar modelos en varias cámaras simultáneamente.

El intervalo de fotogramas para pasar por inferencia lo puede definir el usuario desde la configuración. Cuanto mayor sea este intervalo, menos recursos serán usados

para la detección de objetos por cámara y peor puede llegar a ser la detección de objetos en éstas.

Cuando se detecta un objeto en un fotograma, el recuadro de detección se mantiene en las mismas coordenadas dentro de la imagen hasta que se procese el siguiente fotograma en inferencia. Esta optimización es imperceptible para el usuario, ya que al trabajar con cámaras que capturan a una tasa de 15-30 fotogramas por segundo, por ejemplo, en un intervalo de 4 fotogramas el objeto detectado se habrá movido escasos centímetros. Gracias a esto se libera una cantidad considerable de recursos del sistema no afectando a la detección de objetos.

6.3.3. Paciencia en las detecciones

Inicialmente, teníamos un enfoque en el que se almacenaba una detección si se detectaba un objeto en la cámara, y esperábamos a que el objeto saliera del campo de visión antes de poder realizar una nueva detección. Esto se hacía para evitar enviar múltiples alertas sobre el mismo objeto. Sin embargo, este enfoque resultaba vulnerable a falsos positivos. Cualquier detección que ocurriera durante un solo fotograma, como un teléfono móvil pasando rápidamente, podía provocar una detección y, por ende, el envío de una alerta. Para evitar esto, se desarrolló un sistema que espera y evita estos falsos positivos.

Se ha implementado un historial de fotogramas previos. Este historial almacena el estado de los n fotogramas anteriores, donde los posibles estados son 0, indicando que no se ha detectado nada, y 1, indicando que se ha detectado un objeto. El tamaño del historial puede ser ajustado por el usuario a través de la configuración del sistema. Este historial nos permite tener en cuenta los fotogramas anteriores, con una ligera vista al pasado podemos verificar si el objeto que detectamos en el fotograma actual es reincidente o un posible falso positivo.

¿Cuántos fotogramas anteriores tenemos en cuenta para detectar un objeto? A este parámetro lo llamaremos “umbral” y será ajustable desde la configuración de la aplicación. Con él, conseguimos reducir los posibles falsos positivos, ya que establece cuantas cuantas veces tenemos que detectar un objeto en el historial para así enviar una alerta y almacenar la detección.

En acción, la combinación de nuestro historial y umbral forma una ventana deslizante. En tiempo real, los estados entran por la derecha y salen por la izquierda. Cada vez que un nuevo elemento ingresa en el historial, si este es positivo y la suma del historial supera el umbral, se almacena y envía una detección.



Figura 6.3: Sistema de paciencia. Umbral: 6. Historial: 12

Cuanto menor sea el umbral, nuestras cámaras serán más sensibles a las detecciones. Por ejemplo, si tenemos una cámara con una tasa de 30 fps y establecemos un umbral de 15, cualquier objeto que aparezca en pantalla durante medio segundo será detectado. Respecto al tamaño del historial, influye a la ventana de tiempo sobre la que queremos acumular fotogramas para una posible detección, cuanto mayor el tamaño, más tiempo tendremos en cuenta posibles fotogramas con objetos.

Símil

Podemos visualizar de forma sencilla este sistema de paciencia como un guardia de seguridad. Si nuestro guardia de seguridad observa algo sospechoso durante su turno, no dará voz de alarma instantáneamente, el guardia espera estar seguro de lo que ha visto, estableciendo así su umbral. Dependiendo de lo desconfiado que sea nuestro guardia, recopilará información para confirmar su sospecha durante más o menos tiempo, lo que forma su historial.

7. Pruebas

En este capítulo se detallarán las pruebas llevadas a cabo en el proyecto y se explicará el motivo de sus usos. En particular, nos centraremos en dos categorías de pruebas: las pruebas de aceptación del usuario y las pruebas de validación funcional.

7.1. Pruebas funcionales

Las pruebas funcionales tienen como objetivo verificar que el sistema cumpla con los requisitos y funcionalidades esperadas.

En nuestro contexto de detección de armas, al ser el caso de uso real la visualización de las cámaras en tiempo real, y no tener un conjunto de datos correctamente etiquetado para test, se ha decidido hacer un testeo visual. Al pasar personas frente a la cámara, se comprueba si el sistema detecta y clasifica correctamente las armas en tiempo real.

En estas pruebas, realizadas a 5 y 2 metros, verificamos que se detectan las armas de forma fluida durante todo el vídeo de prueba. Sumando el número total de fotogramas del vídeo y el número total de fotogramas con detección de armas, obtenemos un **58,33 %** de fotogramas detectados, un buen valor teniendo en cuenta que en una gran parte de los fotogramas las armas no son distinguibles divido al movimiento.

En la Prueba visual 1 (Figura 7.1) podemos observar un False Negative, una de las pistolas no está siendo detectada.



Figura 7.1: Prueba visual 1



Figura 7.2: Prueba visual 2



Figura 7.3: Prueba visual 3



Figura 7.4: Prueba visual 4

Por otra parte, podemos probar la correcta recepción de las alertas de Telegram enviadas por los detectores. (Esta última prueba también puede ser considerada una prueba de integridad entre la detección de objetos y el sistema de alertas).

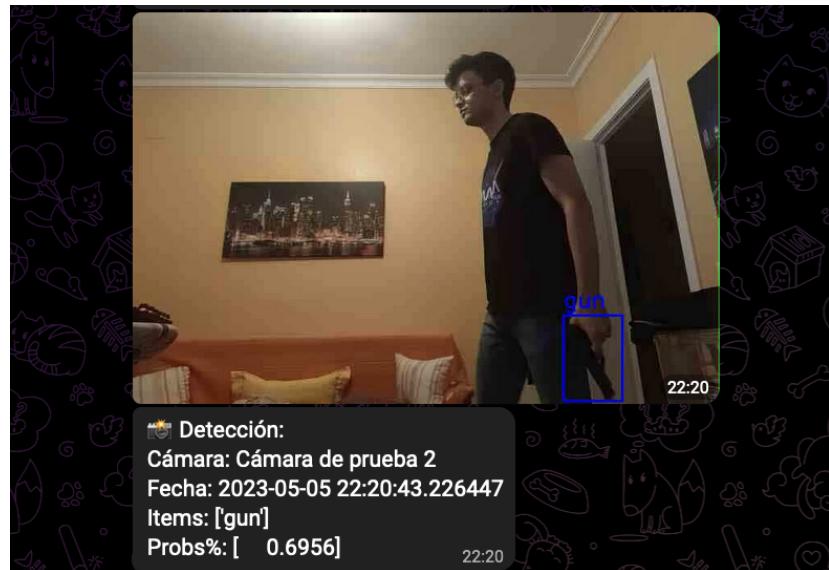


Figura 7.5: Recepción de alerta en Telegram

7.2. Pruebas no funcionales

Las pruebas no funcionales miden aspectos más amplios de calidad y rendimiento. Estas pruebas se centran en verificar y evaluar requisitos no funcionales, como la usabilidad, rendimiento, seguridad, escalabilidad, disponibilidad y confiabilidad del sistema.

7.2.1. Interfaz

En estas pruebas no funcionales, nos aseguramos que la interfaz se visualice y funcione correctamente en diferentes navegadores, sistemas operativos y tamaños de pantalla.



Figura 7.6: Interfaz adaptada a dispositivo móvil

En este proceso, hemos utilizado la herramienta “Web Developer” de navegador, para validar el código HTML y CSS de la interfaz web, asegurándonos que cumple con los estándares y buenas prácticas recomendadas. Además, hemos realizado pruebas de usabilidad y accesibilidad.

W3C El Servicio de Validación de CSS del W3C
Resultados del Validator CSS del W3C para TextArea (CSS versión 3)

Ir a: [Las Advertencias \(19\)](#) [Su Hoja de Estilo validada](#)

Resultados del Validator CSS del W3C para TextArea (CSS versión 3)

iEnhорабуна! No error encontrado.

¡Este documento es [CSS versión 3](#) válido!

Puede mostrar este icono en cualquier página que valide para que los usuarios vean que se ha preocupado por crear una página Web interoperable. A continuación se encuentra el XHTML que puede usar para añadir el icono a su página Web:

```
<p> <a href="http://jigsaw.w3.org/css-validator/check/referer">
    
</a>
</p>
```



```
<p> <a href="http://jigsaw.w3.org/css-validator/check/referer">
    
</a>
</p>
```

Figura 7.7: CSS validado por W3C

7.2.2. Inferencia

Dentro de las pruebas no funcionales realizadas en este proyecto, se destaca una específica relacionada con las optimizaciones en la inferencia. Esta prueba tiene como objetivo garantizar mejoras en el rendimiento y la escalabilidad del sistema al aumentar el intervalo de fotogramas en el que los fotogramas no pasan por el detector. Durante estas pruebas, utilizando un equipo con un procesador Intel® Core™ i7-6700, una tarjeta gráfica NVIDIA GEFORCE GTX 1060 de 6GB de memoria de video y 16 GB de RAM, se observó que al establecer este parámetro en su valor mínimo, 1, era posible utilizar el detector en dos cámaras simultáneamente. Sin embargo, al aumentar dicho parámetro a 4, se logró alcanzar un rendimiento óptimo en 9 cámaras en el mismo hardware.

7.3. Pruebas de validación del usuario

Finalmente, tras realizar todas las pruebas anteriores, realizamos las pruebas de validación del usuario. Estas pruebas se centran en evaluar la experiencia del usuario final al interactuar con el sistema, facilidad de uso y la capacidad del sistema para proporcionar una experiencia fluida.

7.4. Conclusiones

En conclusión, nuestro sistema de detección de armas ha demostrado un rendimiento eficiente en tiempo real, incluso sin la necesidad de contar con un conjunto de

datos de prueba etiquetado. Por otro lado, las pruebas no funcionales han revelado un rendimiento satisfactorio en cuanto a la interfaz y la inferencia. La interfaz se adapta correctamente a diferentes tipos de pantallas, mientras que la inferencia se realiza de forma altamente eficiente gracias a las optimizaciones implementadas.

8. Conclusiones y trabajo futuro

8.1. Conclusiones

Durante el desarrollo de este proyecto, hemos explorado varias tecnologías, desde la Visión por Computador con YoloV5, hasta las tareas de desarrollo front-end y back-end de la aplicación con Django. Sin embargo, enfrentamos desafíos al integrar estas tecnologías en un sistema de cámaras de seguridad en tiempo real.

Ahora que hemos completado el proyecto, es momento de reflexionar sobre sus aspectos positivos y negativos:

- La selección de tecnologías, la mayoría basadas en Python, ha permitido una integración y desarrollo fluidos entre ellas.
- Las optimizaciones implementadas en la arquitectura de inferencia han llevado a un aumento significativo en la cantidad de cámaras que pueden detectar objetos simultáneamente.
- El uso de IceVision ha simplificado el desarrollo, aunque ha limitado la elección de modelos que son compatibles con la aplicación.
- Habría sido interesante utilizar tecnologías más punteras en el desarrollo de la interfaz, como React. Sin embargo, la incorporación de dichas tecnologías podría haber extendido el tiempo de ejecución como los costos del proyecto, lo cual resultaba inviable dadas las horas para este proyecto.

La realización de este proyecto nos ha proporcionado la oportunidad de expandir nuestra perspectiva más allá del entrenamiento en inteligencia artificial, llegando hasta su implementación en un escenario real de aplicación. Además, nos ha dado la posibilidad de adentrarnos en el campo fascinante de la Visión por Computador, el cual no se aborda en profundidad en nuestra carrera.

8.2. Trabajo futuro

Una vez completado el proyecto, tras realizar un análisis posterior, hemos identificado algunas posibles mejoras que podrían implementarse en el futuro.

Cuando se produce una detección, se guarda información relevante como la fecha, hora, cámara, el detector utilizado y la imagen en la que se encuadra el arma. Sin embargo, para adaptar la aplicación a un contexto de investigación y generar imágenes etiquetadas, sería interesante guardar la imagen sin el arma enmarcada y guardar las coordenadas del encuadre. De esta manera, se podrían extraer automáticamente

imágenes anotadas para entrenar modelos futuros, lo que ahorraría tiempo en el proceso de anotación.

Otros aspectos a mejorar en el futuro sería añadir soporte para otros tipos de modelos, como TensorFlow y un sistema de logs al proyecto (tarea que se ha quedado en el backlog por falta de recursos), ya que ayudaría notablemente a solucionar posibles problemas que puedan surgir en la aplicación en un futuro.

Reconocemos que con estas mejoras mencionadas, su desempeño en este último mejoraría significativamente. Por lo tanto, tenemos la intención de continuar desarrollando estas mejoras con el objetivo de ofrecer el mejor producto posible.

Cabe destacar que aunque durante este documento se plantea el uso del proyecto para un entorno de vigilancia y seguridad, puede utilizarse para cualquier campo mientras se entrene un detector adecuado. Algunos ejemplos pueden ser detectores de caras, herramientas y animales.

En general, estamos muy satisfechos con el rendimiento del proyecto, ya que hemos logrado superar ampliamente la idea original planteada. Hemos evolucionado desde una única cámara hasta la incorporación de múltiples cámaras, además de implementar diversas optimizaciones como el sistema de caché y paciencia, junto con múltiples mejoras en cuanto a funcionalidad.

9. Bibliografía

- [1] Viso AI. Computer vision applications in surveillance and security, 2023. URL <https://viso.ai/applications/computer-vision-applications-in-surveillance-and-security/>.
- [2] Airctic. Icevision, 2023. URL <https://airctic.com/0.12.0/>.
- [3] Abdul Hanan Ashraf, Muhammad Imran, Abdulrahman M Qahtani, Abdulmajeed Alsufyani, Omar Almutiry, Awais Mahmood, and M Habib. Weapons detection for security and video surveillance using cnn and yolo-v5s. *CMC-Comput. Mater. Contin.*, 70:2761–2775, 2022.
- [4] Atlassian. Gitflow, 2023. URL <https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow>.
- [5] Bootstrap. Bootstrap, 2023. URL <https://getbootstrap.com>.
- [6] Dave Davies. Yolov5 object detection on windows: Step-by-step tutorial, 2023. URL <https://wandb.ai/onlineinference/YOLO/reports/YOLOv5-Object-Detection-on-Windows-Step-By-Step-Tutorial---VmlldzoxMDQwNzk4>.
- [7] Junta de Andalucía. Ingeniería de requisitos, atributos de los requisitos, 2023. URL <https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/409>.
- [8] Django. Django, 2023. URL <https://www.djangoproject.com>.
- [9] Fast.ai. Fast.ai, 2023. URL <https://www.fast.ai>.
- [10] Fast.ai. Fast.ai course, 2023. URL <https://course.fast.ai>.
- [11] Glassdoor. Glassdoor, 2023. URL <https://www.glassdoor.es>.
- [12] Jose L Salazar González, Carlos Zaccaro, Juan A Álvarez-García, Luis M Soria Morillo, and Fernando Sancho Caparrini. Real-time gun detection in cctv: An open problem. *Neural networks*, 132:297–308, 2020.
- [13] Red Hat. What is agile methodology?, 2023. URL <https://www.redhat.com/es/devops/what-is-agile-methodology>.
- [14] Indeed. Indeed, 2023. URL <https://www.indeed.es/salaries>.
- [15] Project Jupyter. Jupyter, 2023. URL <https://jupyter.org>.
- [16] Nimblework. Pruebas de aceptación - nimblework, 2023. URL <https://www.nimblework.com/es/agile/pruebas-de-aceptacion/>.
- [17] OpenCV. Opencv, 2023. URL <https://opencv.org>.

- [18] Paperspace. Pricing, 2023. URL <https://www.paperspace.com/pricing>.
- [19] Python. Python, 2023. URL <https://www.python.org>.
- [20] PyTorch. Yolov5, 2023. URL https://pytorch.org/hub/ultralytics_yolov5/.
- [21] Delong Qi, Weijun Tan, Zhifu Liu, Qi Yao, and Jingfeng Liu. A dataset and system for real-time gun detection in surveillance video using deep learning. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 667–672. IEEE, 2021.
- [22] Roboflow. Roboflow, 2023. URL <https://roboflow.com>.
- [23] SQLite. Sqlite, 2023. URL <https://www.sqlite.org/index.html>.
- [24] Los Andes Training. ¿qué son las pruebas de aceptación?, 2023. URL <https://www.linkedin.com/pulse/que-son-las-pruebas-de-aceptacion-los-andes-training/?originalSubdomain=es>.
- [25] Ultralytics. Yolov5, 2023. URL <https://github.com/ultralytics/yolov5>.

A. Manual de usuario

En este anexo se presentará un manual de usuario que detallará el uso de la aplicación y su interfaz.

A.1. Interfaz de usuario

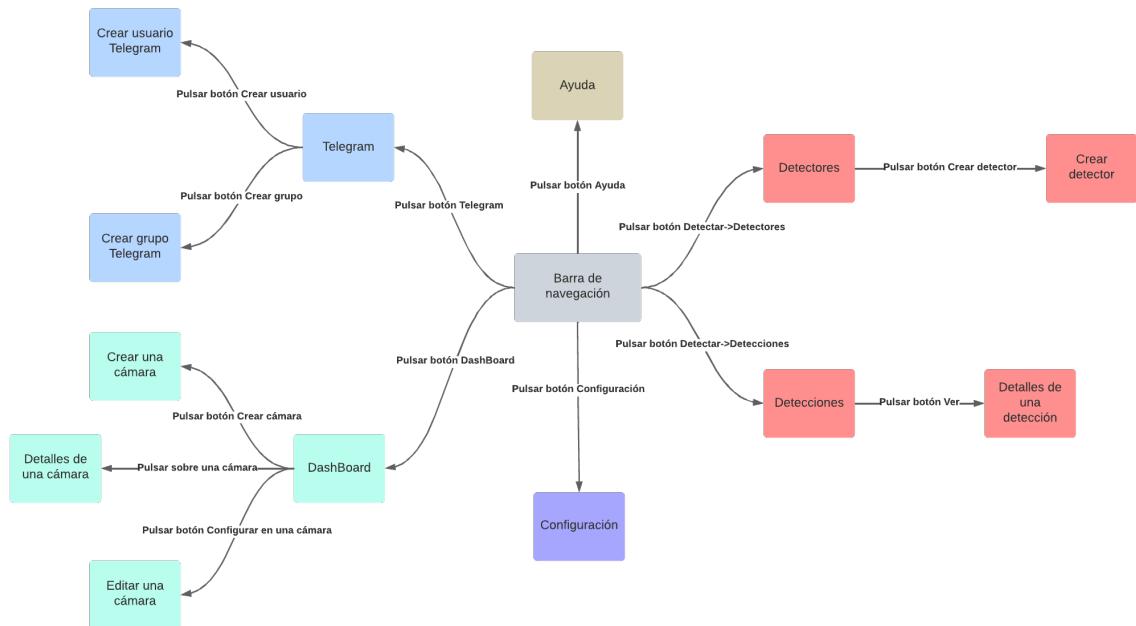


Figura A.1: Mapa conceptual de la interfaz

En esta sección mostraremos y explicaremos las distintas pantallas con las que el usuario se encontrará durante el uso de la aplicación.

A.1.1. Dashboard

Esta es la pantalla principal de la aplicación y en la que el usuario pasará el mayor tiempo. Desde aquí el usuario puede acceder al vídeo en directo de todas las cámaras y encontrará botones para crear o editar cámaras.

En caso de no tener ninguna cámara cargada en la aplicación , la pantalla se mostrará vacía con el mensaje *No se ha cargado ninguna cámara todavía*.

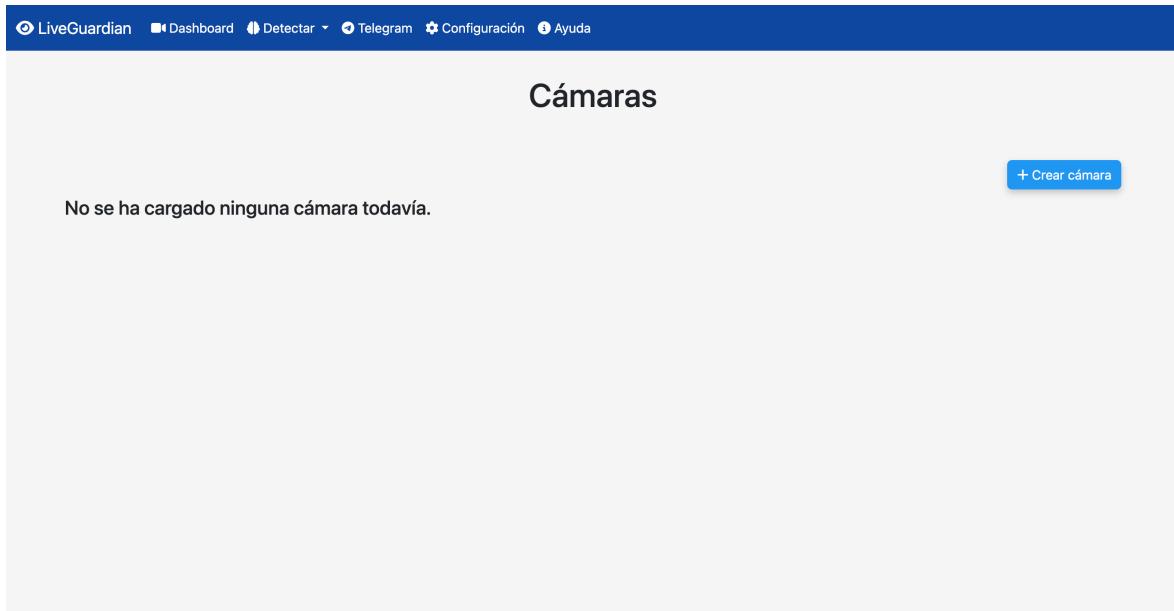


Figura A.2: Dashboard sin cámaras

Si la base de datos contiene cámaras, el usuario podrá acceder a previsualizaciones en tiempo real de todas las cámaras.

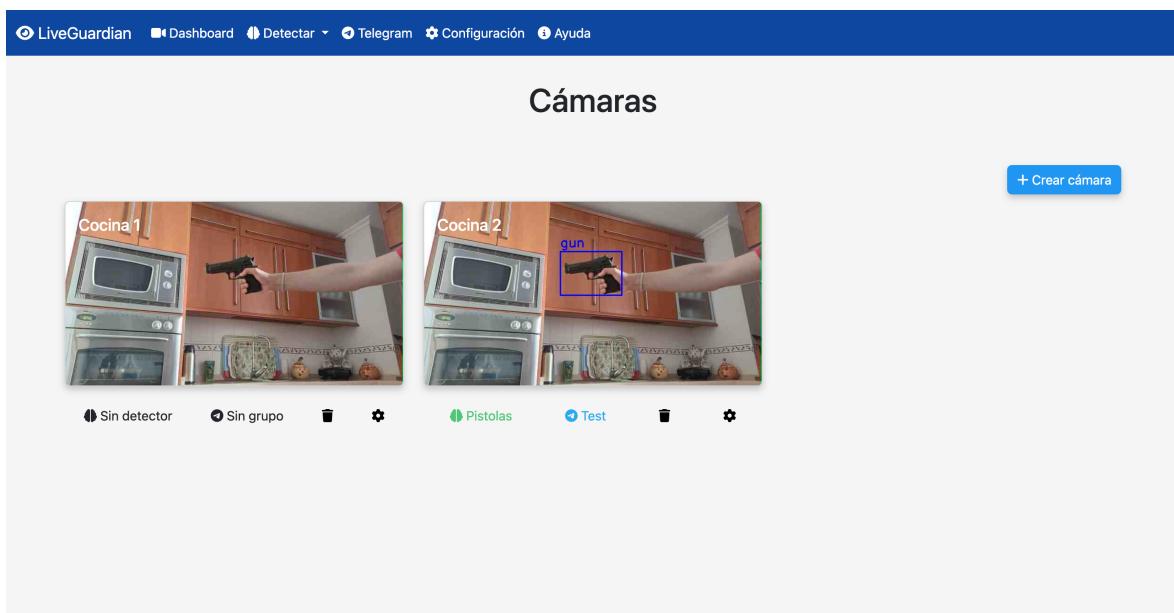


Figura A.3: Dashboard con cámaras

Desde la previsualización, obtenemos información acerca del detector empleado, el grupo de Telegram al que está asignada la cámara y botones para eliminarla o configurarla. Además, si se hace clic sobre la imagen, es posible acceder a una vista detallada de la cámara.

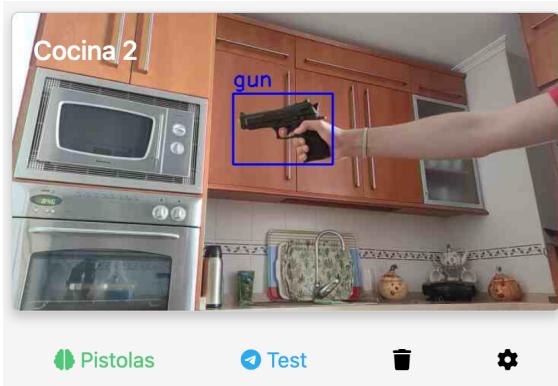


Figura A.4: Previsualización cámara

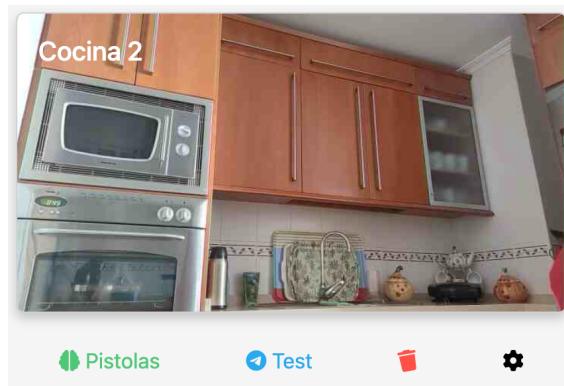


Figura A.5: Animación borrar

Si alguna cámara presenta problemas o pierde la conexión, la aplicación mostrará automáticamente una imagen de error en dicha cámara, junto con un contador que indica el tiempo transcurrido desde que comenzó a fallar. Si la cámara vuelve a estar operativa, retomará la transmisión de imágenes y la detección de objetos.

Figura A.6: Cámara fallando

A.1.2. Detalles de una cámara

Al hacer clic en la previsualización de una cámara, el usuario será dirigido a los detalles de la misma. En esta pantalla, podrá observar la cámara en tamaño ampliado y tendrá a su disposición varias opciones en la esquina superior izquierda de la imagen.

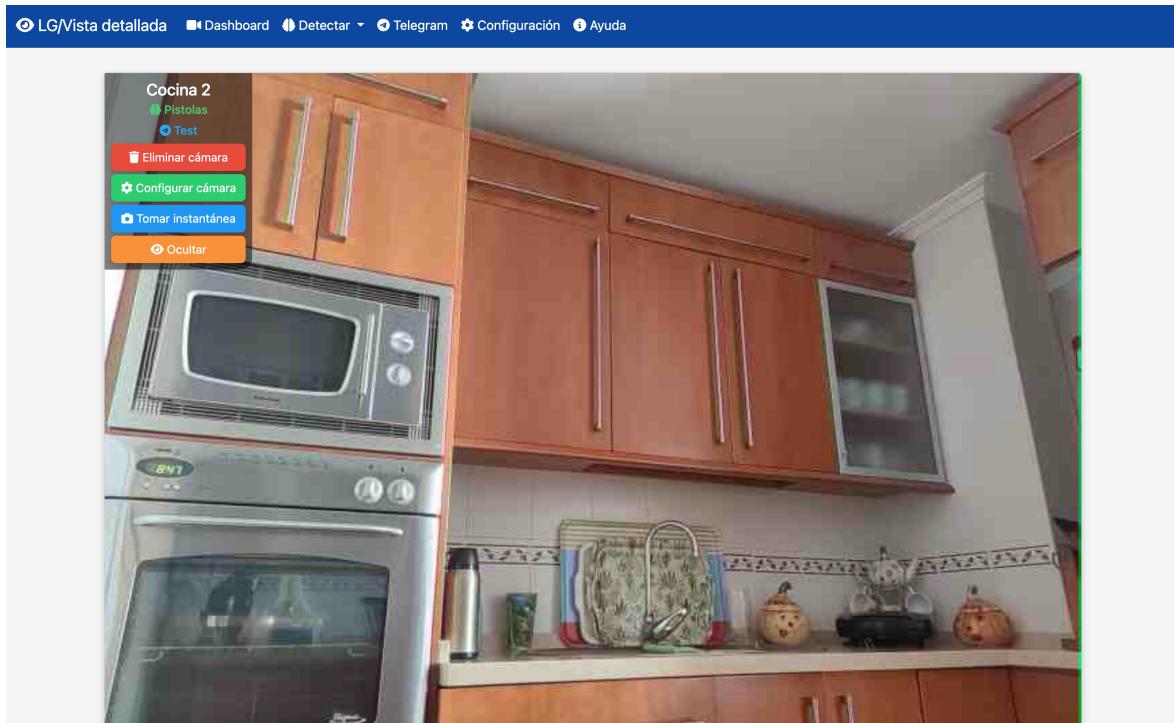


Figura A.7: Detalles cámara

En el menú situado en la esquina superior izquierda, el usuario puede acceder a información detallada de la cámara, como su nombre, el detector utilizado y el grupo asignado, así como a varias opciones, como eliminarla o configurarla, tomar una instantánea (esta instantánea se guardará como una detección sin objetos) y ocultar el menú.



Figura A.8: Panel desplegado

Figura A.9: Panel oculto

A.1.3. Creación/Edición de una cámara

En esta pantalla, el usuario tiene la posibilidad de crear/editar una cámara ingresando su nombre, detector, URL de la cámara y el grupo de Telegram para recibir

alertas en caso de detección. En caso de no seleccionar ningún detector, la cámara simplemene mostrará imagen en tiempo real sin realizar inferencia.

LG/Crear cámara

Nombre de la cámara:
Salón de casa

Seleccione un detector:

Url de la cámara:
http://192.168.1.36:8080/video

Seleccionar grupos:
Test

Crear

Figura A.10: Creación de una cámara

LG/Editar cámara

Nombre de la cámara:
Cocina

Seleccione un detector:

Url de la cámara:
http://192.168.1.36:8080/video

Seleccionar grupos:
Test

Crear

Figura A.11: Editar una cámara

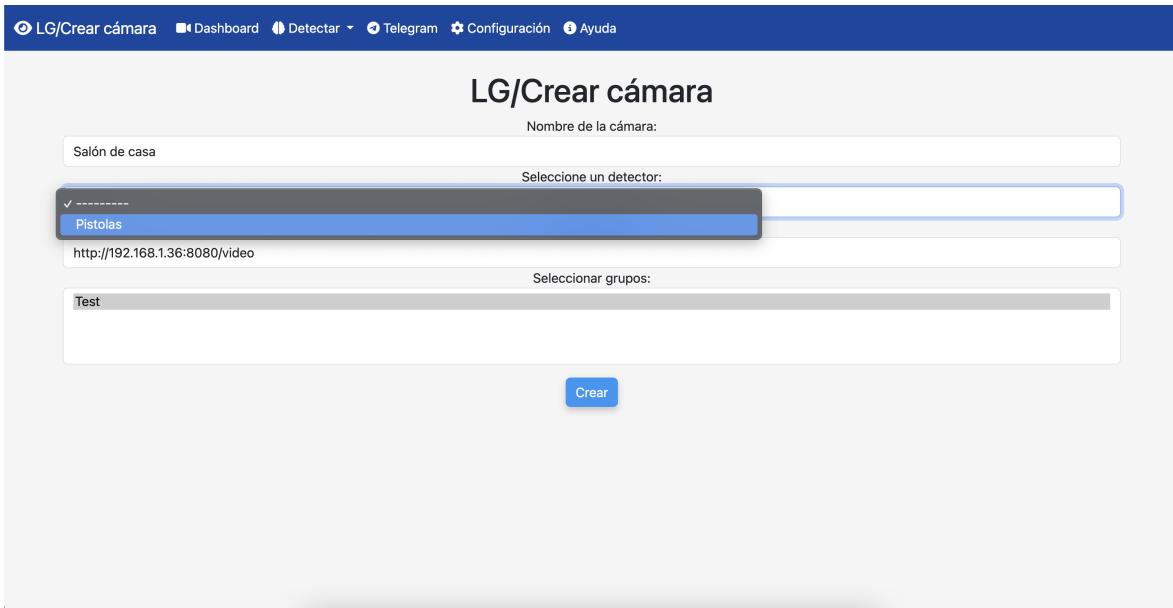


Figura A.12: Selector de detector en formulario cámara

A.1.4. Configuración

En esta pantalla, el usuario tiene la opción de modificar la configuración de la aplicación. Entre las opciones disponibles se incluyen: el número de fotogramas que deben contener una detección de objetos para ser registrada, el tamaño del historial de los últimos fotogramas con o sin detecciones, el intervalo de fotogramas en el cual se procesa la detección de objetos y el token del bot de Telegram que se utilizará para enviar las alertas.

- **Intervalo de fotogramas:** número de fotogramas en el cual se procesa la detección de objetos. Cuanto mayor sea este intervalo, menos fotogramas pasarán por el detector y más libre quedará este.
- **Tamaño del historial:** número de fotogramas anteriores considerados para enviar detecciones.
- **Número de fotogramas para detectar:** umbral de fotogramas con detección requerido en el historial para registrar una detección de objetos.
- **Token bot Telegram:** código que se utiliza para identificar y autenticar un bot en la API de Telegram.

Configuración

Intervalo de fotogramas	3	Este intervalo indica cada cuantos fotogramas será uno introducido en el detector.
Tamaño del historial	40	Número de fotogramas almacenados (un número mayor puede producir retraso en las detecciones)
Nº de fotogramas para detectar	10	Número de fotogramas necesarios en el historial para producirse una detección.
Token bot Telegram	5282233910:AAG_mddkn8zdw_lip-n1zQX_gSURiBLomC0	Token del bot de Telegram utilizado en el sistema.

Guardar

Figura A.13: Configuración de la aplicación

A.1.5. Detectores

En esta pantalla, el usuario puede visualizar una lista con todos los detectores de objetos subidos a la aplicación, además de gestionar los mismos, creando o eliminando detectores.

Detectores

Eliminar	Nombre	Modelo
	Pistolas	models/dataSetGrande.pth

+ Crear detector

Figura A.14: Lista de detectores

A.1.6. Crear detección

En esta pantalla el usuario puede subir sus modelos entrenados para la detección de objetos. Tan solo tiene que asignar un nombre al detector y subir el archivo con su modelo. Este modelo debe de ser un archivo de un modelo entrenado con Icevision, ya que necesitamos sus metadatos para inicializar la arquitectura del modelo.

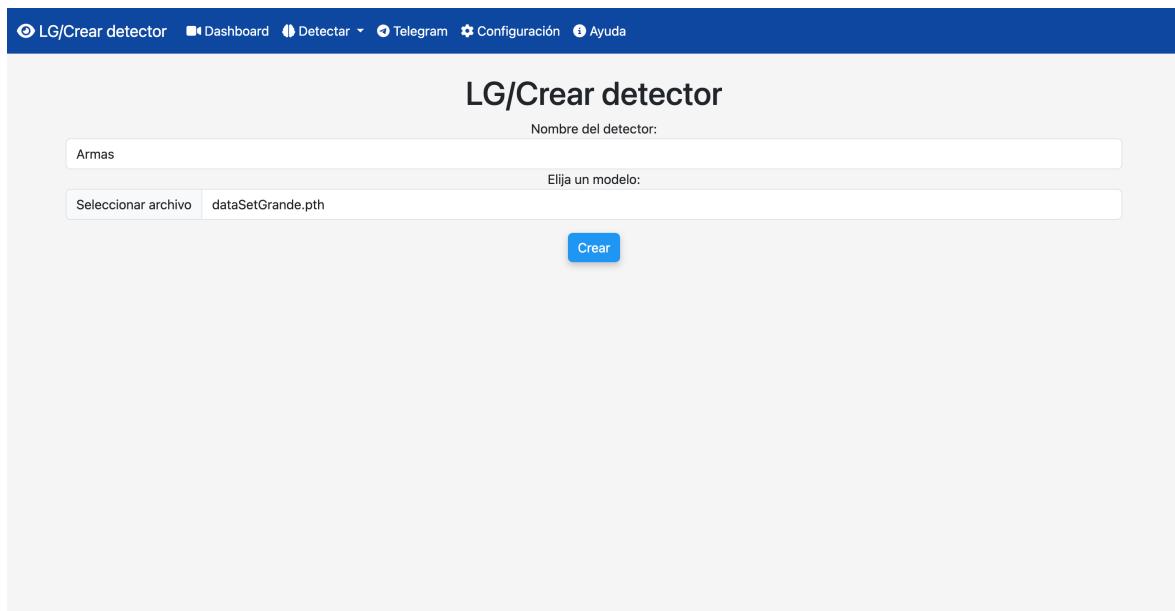


Figura A.15: Subir un detector

A.1.7. Detecciones

En esta pantalla, el usuario puede ver un resumen de todas las detecciones ocurridas en una lista y tiene la opción de acceder a cada una de ellas para obtener una vista más detallada.

Detecciones				
①	🔗 Id	📷 Cámara	📅 Fecha	🔫 Detector
Ver	10	Cocina 2	1 de mayo de 2023 a las 18:22	Pistolas
Ver	9	Cocina 2	1 de mayo de 2023 a las 18:21	Pistolas

Figura A.16: Lista de detecciones

A.1.8. Detección detallada

Cuando el usuario hace clic en el botón *Ver* en la lista de detecciones, accede a una vista detallada que muestra información adicional sobre la detección, como los objetos detectados, las probabilidades de cada objeto y una imagen del momento en que se detectó. Además, el usuario tiene la opción de eliminar la detección y su imagen.

➊ LG/Vista detallada ➋ Dashboard ➌ Detectar ➍ Telegram ➎ Configuración ➏ Ayuda



Detección #10

📷 Cámara:	Cocina 2
🔫 Detector:	Pistolas
📦 Items:	['gun']
✖ Probabilidades:	[0.8669]
📅 Fecha:	1 de mayo de 2023 a las 18:22
Eliminar detección	

Figura A.17: Detalles de una detección

A.1.9. Telegram

En esta pantalla el usuario puede gestionar de forma directa los Usuarios y Grupos de Telegram, permitiendo eliminar existentes o crear nuevos.

The screenshot shows a dashboard for managing Telegram users and groups. At the top, there are navigation links: LG/Telegram, Dashboard, Detectar, Telegram, Configuración, and Ayuda. Below the navigation, there are two main sections: 'Grupos' (Groups) and 'Usuarios' (Users).

Grupos (Groups):

	Nombre	Nº usuarios	Acciones
	Test	1	

Usuarios (Users):

	Nombre	Chat Id	Grupo	Acciones
	José Joaquín Virtudes Castro	900224881	Test	

Both sections have a blue 'Crear' button: '+ Crear grupo' for Groups and '+ Crear usuario' for Users.

Figura A.18: Usuarios y grupos de Telegram

A.1.10. Crear usuario de Telegram

En esta pantalla, el usuario puede crear un usuario de Telegram. Para ello, debe proporcionar un nombre, el ID del chat de Telegram y el grupo al que se asignará dicho usuario.

LG/Crear usuario

Nombre del usuario:

José Joaquín Virtudes Castro

Id del chat:

900224881

Grupos:

Test

Crear

Figura A.19: Crear usuario de Telegram

A.1.11. Crear grupo de Telegram

Para crear un grupo de Telegram, el usuario tan solo tendrá que indicar un nombre.

LG/Crear grupo

Nombre del grupo:

Seguridad interna

Crear

Figura A.20: Crear grupo de Telegram

A.1.12. Mensaje de Telegram

Una vez que el usuario ha configurado un grupo y un detector en una cámara, en caso de que se produzca una detección, el usuario de Telegram recibirá un mensaje estructurado con la siguiente forma:

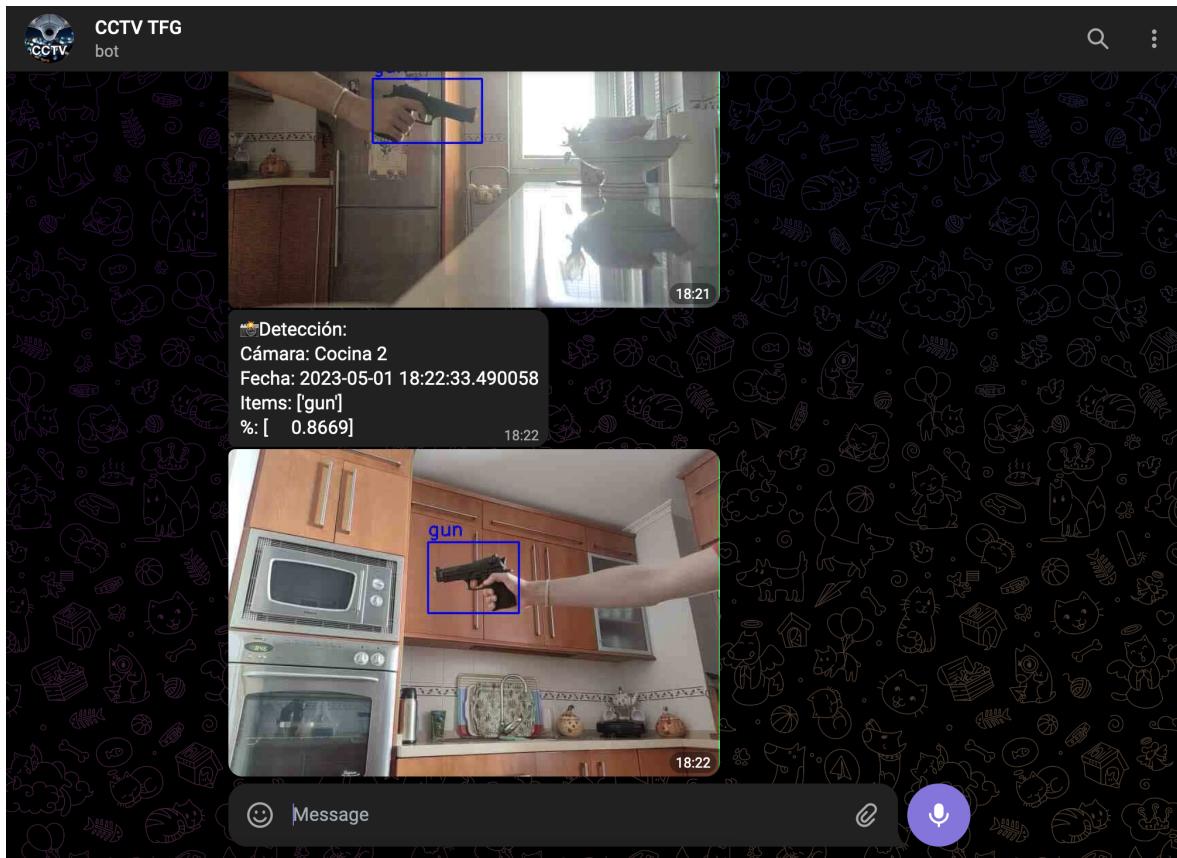


Figura A.21: Chat Telegram

B. Instalación

En este anexo, se proporcionarán las instrucciones necesarias para instalar y poner en marcha las herramientas necesarias para la ejecución de la aplicación. Se presentarán los pasos necesarios para instalar Django, IceVision y OpenCV. Además se hablará del proceso de creación y configuración de un bot de Telegram.

B.0.1. Versiones

Para este proyecto, se han utilizado los siguientes paquetes y herramientas:

- Python 3.9.13
- OpenCV 4.6.0.66
- Django 4.1.3
- numpy 1.22.3
- Icevision 0.12.0
- pyTelegramBotApi 4.7.0
- yolo5-icevision 6.0.0
- torchvision 0.11.1
- torch 1.10.0
- fastai 2.5.6
- Pillow 8.4.0
- mmcv-full 1.3.17
- CUDA 11.4

Como veremos más adelante, dependiendo del entorno y los drivers del sistema, las versiones de estos paquetes puede cambiar. Los paquetes que mayormente se verán afectados serán torch, torchvision, mmcv-full y CUDA.

B.0.2. Python

Si se encuentra en un sistema Windows, puede descargar Python desde el sitio web oficial o desde la tienda de Microsoft. En MacOS puede obtenerlo desde el sitio web oficial o mediante gestores de paquetes como Homebrew. En Linux, puede instalar Python desde el gestor de paquetes que proporcione su distribución.

Una vez contamos con Python instalado en nuestro sistema, es recomendable crear un entorno virtual para el proyecto. Un entorno virtual es un espacio aislado donde puedes instalar paquetes sin afectar a la instalación Python del sistema. Para crear un entorno virtual, se abrirá una terminal y ejecutarán los siguientes comandos:

```
1 #!/bin/bash
2 # Creamos el entorno virtual
3 python -m venv liveguardian
4 # Activamos el entorno virtual
5 source liveguardian/bin/activate
```

Extracto de código B.1: Crear entorno virtual

B.0.3. CUDA

CUDA es un framework para computación paralela que permite aprovechar el poder de procesamiento de tarjetas gráficas NVIDIA.

Durante este apartado explicaremos la instalación de CUDA para Ubuntu (será similar para cualquier otra distribución linux). Para Windows, sugerimos acudir al sitio web oficial de NVIDIA para descargar el instalador de CUDA Toolkit. En el caso de macOS, se debe tener en cuenta que NVIDIA CUDA Toolkit no está oficialmente soportado.

Antes de instalar CUDA Toolkit tenemos que tener instalados los drivers de NVIDIA en nuestro sistema:

```
1 #!/bin/bash
2 # Cambiar <version> por una compatible con el CUDA Toolkit a instalar.
3 sudo apt update
4 sudo apt install nvidia-driver-<version>
```

Extracto de código B.2: Instalar drivers NVIDIA

La forma más sencilla para instalar CUDA Toolkit en Ubuntu es partiendo de los repositorios oficiales de la distribución:

```
1 #!/bin/bash
2 sudo apt update
3 sudo apt install nvidia-cuda-toolkit
```

Extracto de código B.3: Instalar CUDA Toolkit

Una vez instalado, podemos verificar la instalación comprobando la versión:

```
1 #!/bin/bash
2 nvcc --version
```

Extracto de código B.4: Comprobar versión instalada

B.0.4. IceVision

Dependiendo de la versión instalada de CUDA o si se desea realizar inferencia mediante CPU, se tendrán que instalar distintas versiones de torch y torchvision:

```
1 #!/bin/bash
2
3 # CUDA 10.2
4 pip install torch==1.10.0+cu102 torchvision==0.11.1+cu102 -f https://
   download.pytorch.org/whl/torch_stable.html
5 # CUDA 11.1
6 pip install torch==1.10.0+cu111 torchvision==0.11.1+cu111 -f https://
   download.pytorch.org/whl/torch_stable.html
7 # CPU
8 pip install torch==1.10.0+cpu torchvision==0.11.1+cpu -f https://download
   .pytorch.org/whl/cpu/torch_stable.html
```

Extracto de código B.5: Instalar torch y torchvision

De forma opcional podemos instalar **mmcv-full** para ampliar la cantidad de modelos soportados por nuestro sistema. En nuestro caso, para correr nuestro modelo YOLOV5 no será necesaria la instalación de este paquete. Dependiendo de la versión de CUDA o si desea realizar inferencia mediante CPU, se tendrá que instalar una versión distinta:

```
1 #!/bin/bash
2
3 # CUDA 10.2
4 pip install mmcv-full==1.3.17 -f https://download.openmmlab.com/mmcv/dist
   /cu102/torch1.10.0/index.html
5 pip install mmdet==2.17.0
6 # CUDA 11.1
7 pip install mmcv-full==1.3.17 -f https://download.openmmlab.com/mmcv/dist
   /cu111/torch1.10.0/index.html
8 pip install mmdet==2.17.0
9 # CPU
10 pip install mmcv-full==1.3.17 -f https://download.openmmlab.com/mmcv/dist
    /cpu/torch1.10.0/index.html
11 pip install mmdet==2.17.0
```

Extracto de código B.6: Instalación opcional Icevision

Una vez instaladas las dependencias, procedemos a instalar Icevision. En este proyecto optaremos por su versión completa y estable:

```
1 #!/bin/bash
2 pip install icevision[all]
```

Extracto de código B.7: Instalación Icevision

B.0.5. Django

Una vez tenemos creado y activado nuestro entorno virtual Python, la instalación de Django es tan sencilla como:

```
1 #!/bin/bash
2 pip install Django
```

Extracto de código B.8: Instalar Django

B.0.6. Pillow

Necesitaremos esta librería para poder inyectar las imágenes en tiempo real a los modelos:

```
1 #!/bin/bash
2 pip install pillow
```

Extracto de código B.9: Instalar Pillow

B.0.7. OpenCV

OpenCV será el encargado de procesar las imágenes de las fuentes en tiempo real, además de realizar transformaciones a las mismas:

```
1 #!/bin/bash
2 pip install opencv-python
```

Extracto de código B.10: Instalar openCV

B.0.8. pyTelegramBotApi

```
1 #!/bin/bash
2 pip install pyTelegramBotAPI
```

Extracto de código B.11: Instalar pyTelegramBotAPI

El resto de paquetes mencionados son instalados como dependencias de los instalados en este manual. En caso de no instalarse alguno de estos, el proceso de instalación es similar al seguido anteriormente.

B.1. Puesta en marcha

Una vez tenemos toda la instalación necesaria para ejecutar nuestra aplicación, podemos empezar con los preparativos para un primer inicio.

Antes de iniciar la aplicación, será necesario inicializar la base de datos. Para esto nos situaremos en el directorio principal de la aplicación, donde encontraremos el fichero llamado **manage.py** y ejecutaremos los siguientes comandos (es necesario tener activado el entorno virtual mencionado en la instalación):

```
1 #!/bin/bash
2 python manage.py makemigrations web
3 python manage.py migrate web
```

Extracto de código B.12: Inicializar base de datos

Una vez que hayamos configurado la base de datos, necesitaremos hacer algunos cambios en el archivo **settings.py**. Allí encontraremos dos campos que tenemos que editar: **ALLOWED_HOSTS** y **DEBUG**.

En el primer campo, añadiremos en forma de string la IP del host de la aplicación, en el segundo, en caso de ejecutar la aplicación en un entorno de producción, **DEBUG** de **True** a **False** para desactivar los mensajes de desarrollo.

```
1 # settings.py
2 DEBUG = True
3 ALLOWED_HOSTS = ['Ip del Host']
```

Extracto de código B.13: Editar **settings.py**

Finalmente, iniciar la aplicación es tan sencillo como ejecutar el siguiente comando con la IP del host y el puerto en el que se desea desplegar la app:

```
1 #!/bin/bash
2 python manage.py runserver <ip_host>:<puerto>
```

Extracto de código B.14: Inicializar base de datos