PROYECTO FINAL DE CARRERA

Ingeniería en Sistemas de Información

UTN Facultad Regional Santa Fe

"Herramienta de software para la detección automática de armas mediante técnicas basadas en aprendizaje profundo"

ARCA VISINTINI, Facundo - LU 23178

ROA, Santiago - LU 23390

Director: Dr. Jorge Roa

Co Directora: Dra. María de los Milagros Gutiérrez

MARZO DE 2021

Índice

1 Introducción

"La **seguridad ciudadana** es el proceso de establecer, fortalecer y proteger el orden civil democrático, eliminando las amenazas de violencia en la población y permitiendo una coexistencia segura y pacífica. Se le considera un bien público e implica la salvaguarda eficaz de los derechos humanos inherentes a la persona."

El crimen y la inseguridad son temáticas prevalentes en las sociedades modernas, particularmente en nuestro país, y más específicamente en la Provincia de Santa Fe, en donde la tasa de homicidios dolosos cada 100.000 habitantes es la más alta del país (9,7), y excede ampliamente tanto a las tasas de otras Provincias más densamente pobladas como Córdoba (3,3), Buenos Aires (5,2) y CABA (3,3), así como a la media nacional (5,1) **[8]**. Sus consecuencias afectan de manera directa o indirecta a toda la sociedad, y condicionan nuestros hábitos cotidianos y cómo interactuamos como comunidad.

Todos hemos sido afectados, o conocemos a alguien que ha sido afectado por un hecho violento de inseguridad. El conjunto de daños que genera un crimen es sumamente amplio, desde la pérdida de bienes materiales hasta la pérdida de vida.

La criminalidad representa, además, un enorme gasto económico para el Estado y, consecuentemente, para cada miembro de la sociedad. El mismo debe utilizar los fondos públicos para sostener actividades policiales, el sistema judicial y penitenciario, programas de reinserción, incluyendo los salarios de todo el personal propio de estas tareas.

Para poner en perspectiva, se estima que, en América Latina y Caribe, el costo monetario de la criminalidad es de aproximadamente U$D 261.000 millones, un 3.55% del PBI total de la región.

Particularmente en Argentina, se estima que los costos asociados a la criminalidad en el año 2014 pueden haber sido tan altos como U$D 29.380 millones (2.97% del PBI nacional). En términos cotidianos, para ese mismo año, se estima que el costo anual per cápita de la criminalidad puede haber sido de hasta U$D 688.56. **[21]**

Estos fondos podrían ser utilizados en áreas que aporten a la comunidad y a la actividad económica de la región.

Queda en evidencia que es fundamental buscar formas de optimizar los mecanismos disponibles para la detección y prevención de hechos de inseguridad.

Una de las opciones modernas más elegidas, tanto por ciudadanos particulares, para salvaguardar sus intereses, como el Estado, como política de seguridad, es la instalación de cámaras de monitoreo y seguridad.

En la actualidad nuestra provincia cuenta con varios centros de monitoreo ubicados en las grandes urbes, los cuales, a fecha de 28 de agosto de 2015, reportaban poseer una cantidad de 1200 cámaras ubicadas en su mayoría en la ciudad de Rosario y Santa Fe respectivamente.

"Con una inversión de 13,7 millones de dólares, se suman a los 600 equipos ya instalados en las ciudades de Rosario y Santa Fe. También se equiparán los dos nuevos helicópteros."

"En una primera etapa, el Gobierno de Santa Fe puso en funcionamiento con una inversión de 10 millones de dólares 400 cámaras de videovigilancia en Rosario, 200 en la ciudad de Santa Fe y 20 en San Lorenzo, que monitorean las 24 horas del día en distintos puntos de los mencionados territorios" **[26]**

Tomando en cuenta que una persona sólo puede observar una cámara a la vez en un instante determinado, que puede distraerse, debe tomar descansos, días por enfermedad, etc., quedan claramente expuestas las ventajas de un sistema autónomo.

Es en este contexto en el cual presentamos nuestro proyecto final de carrera, que busca aportar un avance inicial para la integración de una de las tecnologías más avanzadas de procesamiento de imágenes a la vida cotidiana, ayudando a facilitar la identificación y prevención de posibles hechos de inseguridad.

El objetivo puntual del mismo es el desarrollo de una herramienta de software para la detección automática de armas de fuego en videos de cámaras de seguridad mediante el uso de técnicas de análisis de imágenes en tiempo real.

La intención es proveer un acercamiento al proceso de desarrollo de herramientas automáticas de detección de imágenes, así como ofrecer a su audiencia un sistema que permita detectar con la mayor precisión y en el menor tiempo posible la presencia de un arma de fuego.

Creemos fuertemente que el tiempo de reacción es un factor decisivo en este tipo de hechos, reaccionar con rapidez puede reducir en gran medida los hechos fatales y/o los daños materiales causados, por lo que contar con un sistema que reconozca un arma de fuego en los primeros instantes en los que la misma está presente, puede ser increíblemente ventajoso en las

situaciones mencionadas (advertir a las fuerzas policiales, neutralizar un atacante, evacuar a personal, entre otros).

Además de esto, la automatización del proceso de detección de los hechos violentos, al minimizar la necesidad de supervisión constante por personal, facilita en gran medida la escalabilidad del sistema de monitoreo si, por ejemplo, se quiere adquirir más cámaras y/o fuentes de información.

Los sistemas de detección automática de objetos ya son una realidad: en las cámaras de nuestros celulares, automóviles, técnicas médicas de vanguardia, e incontables campos de investigación y desarrollo.

Sin embargo, en su mayoría, son de propósito general. El objetivo de este proyecto es proveer una herramienta especializada para este tipo de situaciones tan críticas, teniendo en cuenta el contexto socioeconómico de nuestro país, priorizando criterios como la prevalencia de cada tipo de arma, así como los factores limitantes propios de la calidad de las cámaras a las cuáles hay acceso.

2 Marco teórico

2.1 Inteligencia Artificial (IA)

No existe una definición clara, concisa e infalible sobre qué se entiende por "inteligencia artificial", particularmente, sobre cuándo una herramienta computacional es "inteligente", y cuándo no lo es.

Se logra identificar que las definiciones de IA varían a lo largo de dos grandes ejes:

El eje de la **racionalidad/subjetividad**, que evalúa qué premisas deben guiar a una herramienta computacional para ser considerada inteligente, su búsqueda por la decisión más racional, o su cercanía a la inteligencia humana, de tinte menos objetivo.

El eje de la **granularidad**, que evalúa qué tan internalizadas están esas premisas en la estructura de una herramienta considerada inteligente. En otras palabras, ¿debe **ser** racional o humana, o simplemente debe **parecer** racional o humana?

Según Russel y Norvig **[34]**, pueden distinguirse 4 grandes grupos que engloban las características comunes de la mayoría de las definiciones de IA, según el análisis anterior:

- Pensamiento interno humano → "Ser como un humano"
    - "La automatización de actividades que asociamos con el pensamiento humano, como la toma de decisiones, resolución de problemas y aprendizaje" [Bellman, 1978]
- Pensamiento interno racional → "Ser un agente racional"
    - "El estudio de facultades mentales mediante el uso de modelos computacionales" [Charniak y McDermott, 1985]
- Comportamiento externo humano → "Parecer un humano"
    - "Una computadora merece ser llamada inteligente si puede convencer a una persona de que es un humano" [Turing, 1950]
- Comportamiento externo racional → "Parecer un agente racional"
    - "La inteligencia computacional es el estudio del diseño de agentes inteligentes" [Poole, 1998]

Queda en evidencia que la naturaleza de un agente computacional inteligente está sujeta a la necesidad y la expectativa de quien haga uso del mismo.

A lo largo de este proyecto, se entiende como Inteligencia Artificial al **estudio de agentes computacionales que se comportan de manera racional,** es decir, agentes que idealmente, ante un problema, tomen la mejor decisión, lo más óptima y objetiva posible, no necesariamente humana, e independientemente de los mecanismos internos que utiliza para llegar a dicha decisión, sean o no similares al razonamiento lógico tradicional.

2.1.1 Métodos de aprendizaje

Un agente inteligente aprende a través del análisis de datos que percibe de su entorno. Lo hace con el propósito de mejorar su performance en la tarea a la cual está asignado. La naturaleza de este proceso de aprendizaje puede ser:

- **Inductiva**: Propone una regla específica y busca maximizar su capacidad de generalización.
- **Deductiva**: Parte de una regla general y busca especificarla al problema en cuestión.

El agente a su vez debe probar las reglas que propone, sea cual fuera el método que utilizó para obtenerlas. Esto se logra mediante el tipo de **feedback** o retroalimentación que éste recibe de su entorno.

Existen tres grandes tipos de aprendizajes, según el tipo de feedback que el agente percibe:

- **No Supervisado:** El agente no recibe feedback explícito. Aprende a identificar patrones similares entre sí, con el propósito de agruparlos.
- **Por refuerzo:** El agente recibe una medida de qué tan bueno ha sido su comportamiento, pero el feedback no incluye ninguna información sobre cuál es la causa de esa medida, ni qué conjunto de decisiones lo llevaron a ese valor.
- **Supervisado:** El agente aprende de un conjunto de pares entrada/salida ya existentes y ponderados. Cuando toma una decisión, lo hace teniendo en cuenta una entrada, y puede evaluar el resultado que obtuvo con respecto a la salida esperada. Su propósito se reduce, entonces, a encontrar una función que mapee lo más certeramente los pares entrada/salida que se le proveen, es decir, busca **maximizar su precisión**.

Nuestro proyecto es un ejemplo de inteligencia artificial con un aprendizaje de naturaleza **deductiva** y **supervisada**.

2.2 Machine Learning y Deep Learning

Machine Learning (ML) es un subconjunto de las ciencias de la computación y de la inteligencia artificial, cuyo objetivo es la construcción de algoritmos que, para ser de utilidad, depende de una colección de ejemplos de algún fenómeno. **[5] Deep Learning (DL)** es un subconjunto del ML que se caracteriza puntualmente por el estudio de algoritmos que aplican redes neuronales multinivel o con varias "capas" para el aprendizaje.

También puede definirse como el proceso de resolver un problema práctico mediante la conformación de un dataset y la construcción algorítmica de un modelo estadístico basado en el mismo. El modelo generado se supone que es de utilidad para la resolución del problema.

Los algoritmos de machine learning, por extensión, pueden aplicar cualquiera de los métodos de aprendizaje mencionados anteriormente. Puntualmente, un **algoritmo de machine learning con entrenamiento supervisado** funciona de la siguiente manera:

El proceso de entrenamiento supervisado comienza con la obtención de los datos. Como se mencionó anteriormente, estos toman un formato de pares (entrada, salida). La entrada puede ser cualquier cosa (en nuestro caso, imágenes). Las salidas suelen ser valores numéricos.

De manera general, se define que el dataset es la colección de valores etiquetados

$$\{(x_i, y_i)\}\, i \in (1, N)$$

Cada elemento x_i se llama **vector de características**. Un vector de características contiene un conjunto de valores que modelan la naturaleza de la entrada, en los atributos relevantes para el algoritmo de ML. Cada una de las dimensiones $j \in (1, D)$ del vector de características contiene un valor que describe a la entrada de alguna forma.

Por ejemplo, si la entrada del algoritmo representa una persona, su vector de características podría incluir su altura, su peso, su edad, etc.

Cada uno de estos valores se define como una **característica** y se denota como $x^{(j)}$.

Es importante recalcar que todos los vectores de características comparten su estructura entre sí, por lo que, si la j-ésima característica en x_i representa la altura de una persona, también lo hace para todas las otras entradas del dataset.

La salida o etiqueta y_i puede pertenecer a un conjunto finito de clases discretas (persona, perro, gato, etc), ser un valor numérico real, o una estructura más compleja (vector, matriz, etc).

2.3 Computer Vision

Computer vision (CV) es el campo de estudio que busca desarrollar técnicas que le permiten a las computadoras entender el contenido presente en una imagen digital. Pertenece al campo de IA y los algoritmos más utilizados actualmente son redes neuronales multinivel, por lo que está estrechamente relacionado con DL.

“La visión por computadora es la ciencia de los sistemas de software que pueden reconocer y comprender imágenes y escenas.” **[35]**

También se compone de varios aspectos, como el reconocimiento de imágenes, la detección de objetos, la generación de imágenes, la superresolución de imágenes y más. La detección de objetos es probablemente el aspecto más profundo de la visión por computadora, debido a la cantidad de casos de uso práctico que provee.

La detección de objetos se refiere a la “capacidad de los sistemas informáticos y de software para localizar objetos en una imagen/escena e identificar cada objeto”. **[20]**

La detección de objetos se ha utilizado ampliamente para detección de rostros, detección de vehículos, conteo de peatones, imágenes web, sistemas de seguridad y vehículos sin conductor.

Un ejemplo de soluciones de reconocimiento de imágenes es el reconocimiento facial; por ejemplo, para desbloquear su smartphone debe dejar que escanee su cara. Entonces, en primer lugar, el sistema debe detectar el rostro, luego clasificarlo como rostro humano y sólo entonces decidir si pertenece al propietario del smartphone.

2.4 Algoritmos de Computer Vision

La siguiente sección contiene un análisis comparativo entre las principales opciones de algoritmos de Computer Vision modernos. Se explica su naturaleza y funcionamiento, y se hace hincapié en las métricas de su performance.

La primera característica fundamental que suele considerarse para evaluar la performance de una red neuronal es su precisión. Una red que no detecta precisamente los objetos que debería no es de utilidad.

Sin embargo, debido a que una de las necesidades de este proyecto es que la herramienta pueda ejecutarse en tiempo real, la velocidad de ejecución es un factor fundamental.

Por lo tanto, se busca la herramienta que maximice la relación precisión/velocidad de ejecución.

La rápida adopción del aprendizaje profundo en la última década dio lugar a una gran variedad de algoritmos y métodos de detección de objetos modernos: R-CNN, Fast R-CNN, Faster R-CNN, RetinaNet, YOLO y SSD, entre otros, algunos de mayor precisión y otros con mejor performance.

A continuación, se describen las características principales de los detectores de imágenes considerados, sus ventajas y desventajas con respecto a las necesidades del proyecto, y se justifica la elección del algoritmo utilizado en este proyecto.

R-CNN, Fast R-CNN y Faster R-CNN:

Existe un problema con el enfoque tradicional de una red convolucional (CNN) aplicado para la detección de objetos. Una CNN tradicional no tiene la capacidad de detectar múltiples instancias de un mismo objeto en la imagen.

Se podría entrenar una CNN para detectar si en una imagen existe un arma (o cualquier otro objeto), pero no se podría saber en qué parte de la imagen ésta se encuentra y, por lo tanto, tampoco se podrían distinguir dos armas distintas en una imagen.

Una solución burda a este problema es subdividir la imagen en partes sin ningún criterio, aplicar una CNN a cada una de ellas, y evaluar la salida en conjunto. Sin embargo, esto es computacionalmente impráctico, ya que requiere entrenar cada una de las CNN individualmente.

R-CNN surgió como la primera solución viable a este problema. La idea fundamental es introducir un algoritmo de proposición de regiones (2000 regiones puntualmente) para evitar tener que trabajar con la cantidad enorme de regiones que implicaría subdividir toda la imagen siempre de la misma forma.

Seguido, cada región es tratada de la misma forma como si fuera una imagen en una CNN tradicional: se extraen sus características mediante la(s) capa(s) de convolución y luego un clasificador toma ese vector de características para definir a qué objeto del output se asemeja más.

Este algoritmo sigue limitado por el hecho de que su entrenamiento es computacionalmente laborioso, la detección es lenta (por encima de los 40s por cada imagen) y el algoritmo de proposición de regiones es estático (no mejora).

Fast R-CNN fue una mejora a R-CNN que consistió en modificar la CNN para que parte de su input sean las regiones propuestas. Esto permite procesar las 2000 regiones propuestas en una única evaluación de la CNN, en vez de hacerlo 2000 veces para cada región.

Esta versión incrementó la velocidad de entrenamiento y detección drásticamente (casi 10x más rápido) sin empeorar la precisión. **[12]**

Su limitación principal es que el algoritmo de selección de regiones sigue siendo estático y, ahora, es el cuello de botella de todo el proceso (lleva la mayor parte del tiempo).

Faster R-CNN introduce una nueva red neuronal al esquema llamada RPN (Region Proposal Network). Esta red utiliza la salida de la CNN para proponer las regiones, integrándose esta tarea al proceso de machine learning. Esto permite que la proposición de regiones aprenda conforme avanza el entrenamiento, y obtiene mejoras de performance notables, ya que no se requiere más un algoritmo externo de selección de regiones. **[11][38]**

Su estructura entonces es la siguiente:

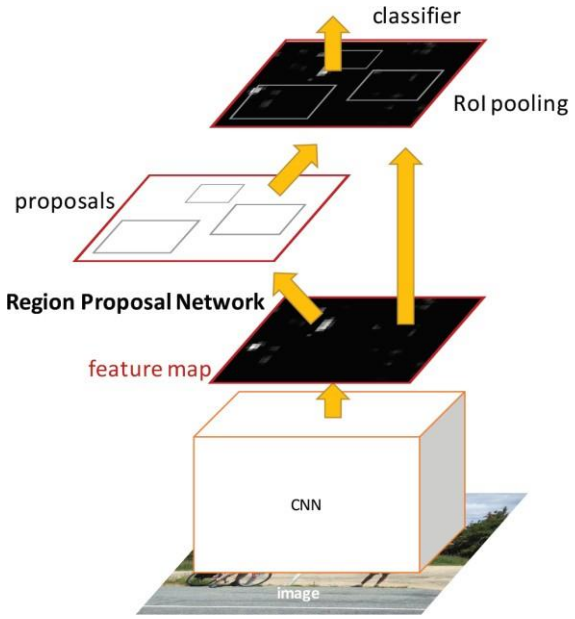
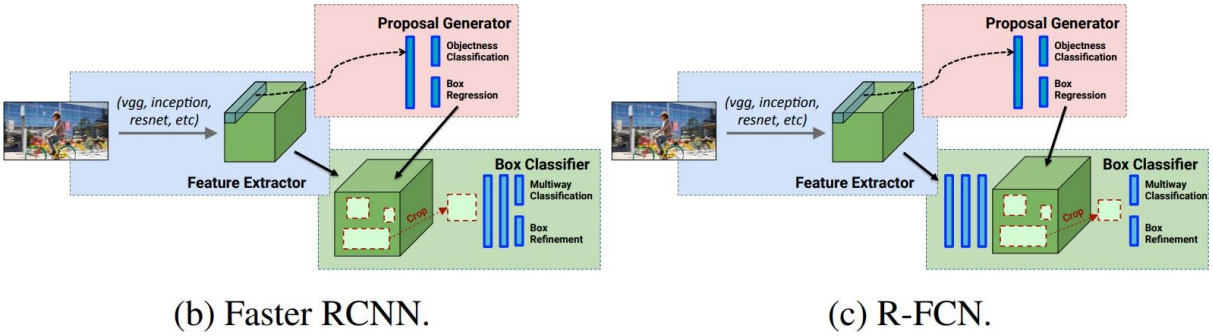


Figura 2.4-1 - Estructura Faster R-CNN – Extraído de [38].

Comparativamente con Fast R-CNN, ambos entrenados en los dataset de propósito general PASCAL VOC 2007 + PASCAL VOC 2012, su precisión es levemente superior (70.4% vs 68.4%) y su velocidad es aproximadamente 10 veces mayor. **[38]**

R-FCN

R-FCN introduce una modificación a Faster R-CNN que consiste en postergar la extracción de características a la última capa de la red previo a la predicción (a diferencia de Faster R-CNN que lo hace en la primera capa). Esto disminuye la cantidad de cálculos realizados sobre las regiones propuestas, la mayoría se realiza de forma compartida en la imagen completa. **[6][15]**



(b) Faster RCNN. (c) R-FCN.

Figura 2.4-2 - Comparativa estructural de Faster RCNN y R-FCN – Extraído de [6].

En la práctica R-FCN sacrifica un poco de precisión por un notable aumento en velocidad, como puede observarse en la siguiente tabla:

	training data	mAP (%)	test time (sec/img)
Faster R-CNN [9]	07++12	73.8	0.42
Faster R-CNN +++ [9]	07++12+COCO	**83.8**	3.36
R-FCN multi-sc train	07++12	77.6†	0.17
R-FCN multi-sc train	07++12+COCO	**82.0**‡	0.17

VOC 2012 for R-FCN

Tabla 2.4-1 - Comparativa de mAP y performance entre Faster RCNN y R-FCN – Extraído de [6].

SSD

SSD (Single Shot Detector) introduce un cambio estructural con respecto a los detectores mencionados anteriormente. En vez de utilizar dos redes, una para la extracción de características y una para la proposición de regiones, utiliza una sola. Esto se debe a que en vez de hacer proposición de regiones (mediante algún algoritmo o una RPN), define una serie de regiones por defecto con distintos tamaños y relaciones de aspecto. **[28][43]**

Durante la ejecución, la red calcula la probabilidad de ocurrencia de cada objeto para cada una de las regiones por defecto y las modifica y/o combina para mejor ajustarse a su forma.

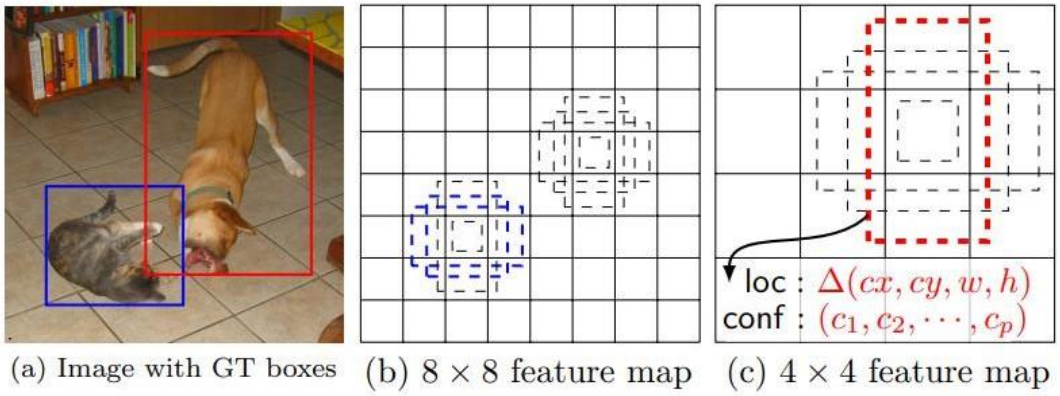


Figura 2.4-3 - Regiones por defecto utilizadas por SSD - – Extraído de [28].

Esta estrategia de detección es más liviana, y por ende más rápida de entrenar y de ejecutar. Esto conlleva a resultados notablemente mejores en materia de velocidad, manteniendo una buena precisión.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Tabla 2.4-2 - Comparativa de mAP y FPS de Faster RCNN y SSD (múltiples resoluciones) – Extraído de [28].

En comparación con Faster R-CNN, ambos entrenados en el dataset PASCAL VOC 2007, se ve una precisión levemente mejor y mucha mayor velocidad de procesamiento.

YOLOv3

YOLOv3 es la última versión de la serie de algoritmos YOLO de detección de imágenes en tiempo real. Es actualmente uno de los algoritmos de computer vision más rápidos que existen. Similar a SSD, utiliza regiones de proposición predefinidas, en vez de una red o algoritmo para generarlas.

Esta versión introduce una serie de modificaciones tendientes a mejorar la performance y precisión del algoritmo con respecto a las iteraciones anteriores:

- YOLOv3 introduce clasificadores independientes para calcular la probabilidad de que la entrada pertenezca a cada una de las clases. Esto permite realizar clasificación múltiple, ya que las clases no son mutuamente excluyentes.
- Extractor de características piramidal, que permite realizar predicciones sobre la imagen en distintas escalas, mejorando la precisión particularmente en objetos pequeños.
- Darknet-53, un extractor de características más eficiente, que mantiene la precisión con respecto a la versión anterior (Darknet-19).

En la práctica, entrenado y probado sobre el dataset COCO, YOLOv3 tiene una precisión similar a SSD, pero un tiempo de inferencia (performance) 3 veces más rápido. Su mAP (precisión) es menor que otras redes como RetinaNet, pero su velocidad sigue siendo notablemente superior. **[37]**

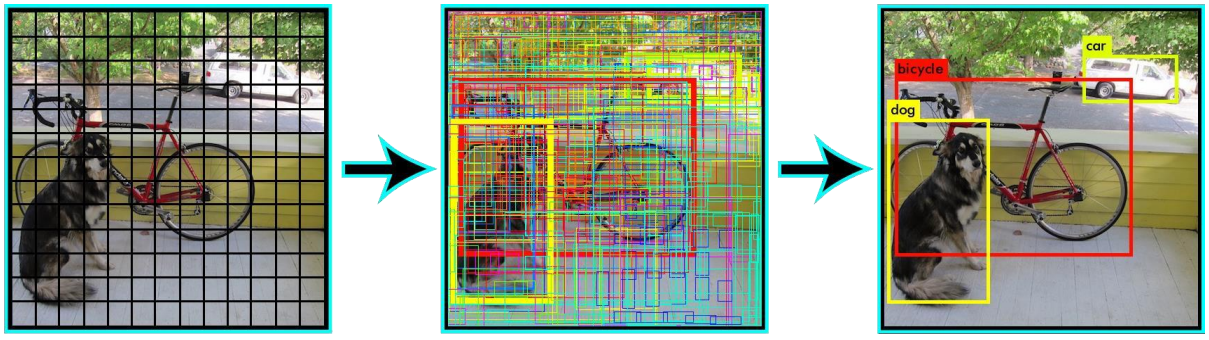


Figura 2.4-4 - Esquema del extractor de características de YOLOv3 – Extraído de [16].

RetinaNet

RetinaNet es, al igual que YOLO y SSD, un detector de única pasada. Sus características más distintivas con respecto a los otros algoritmos son:

- Invariancia de escala: Posee un extractor de características de estructura piramidal para la detección de elementos independientemente del tamaño que ocupen en la imagen general.[1]
- Focal Loss (FL, pérdida focal): Función que busca mejorar la calidad del entrenamiento, priorizando el aprendizaje en detecciones de objetos que no son "background" (fondo), es decir, en objetos clasificados incorrectamente **[42]**.

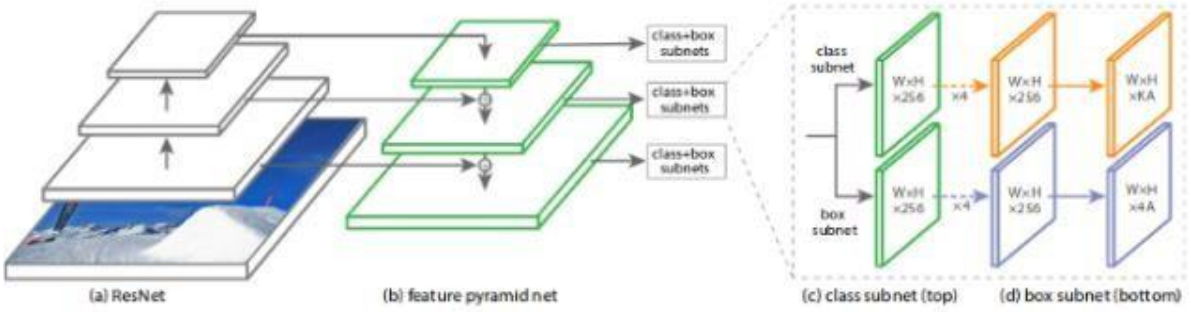


Figura 2.4-5 - Estructura de componentes de RetinaNet – Extraído de [42].

[1] Esta característica fue incluida en YOLOv3 posteriormente a la implementación de RetinaNet. YOLOv2 no posee esta característica.

Comparación de resultados

A continuación, se muestran resultados comparativos compilados de distintas fuentes (papers de cada una de las implementaciones, así como artículos que analizan y comparan las distintas herramientas **[17]**):

El siguiente gráfico compara la mAP[2] de los distintos algoritmos considerados en varias resoluciones. Todos los modelos fueron entrenados con datos de los dataset PASCAL VOC 2007 y 2012. Las pruebas fueron realizadas sobre el dataset PASCAL VOC 2012:

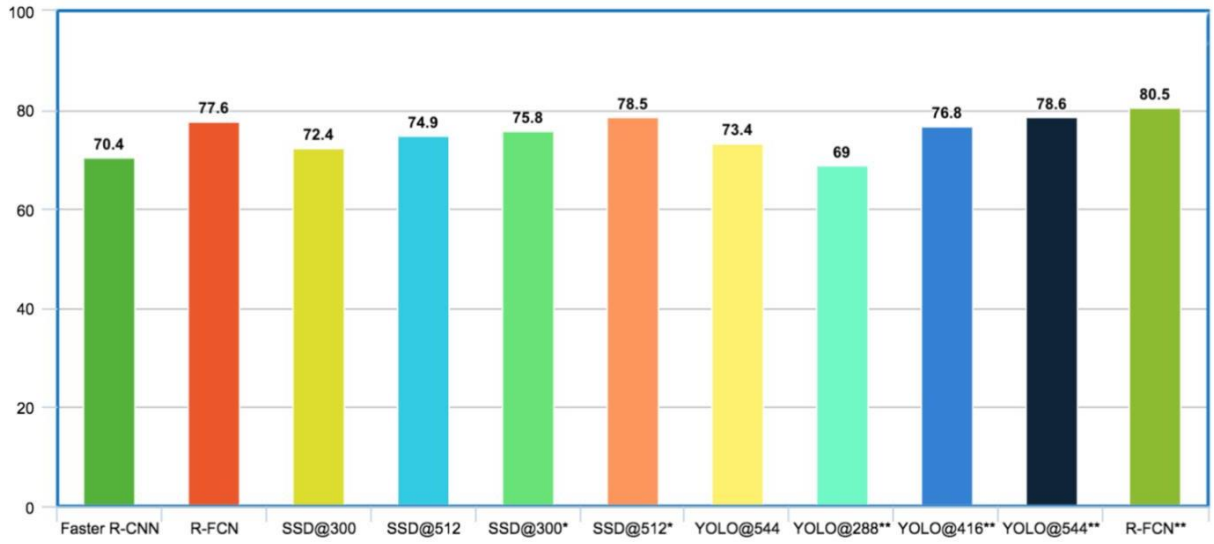


Figura 2.4-6 - Comparativa mAP de distintos algoritmos y resoluciones en PASCAL VOC 2012 – Extraído de [17].

La siguiente tabla muestra las precisiones testeadas en el dataset MS COCO:

	backbone	AP	AP_{50}	AP_{75}
Two-stage methods				
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2
One-stage methods				
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4

Figura 2.4-7 - Comparativa mAP para distintos algoritmos en MS COCO – Extraído de [17].

[2] Mean average precision - Precisión media

Con respecto a la velocidad de detección, la comparación es más compleja ya que las fuentes de las distintas implementaciones han sido probadas sobre múltiples tecnologías y con diversos intervalos de confianza.

El siguiente gráfico muestra la precisión media (mAP) adquirida sobre el tiempo de inferencia en milisegundos, según reportado por los distintos autores de cada implementación, en el dataset MS COCO:

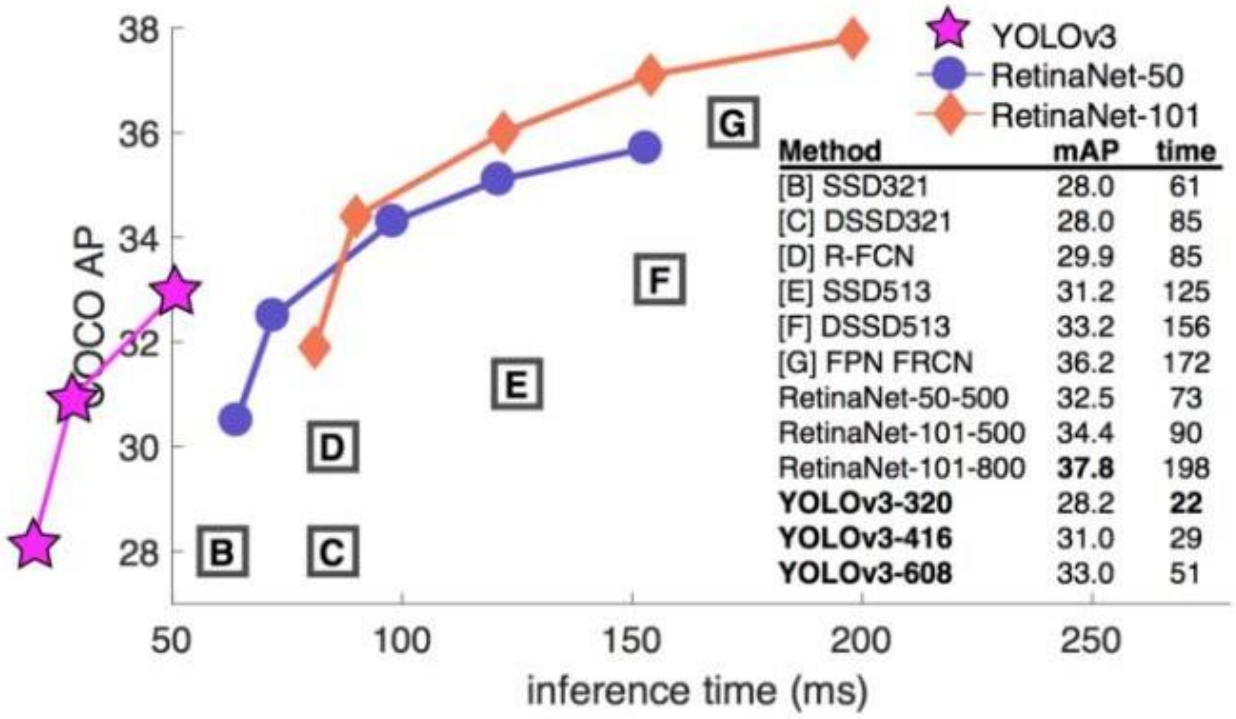


Figura 2.4-8 - Comparativa mAP vs tiempo de inferencia (ms) entre distintos algoritmos – Extraído de [17].

Estos valores deben tomarse con cuidado, ya que existen múltiples factores que pueden influir a la performance, tales como el tipo de extractor de características que se utilice, el hardware donde se hayan realizado las pruebas, el dataset y el nivel de confianza.

2.5 YoloV3

En esta sección se explicará brevemente la estructura y el funcionamiento de la red neuronal YOLOv3 para detección de objetos que fue utilizada en este proyecto.

La arquitectura de YOLOv3 es la siguiente **[46]**:

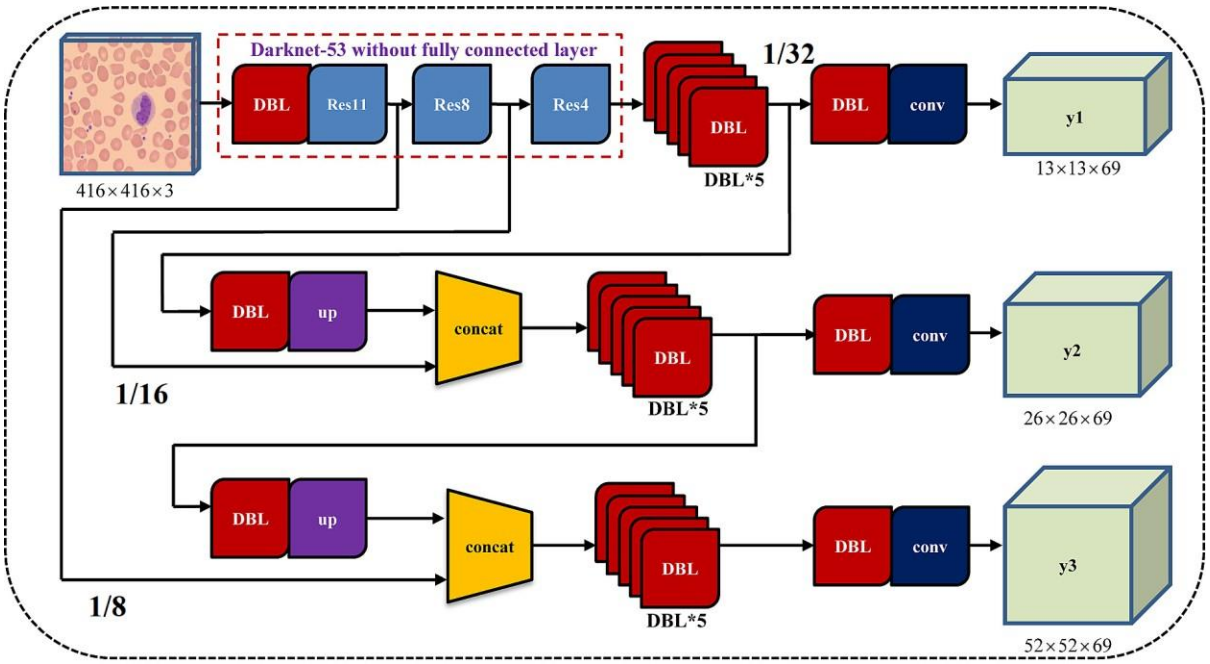


Figura 2.5-1 - Estructura de componentes de YOLOv3 – Extraído de [35].

Lo importante a identificar es el extractor de características Darknet-53 y el resto de los componentes convolucionales que componen la red.

En la imagen se puede apreciar la característica distintiva que se mencionaba en la comparación. YOLOv3, para cada imagen, genera 3 mapas de características (feature maps) en 3 distintas escalas (1/32, 1/16 y 1/8 de la imagen original), que utiliza para realizar las predicciones.

Para las escalas mayores (1/16 y 1/8) se combina la entrada a las capas convolucionales (DBL) con un upsampling **[1]** de la escala previa.

El proceso de detección comienza con la división de la imagen en celdas. La cantidad de celdas es proporcional al tamaño del input de entrada (N) y a la escala en cuestión.

Debido a que YOLOv3 realiza predicciones en 3 escalas, la extracción de características se realiza en 3 puntos de la red, y está influenciado por el factor de escala acumulado de los sucesivos downsampling **[1]** de la imagen que las capas convolucionales van aplicando. Asumiendo un input size de entrada de N=416, se tienen los siguientes mapas de características (feature maps):

- y1, con un factor de escala de downsampling de 1/32, generando el feature map de 13x13 celdas explicado anteriormente.
- y2, con un factor de escala de downsampling de 1/16 concatenado con upsampling 2x de y1, generando un feature map de 26x26 celdas.
- y3, con un factor de escala de downsampling de 1/8 concatenado con upsampling 2x de y2, generando un feature map de 52x52 celdas.

Cada celda, en cada mapa de características, se encarga de predecir una serie de cuadros delimitadores ("bounding boxes"). Para cada uno de ellos, la red predice la confianza de que el cuadro delimitador realmente encierra un objeto y la probabilidad de que el objeto encerrado sea una clase particular.

La mayoría de estos cuadros delimitadores se eliminan, porque su confianza es baja o porque encierran el mismo objeto que otro cuadro delimitador con una puntuación de confianza muy alta. Esta técnica se denomina non-maximum suppression (supresión no máxima).

Figura 2.5-2 - Demostración de Non-Max suppression en YOLOv3[3]

[3] Extraído de https://medium.com/analytics-vidhya/object-detection-using-yolov3-d48100de2ebb

Extractor de características: Darknet-53

YOLOv2 utilizaba un extractor de características llamado Darknet-19, compuesto por 19 capas convolucionales complementadas con 11 capas más para la detección de objetos. Con una arquitectura de 30 capas, YOLOv2 a menudo tenía problemas con la detección de objetos pequeños. Esto se atribuyó a la pérdida de características asociadas al downsampling de la imagen de entrada.

Darknet-53 es un nuevo extractor de características con eliminación de bloques residuales, upsampling (para detección a múltiples escalas) y saltos de conexiones (para mayor eficiencia en el upsampling).

Darknet-53 está compuesto por 53 capas convolucionales en el extractor de características, y luego se le agregan 53 capas más para el proceso de detección. YOLOv3 entonces tiene un total de 106 capas convolucionales. Las predicciones a múltiple escala se realizan en las capas 82,94 y 106 (y1, y2 e y3 respectivamente).

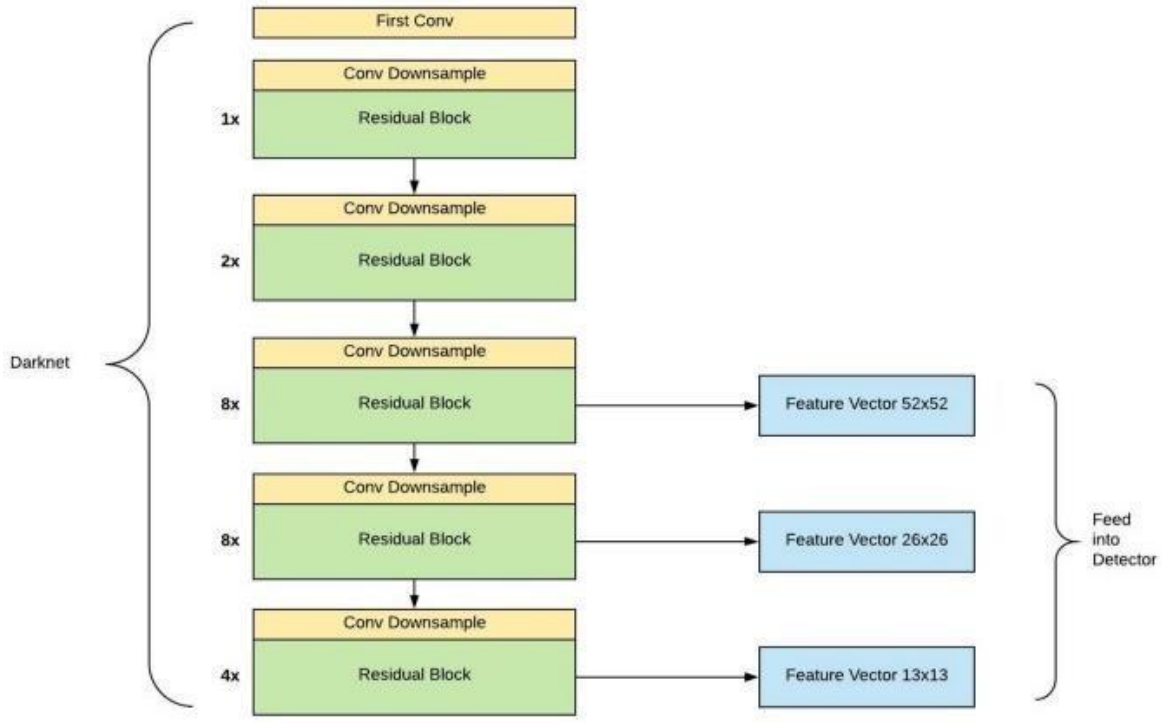


Figura 2.5-3 - Estructura del extractor de características Darknet-53[4]

[4] Extraído de https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e

Detección:

La detección se realiza aplicando núcleos de detección 1x1 a cada uno de los 3 mapas de características extraídos.

El tamaño de este núcleo es $1\ x\ 1\ x\ B\ x\ (5 + C)$, donde:

- B es el número de cuadros delimitadores que puede predecir una celda en el mapa de características. En YOLOv3 siempre B=3.
- Cada cuadro delimitador tiene (5+C) elementos.
    - 5 elementos: los primeros 4 corresponden a la posición (Xc e Yc) y el tamaño (W y H) del cuadro, acompañado de un valor que determina qué confianza existe de que en ese cuadro hay un objeto presente (Pobj)
    - C es el número de clases, un valor para cada clase que representa que tan probable es que ese objeto pertenezca a esa clase (C1 … Cn)

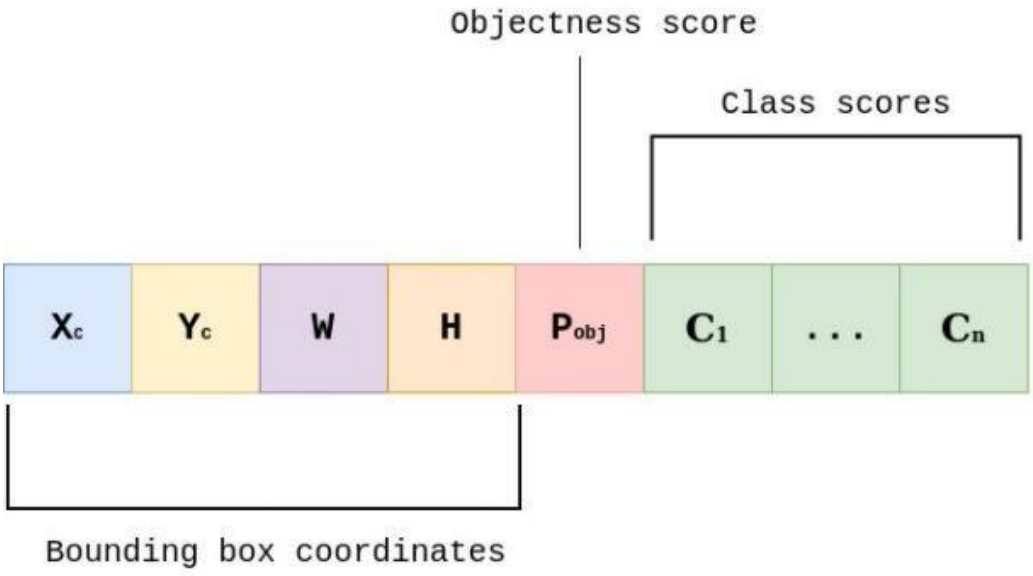


Figura 2.5-4 - Estructura de la salida del núcleo de detección 1x1 en YOLOv3 – Extraído de [39].

La primera detección la realiza la capa 82. Para las primeras 81 capas, la red muestrea la imagen hacia abajo, de modo que la capa 81 tiene una zancada de 32. Si se tiene una imagen de 416 x 416, el mapa de características resultante tendría un tamaño de 13 x 13. Una detección se hace aquí usando el kernel de detección 1 x 1, lo que nos da un mapa de características de detección de 13 x 13 x 255.

Luego, el mapa de características de la capa 79 se somete a algunas capas convolucionales antes de ser muestreado en 2x a dimensiones de 26 x 26. Este mapa de características se concatena en profundidad con el mapa de características de la capa 61. Seguido a esto, los mapas de características combinados se vuelven a someter a algunas capas convolucionales 1 x 1 para fusionar las características de la capa anterior (61). Se aplica un kernel de detección, que arroja un mapa de características de detección de 26 x 26 x 255.

Se sigue un procedimiento similar nuevamente, donde el mapa de características de la capa 91 se somete a algunas capas convolucionales antes de concatenarse en profundidad con un mapa de características de la capa 36. Como antes, siguen algunas capas convolucionales 1 x 1 para fusionar la información de la capa anterior (36). Aquí se aplica un último kernel de detección, obteniendo un mapa de características de tamaño 52 x 52 x 255.

Las detecciones en diferentes capas ayudan a abordar el problema de la detección de objetos pequeños, una queja frecuente con YOLOv2 y algo que era de suma utilidad para nuestro proyecto.

Las capas muestreadas concatenadas con las capas anteriores ayudan a preservar las características de grano fino que ayudan a detectar objetos pequeños. La capa 13 x 13 es responsable de detectar objetos grandes, mientras que la capa 52 x 52 detecta los objetos más pequeños, y la capa 26 x 26 detecta objetos medianos.

3 Metodología

3.1 Forma de trabajo

La metodología propuesta para la resolución del proyecto se dividió en dos grandes etapas: de investigación y desarrollo, de características retroalimentativas.

Se buscó realizar avances incrementales durante ambas etapas, validando el progreso con la dirección del proyecto mediante puntos de control previos al cierre de cada una de las tareas definidas.

Debido a la naturaleza investigativa del proyecto, surgieron cambios a lo largo del desarrollo del mismo que obligaron a reacomodar plazos y expectativas de cada una de las etapas y actividades planificadas inicialmente.

Se realizó una implementación de la herramienta de detección que satisface los requerimientos funcionales esperados, así como las expectativas del proyecto. Se espera que sobre dicha base se puedan integrar mejoras y/o guiar futuras implementaciones que hagan uso del dataset y la red neuronal obtenidas en este proyecto.

3.2 Etapas

Se identificaron dos grandes etapas a la hora de encarar este proyecto, la etapa de investigación y la de desarrollo:

3.2.1 Investigación

La mayor parte de esta etapa consistió en el proceso de interiorización sobre las distintas técnicas y herramientas existentes en la actualidad para la realización del proyecto.

Se investigaron cuáles eran las mejores opciones para encarar el proyecto en materia de lenguajes de implementación, algoritmos de entrenamiento, herramientas de soporte, entre otros, así como se reforzaron conocimientos teóricos ya existentes y se incorporaron nuevos que fueron de ayuda a la hora de continuar con las etapas futuras.

3.2.2 Desarrollo

La etapa de desarrollo del proyecto consistió en la conformación del dataset etiquetado de armas de fuego, el entrenamiento, configuración y selección de la red neuronal, y la implementación de la herramienta de software que hace uso de la misma.

El objetivo de esta herramienta es permitir al usuario hacer uso del modelo a través de una interfaz amigable e intuitiva.

Durante esta etapa, se buscó definir una estructura que permita a la aplicación evolucionar con el paso del tiempo para incluir funcionalidades que podrían agregarse, en un contexto no limitado por los recursos disponibles de este

Con el objetivo de agilizar y facilitar la implementación de esta herramienta, se optó por el desarrollo de una interfaz gráfica de usuario de tipo escritorio en Python. Esto se debe a que los componentes fundamentales para el funcionamiento de la red neuronal fueron implementados en este lenguaje y, si bien existe la posibilidad de utilizar algún otro framework más moderno, se cree que esta implementación satisface los requerimientos fundamentales para demostrar el proceso de detección de las armas de fuego efectivamente.

Por limitaciones de tiempo y recursos, se deja abierta la posibilidad de expandir en este aspecto del proyecto en el futuro.

3.3 Actividades

Con el objetivo de obtener una mayor granularidad para el avance del proyecto, se definieron un total de 5 actividades fundamentales, así como un conjunto de tareas atómicas para cada una de ellas, que modelan el progreso.

Para gestionar el flujo de las actividades, se establecieron puntos de control con fechas pactadas para poder realizar un seguimiento y respetar de la mejor manera posible el cronograma y el plan de trabajo.

Al cierre de cada actividad se presentaron los avances realizados, así como un análisis de los inconvenientes identificados, esfuerzo requerido y progreso general del proyecto; en conjunto con una revisión de la próxima actividad.

3.3.1 Actividad 1

"Seleccionar técnicas y herramientas que sean capaces de realizar el procesamiento automático de imágenes mediante una red neuronal entrenada."

- Relevamiento de información sobre algoritmos, técnicas y herramientas a utilizar (tales como redes neuronales convolucionales, redes LSTM, Yolo, Tensorflow, etc.).
- Análisis, evaluación y selección de las técnicas y herramientas a utilizar en el proyecto.

3.3.2 Actividad 2

"Diseñar y definir un dataset etiquetado para realizar el entrenamiento supervisado de la red neuronal, mejorando la capacidad de detección de la herramienta."

- Especificación de las clases relevantes a ser detectadas (ej: armas blancas y armas de fuego).
- Obtención de imágenes que conforman el dataset.
- Preprocesamiento de las imágenes para adecuar a ML.
- Etiquetado de las imágenes del dataset.

3.3.3 Actividad 3

“Diseño, implementación y entrenamiento de modelos de redes neuronales basados en deep learning mediante el dataset definido.”

- Diseño de la arquitectura de la red neuronal.
- División del dataset creado en dataset de entrenamiento y dataset de prueba/validación.
- Selección de estrategias de entrenamiento a utilizar y configuración de parámetros de los modelos neuronales.
- Implementación de los modelos de redes neuronales.
- Entrenamiento de los modelos con el dataset definido y selección de modelo más adecuado.

3.3.4 Actividad 4

“Validar y optimizar el modelo neuronal entrenado, mediante ajustes manuales y/o estrategias de prueba.”

- Definición de casos de prueba.
- Evaluación de la efectividad del modelo neuronal.
- Formulación de conclusiones con respecto al modelo evaluado.

3.3.5 Actividad 5

“Desarrollar una herramienta de software prototipo que utilice el modelo neuronal entrenado para la detección de armas.”

- Análisis de requerimientos y restricciones.
- Diseño de la herramienta.
- Implementación de la herramienta.

3.4 Riesgos

Durante el desarrollo de este proyecto se identificaron los siguientes riesgos:

#	**Riesgo**	**p**	**i**	**ER**	**Categoría**
1	Uno de los integrantes abandona el proyecto	1	4	4	Aceptable
2	Se realizan estimaciones inexactas respecto a la cantidad de horas requeridas para realizar el proyecto	5	2	10	Riesgoso
3	Las herramientas a utilizar para la implementación cambian/se actualizan/ se dan de baja y se requiere revisarlas	2	2	4	Aceptable
4	Uno de los miembros del grupo debe ausentarse temporalmente por razones personales	2	3	6	Poco riesgoso
5	Un gran número de solicitudes de cambio dramáticamente aumenta la complejidad del proyecto y distrae las características clave.	2	3	6	Poco riesgoso
6	El alcance definido para el proyecto no es viable	3	2	6	Poco riesgoso
7	Aspectos legales provocan que el proyecto sea inviable	1	2	2	Aceptable
8	La aplicación terminada no logra una performance que justifique su utilización	3	4	12	Riesgoso
9	No se cumplen las fechas pactadas para la entrega de los hitos	3	5	15	Muy riesgoso
10	El cliente no tiene disponibilidad para participar en los ciclos de revisión del proyecto	1	2	2	Aceptable
11	Los módulos propensos a tener errores necesitan más trabajo de comprobación, diseño e implementación	2	1	2	Aceptable
12	El trabajo con un entorno software desconocido causa problemas no previstos	3	2	6	Poco riesgoso
13	Depender de una tecnología que aún está en fase de desarrollo alarga la planificación	2	1	2	Aceptable
14	Las bibliotecas de código existentes tienen poca documentación por lo que se requiere tiempo extra para comprenderlas	4	2	8	Poco riesgoso

Tabla 3.4-1 – Riesgos identificados en el proyecto

Teniendo en cuenta que:

- **p** es una medida cualitativa de la **probabilidad de ocurrencia** del riesgo (entre 1 y 5)
- **i** es una medida cualitativa del **impacto de ocurrencia** del riesgo (entre 1 y 4)
- **ER** o **exposición al riesgo** es una métrica calculada según:

$$ER = Probabilidad * Impacto = p * i$$

Según la exposición al riesgo (ER) se identificaron las cuatro **categorías** de riesgo:

Categorías de Riesgo	**ER**
Aceptable	[1,4]
Poco riesgoso	(4,8]
Riesgoso	(8,12]
Muy riesgoso	(12,16]

Tabla 3.4-2 – Categorías de riesgos

4 Tecnologías utilizadas

4.1 Lenguajes y frameworks

4.1.1 Python

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

La solución está implementada en su totalidad en este lenguaje debido a la gran versatilidad, compatibilidad, rapidez y popularidad del mismo, particularmente en áreas de investigación y tratamiento de grandes volúmenes de datos.

Si bien no es un lenguaje con el cual estábamos familiarizados antes de comenzar el proyecto, el hecho de que casi la totalidad de la documentación en este campo de la Inteligencia Artificial, así como también las librerías para el trabajo matemático, se encuentran expresadas en este mismo, nos convenció a elegirlo.

El uso de algún otro lenguaje hubiese requerido una cantidad muy elevada de retrabajo para dar soporte a funcionalidades que ya están implementadas en este lenguaje y son de uso común en el ámbito de Computer Vision.

Teniendo en cuenta esto, se tuvo que realizar una serie de capacitaciones previas para poder manejar este lenguaje y facilitar las tareas de desarrollo. La capacitación fue exitosa, y no presentó mayores dificultades, debido a la base de conocimiento ya poseída por lo aprendido en la carrera.

4.1.2 NumPy

Es una biblioteca de funciones matemáticas de Python. Agrega soporte para vectores, matrices, algebra lineal, transformaciones, entre otros.

Esta librería es uno de los principales motivos por los cuales Python es un lenguaje exitoso en el campo investigativo y de tratamiento de datos multidimensionales.

4.1.3 Tensorflow

Librería open source desarrollada por Google para computación numérica y machine learning a gran escala. Ha sido implementada en Python y C++ **[24]**.

Tensorflow engloba un conjunto de algoritmos de Machine y Deep Learning y provee una interfaz que facilita el desarrollo sobre los mismos.

En nuestro caso se decidió utilizar esta librería debido a que facilita el proceso de trabajar con tensores.

Un tensor es una alternativa más dinámica de modelar conjuntos multidimensionales, similar a una matriz de alto nivel, pero que permite realizar operaciones que éstas no soportan.

La mayoría de las máquinas no pueden aprender sin tener datos. Y los datos modernos suelen ser multidimensionales. Los tensores juegan un papel importante en ML al codificar datos multidimensionales. Por ejemplo, una imagen generalmente se representa mediante tres campos: ancho, alto y profundidad (color). Tiene mucho sentido codificarla como un tensor 3D. Sin embargo, la mayoría de las veces se trata de decenas de miles de imágenes. Por tanto, aquí es donde entra en juego el cuarto campo, el tamaño de la muestra.

4.1.4 CUDA

Es una plataforma de computación paralela y un modelo de programación inventado por NVIDIA.

CUDA permite a los desarrolladores hacer uso directo de la capacidad de procesamiento de las tarjetas gráficas NVIDIA (GPUs), lo que aumenta drásticamente el rendimiento de algunos tipos de algoritmos de Machine Learning.

Esto se debe a que una GPU está orientada a realizar una gran cantidad de operaciones en paralelo, así como mantener un gran volumen de datos distribuidos en memoria de acceso rápido no secuencial.

Una red neuronal profunda con una gran cantidad de capas y una mayor cantidad de nodos puede aprovechar este hardware para alocar los nodos en la memoria de la GPU y leer/escribir sus valores con mayor velocidad.

Debido a la naturaleza de este proyecto, donde el análisis en tiempo real de imágenes es crucial para el objetivo del mismo, la utilización de esta plataforma nos permite aumentar la calidad de las imágenes a procesar, así como también la rapidez con la que se procesa cada imagen que compone al video.

CUDA sólo puede utilizarse sobre GPUs NVIDIA. Puntualmente en este proyecto se utilizaron una RTX 2060 6GB y una GTX 1660S 6GB.

El acceso a hardware de mayor potencia puede facilitar las tareas de entrenamiento y ejecución de la red neuronal desarrollada en este proyecto.

4.1.5 OpenCV

OpenCV (Open Source Computer Vision) es una biblioteca de software de computer vision y aprendizaje automático de código abierto. Se creó para proporcionar una infraestructura común para aplicaciones de visión por computadora y para acelerar el uso de la percepción de máquinas en los productos comerciales.

En este proyecto, fue incorporada para realizar el manejo de los distintos formatos de imágenes (cámara en vivo o archivos), para así evitar errores comunes de formato entre distintas imágenes, así como también para realizar la visualización del video analizado en la interfaz gráfica.

4.1.6 PyQt5 Designer

El diseño de las interfaces de la GUI representaba un desafío por la falta de familiarización con los frameworks de este lenguaje. Luego de una investigación se decidió utilizar PyQt5. Este framework cuenta con un útil editor de interfaces denominado PyQt5Designer.

Este editor permite crear interfaces con una metodología *"drag and drop"*. El resultado de este proceso es un archivo .UI que debe ser convertido a un archivo de Python (.py) para así poder codificar manualmente la lógica de la interfaz.

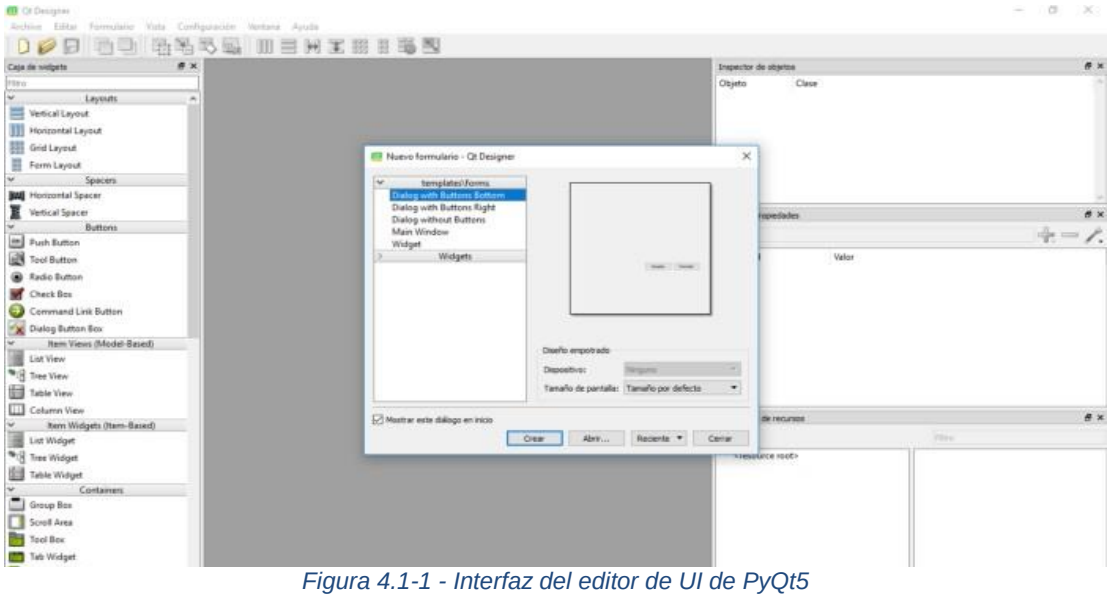


Figura 4.1-1 - Interfaz del editor de UI de PyQt5

4.1.7 Supervisely

Es una plataforma web para el desarrollo y gestión de proyectos de computer vision. Es utilizada tanto en el ámbito académico, por investigadores, como en el comercial, por empresas y desarrolladores.

Esta herramienta permite crear, etiquetar, gestionar y realizar data augmentation sobre el dataset, así como experimentar con el uso de distintas redes neuronales.

La misma fue seleccionada para la fase de investigación. Se utilizó para la confección y anotación (labeling) del dataset de armas de fuego, así como también para entrenar la red neuronal.

Una de las precondiciones impuestas para este proyecto era la capacidad de poder realizar detecciones de manera local, por lo que todos los elementos debían de ser capaces de extraerse de la plataforma en la que se habían generado, para así poder ejecutarse en una instancia local.

Supervisely permite, una vez finalizado su uso, descargar todos los resultados obtenidos (modelo entrenado, fotos etiquetadas, etc.) para su uso a discreción del desarrollador.

De esta manera, el proyecto y sus contribuciones no quedan ligadas a la plataforma en sí.

Dentro de las características más destacables de esta plataforma se pueden mencionar:

- Etiquetado de imágenes.
- Gestión de datasets.
- Información para análisis.
- Data transformation language.

4.2 Herramientas de soporte

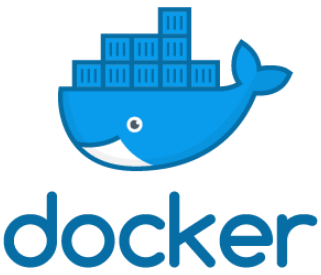
4.2.1 Git

Git es una herramienta de control de versionado. Permite realizar un seguimiento y avance ordenado de un proyecto de desarrollo de software de cualquier naturaleza.

Se establece un repositorio central desde el cual cada desarrollador puede "clonar" una copia a su máquina local para trabajar en los cambios asignados. Luego los cambios de todos los desarrolladores son subidos y mergeados en un producto final.

Se decidió utilizar los servicios de github.com para hostear el repositorio debido a que ya habíamos utilizado sus servicios en diferentes trabajos prácticos, así como también en nuestros ámbitos laborales.

4.2.2 Docker



Docker es una herramienta open source que facilita la automatización del despliegue de aplicaciones en unidades llamadas contenedores.

Un contenedor es una "unidad estándar de software que empaqueta el código y todas sus dependencias". Esto permite que el software se ejecute rápidamente y con pocas complicaciones en distintos entornos.

A diferencia de una máquina virtual, Docker corre sobre los kernel de los sistemas operativos que soporta, mejorando su performance y facilitando el acceso a componentes de hardware.

En nuestro caso, se utilizó tanto para el entrenamiento (haciendo uso del contenedor de Docker de Supervisely), así como para el desarrollo, ya que Tensorflow se ejecuta, también, a través de un contenedor de Docker.

4.2.3 Visual Studio Code

IDE (Integrated Development Environment) creado por Microsoft para el desarrollo de aplicaciones en múltiples lenguajes.

Se utilizó este IDE para el desarrollo de la herramienta, debido a la familiaridad con el mismo y a las siguientes ventajas:

- Integración directa con versionado en Git.
- Resaltado de sintaxis para múltiples lenguajes (incluyendo Python).
- Debugger incorporado.
- Cliente ligero y gratuito.

4.2.4 Google Drive

Servicio de Google para el hosteo de archivos y edición compartida de los mismos, se utilizó en gran medida ya que permite trabajar simultáneamente en tiempo real, lo que facilitó mucho la redacción de los informes.

Dentro del paquete de servicios que ofrece Drive, se utilizaron principalmente los servicios Google Docs y Google Sheets, para edición de texto y de hojas de cálculo.

4.2.5 Discord

Servicio de mensajería y voz en tiempo real para realizar llamadas, videollamadas, escritorio compartido y demás.

Fue utilizado para la coordinación interna del trabajo, particularmente en el contexto de la pandemia por el COVID-19, por la cual las reuniones presenciales se vieron suspendidas.

5 Construcción del Dataset y entrenamiento de la red neuronal

La definición general de un Dataset es un “conjunto de datos tabulados”. Esta definición puede aplicarse a datos de naturaleza muy distinta (números, textos, imágenes, mapas, etc.). En el contexto de un proyecto de Computer Vision (como el nuestro) el dataset está compuesto, generalmente, por un conjunto de pares del tipo [Imagen, etiqueta/s].

Cada etiqueta consiste de una subárea de la imagen (un recuadro) que contiene un objeto a identificar, en nuestro caso, “Handgun” (arma de fuego).

El proceso de construcción del dataset consiste entonces, en la recolección y etiquetado de las imágenes que componen al mismo. Como es de imaginar, esta tarea es laboriosa y lleva un esfuerzo temporal importante, principalmente en lo que concierne al etiquetado de las imágenes.

Existen alternativas para la construcción de Datasets para modelos puntuales, principalmente en la forma de Datasets generales (entrenados para detectar una gran cantidad de objetos) ya etiquetados, disponibles al público o propietarios de ciertas empresas, como por ejemplo el dataset “Open Images” de Google.

Sin embargo, se optó por la construcción de un dataset propio, ya que el propósito del proyecto es la detección de una clase muy específica de objetos (armas de fuego), y la naturaleza general de los datasets disponibles no coincide con los objetivos planteados.

Entrando en el campo del análisis y recolección de imágenes, para la conformación del dataset se realizó un estudio previo sobre el escenario actual de los hechos de inseguridad en nuestro país, para así poder seleccionar un conjunto de imágenes que se adapten de mejor manera al ambiente en el que se desempeñará la herramienta.

Se concluyó que el proyecto estaría orientado a la detección de armas de fuego del tipo pistolas y revólveres, de calibre pequeño/mediano, ya que éstas protagonizan, por una gran mayoría, el grueso de los sucesos violentos armados en el país.

Debido a que el buen funcionamiento de un algoritmo de Machine Learning depende fuertemente del tamaño y la calidad del dataset utilizado, es fundamental hacer énfasis en este proceso, y es por esto que gran parte del esfuerzo temporal del proyecto consistió en la recolección de imágenes y el etiquetado del dataset.

Durante el proceso de desarrollo de un algoritmo de Machine Learning se utilizan 3 datasets:

- Entrenamiento.
- Validación.
- Testing.

5.1 Recolección de imágenes

En esta sección se ahondará en los criterios que se tuvieron en cuenta a la hora de seleccionar las imágenes que componen el dataset, así como los métodos que se emplearon para conseguirlas.

5.1.1 Criterios de recolección

Para formular un dataset compuesto por un conjunto cohesivo de imágenes, que esté orientado a satisfacer los objetivos del proyecto, se tuvo que establecer un conjunto de criterios a tener en cuenta durante el proceso de recolección.

Estos criterios planteados fueron utilizados para analizar la naturaleza de la imagen. No es una lista taxativa, sino más bien orientativa:

- La resolución de la imagen debe ser no menor a HD (1280x720).

Esto se debe a que, para realizar el entrenamiento, todas las imágenes se normalizan a una resolución igual. Al expandir una imagen de resolución baja a una resolución mayor, se pierde calidad, y su utilidad disminuye.

- El arma visible en la imagen no debe ser el elemento principal de la misma.

Esto se debe a que el propósito de la herramienta no es detectar una imagen de un arma de fuego en primer plano, sino más bien en el contexto de una imagen de una cámara de seguridad, que captura un panorama más amplio, donde el arma ocupa una pequeña porción del total de la imagen.

- El arma debe ser explícita y objetivamente visible en la imagen.

Se intentó minimizar la recolección de imágenes donde el arma no se veía claramente. Se notó que en algunas imágenes, nuestra capacidad para identificar un arma en la imagen en el proceso de etiquetado era más contextual que visual, es decir, se podía suponer qué era el arma y dónde estaba por factores posturales y de comportamiento de los individuos involucrados. Esto no es objetivo, por lo cual se buscó evitar este tipo de imágenes.

- El arma debe estar siendo sujetada por una persona.

Nuevamente, se priorizó la búsqueda de imágenes donde el arma esté siendo utilizada por una persona, ya que esto representa el verdadero riesgo de una situación violenta, y no así el arma en sí misma.

5.1.2 Métodos de recolección

Se utilizaron principalmente dos métodos de recolección de imágenes para la composición del dataset:

- Descargas masivas de resultados de buscadores de imágenes.

Haciendo uso de buscadores de imágenes (principalmente Google Imágenes) se realizaron búsquedas asociadas al tipo de imágenes requeridas. Luego, con la ayuda de extensiones de navegador, se descargaron todos los resultados de la búsqueda. Estos resultados luego fueron filtrados, teniendo en cuenta los criterios mencionados anteriormente.

- Capturas de pantallas de videos.

De dos maneras distintas:

  - Automáticamente: Con videos descargados, se utilizaron herramientas que permiten extraer un cuadro del video como imagen cada cierta cantidad de tiempo.
  - Manualmente: Sobre videos reproducidos por streaming (principalmente en YouTube), se realizaron capturas de imágenes manuales.

5.2 Etiquetado de imágenes

El etiquetado del dataset es un proceso que consiste en que un usuario humano analice el dato y le asigne una característica distintiva. En el contexto de nuestro proyecto, y de Computer Vision en general, el proceso de etiquetado consiste en la definición de al menos un "Bounding Box" o cuadro delimitador (de ahora en más BBox) en una imagen.

Cada BBox representa una instancia del objeto al cual la misma está asociada. Para nuestro proyecto, el único objeto definido es "Handgun".

Cabe destacar que la necesidad de etiquetar imágenes surge de la naturaleza supervisada del tipo de entrenamiento elegido. Existen otros algoritmos que no necesitan etiquetado.

Figura 5.2-1 - Ejemplo BBox de arma

5.2.1 Método de etiquetado

El etiquetado de imágenes en este proyecto fue realizado de manera manual, mediante la herramienta de etiquetado de Supervisely, la cual facilita varios aspectos de este proceso, ya que permite:

- Definir cada BBox de manera gráfica.
- Filtrar las imágenes que han sido etiquetadas de las que no.
- Obtener estadísticas sobre qué porcentaje del dataset está etiquetado.
- Guardar las etiquetas realizadas en tiempo real.

5.2.2 Datasets de training, validation y testing

Una vez que el dataset se construyó completamente y fue etiquetado, se procedió a realizar las divisiones correspondientes:

- Dataset de testing **[49]**

Se definió un dataset de testing compuesto por aproximadamente 300 imágenes. Este dataset no es utilizado de ninguna forma en el proceso de entrenamiento, y es excluido para ejecutarse una vez que la red ya está entrenada. Este método de separación se conoce como "Holdout method".

- Datasets de training y validation **[48]**

El dataset restante se divide en dos: el dataset de training (entrenamiento), que es utilizado durante el entrenamiento para ajustar los pesos de la red, y el dataset de validation (validación), que es utilizado también durante el entrenamiento, pero para ajustar hiperparámetros[5] de la red.

Este proceso es automatizado mediante el uso del DTL de Supervisely. Se ejecuta un script que asigna un "tag" (etiqueta) a la imagen según corresponda: "train" para las imágenes que pertenecen al dataset de entrenamiento, y "val" para las del dataset de validación.

Se definió una proporción de distribución de 90/10 en entrenamiento y validación respectivamente. El proceso de selección de qué imagen termina en qué dataset es aleatorio, para evitar introducir cualquier tipo de sesgo.

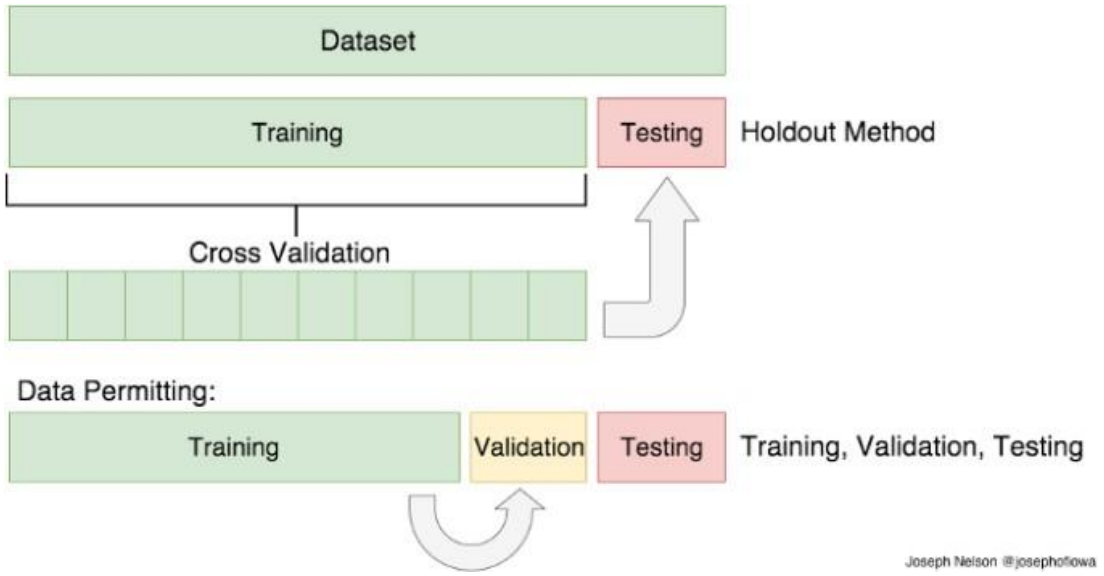


Figura 5.2-2 - Esquema de division de dataset en train, val y testing – Extraído de [26].

[5] Los hiperparámetros de la red son valores ajustables adicionales a los pesos de cada nodo. Los más comunes son la topología de la red, la cantidad de niveles, el tamaño del lote de procesamiento, entre otros.

5.3 Gestión de Dataset

El uso de la herramienta Supervisely, por su naturaleza como entorno de desarrollo de soluciones de Computer Vision, otorga un paquete de funcionalidades tendientes a facilitar el manejo de datasets de gran tamaño y con diversidad de elementos, así como también herramientas de gestión de proyecto y equipos de trabajo.

La principal ventaja que nos otorgó el uso de esta herramienta es su naturaleza completamente en línea: se define un espacio de trabajo ("workspace"), donde un grupo de usuarios (en este caso, los autores) importan las imágenes que constituyen el dataset en lotes de imágenes:

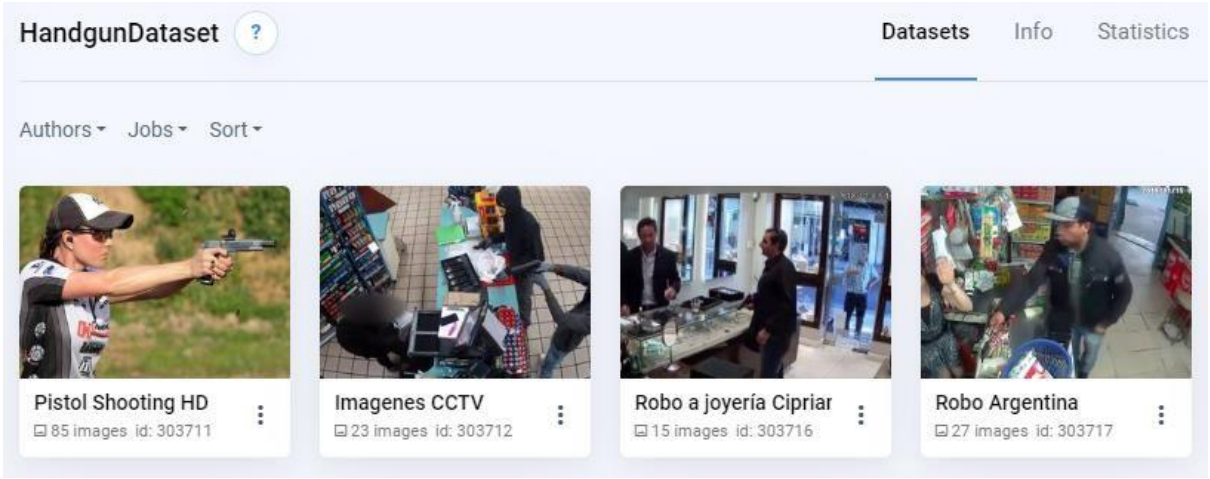


Figura 5.3-1 - Colecciones de imágenes importadas al dataset de Supervisely.

Sobre cada uno de estos lotes de imágenes se obtiene información estadística relevante sobre el progreso en las tareas de etiquetado: cuántas imágenes han sido etiquetadas, cuántas no, que cantidad de objetos se han etiquetado por lote, entre otros:

	Pistol Shooting HD	Imagenes CCTV	Robo a joyería Cipriano	Robo Argentina	Un robo en arger primera persona	Total
Handgun	85 (100.00%)	23 (100.00%)	15 (100.00%)	27 (100.00%)	10 (100.00	3651 (100.00%)
Partially / completely marked	85 (100.00%)	23 (100.00%)	15 (100.00%)	27 (100.00%)	10 (100.00	3651 (100.00%)
Image not marked	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%	0 (0.00%)
Images in dataset	85	23	15	27	10	3651

Figura 5.3-2 - Estadísticas de etiquetado sobre el dataset en Supervisely.

Los lotes o grupos de imágenes pueden ser asignados a distintos usuarios, los cuales pueden cumplir roles de etiquetador, gestor de proyectos, entre otros.

El proceso de etiquetado también es realizado mediante la herramienta de etiquetado en línea de Supervisely - “Supervisely Annotation Tool”:

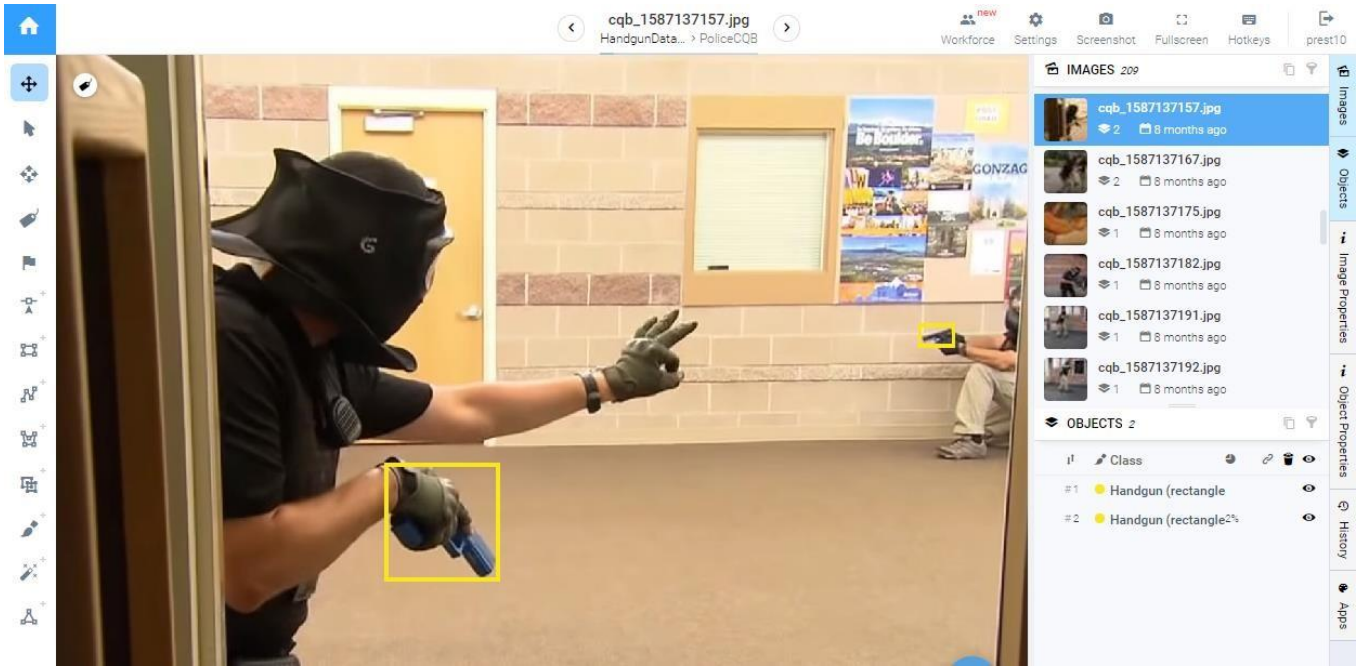


Figura 5.3-3 - Captura de la herramienta de etiquetado de imágenes de Supervisely.

La misma otorga una interfaz gráfica completa para el etiquetado de objetos en la imagen. Las etiquetas se guardan y sincronizan a medida que se generan. Ofrece funcionalidades como atajos de teclado, filtros, historial de cambios, entre otros.

Supervisely, entonces, integra completamente el manejo del dataset de forma online, desde el momento en que las imágenes se suben, hasta que se obtiene un dataset apto para alimentar una red de computer vision.

Esto es muy ventajoso, ya que no requiere de la instalación de ninguna herramienta en entornos de trabajo local, así como tampoco es necesario hacer tareas de versionado para el proceso de etiquetado.

Cabe destacar que el dataset generado (imágenes + etiquetas) puede ser exportado en múltiples formatos de uso común (JSON, XML, etc.), por lo que no se genera una dependencia con respecto a la herramienta.

5.4 Entrenamiento y configuración de la red neuronal

El proceso de entrenamiento se realizó mediante Supervisely. Debido a que para entrenar una red neuronal es necesario el uso de una tarjeta gráfica (GPU), y Supervisely no provee gratuitamente este servicio, se definió un clúster local.

El clúster local es una computadora personal perteneciente a uno de los autores, con una tarjeta gráfica NVIDIA, con soporte para CUDA, corriendo un servicio de Docker provisto por Supervisely, sobre Ubuntu 16.04.

Levantar el clúster consiste en instalar una serie de dependencias de acceso a la GPU, ejecutar el servicio de Docker y mantener una conexión a internet activa. Luego, desde la interfaz web de Supervisely, se accede a los recursos de hardware del clúster para realizar el entrenamiento de manera local.

Esto permitió, manteniendo el clúster disponible, realizar corridas de entrenamiento simplemente accediendo a la plataforma web, así como también facilitó enormemente el trabajo en simultáneo y evitó complejidades que hubiesen surgido al realizar estas tareas fuera de un contenedor de Docker y sin una interfaz gráfica dedicada.

5.4.1 Método de corridas de entrenamiento

Cada corrida de entrenamiento consiste en la selección de los elementos del dataset a considerarse, la configuración de distintos parámetros como “batch size” (tamaño de lote), “input size” (tamaño de entrada de imágen), “learning rate” (tasa de aprendizaje) y cantidad de épocas, entre otros:

```
{
  "lr": 0.0001,
  "epochs": 5,
  "batch_size": {
    "train": 8
  },
  "input_size": {
    "width": 416,
    "height": 416
  },
  "bn_momentum": 0.01,
  "gpu_devices": [
    0
  ],
  "data_workers": {
    "val": 1,
    "train": 2
  },
  "dataset_tags": {
    "train": "train"
  },
  "subdivisions": {
    "val": 1,
    "train": 1
  },
  "checkpoint_every": 1,
  "print_every_iter": 5,
  "recompute_anchors": false,
  "weights_init_type": "transfer_learning",
  "enable_augmentations": false
```

Figura 5.4-1 - Configuración de parámetros de corrida en Supervisely

Luego, el entrenamiento es realizado en el clúster local, y mediante la interfaz web se pueden observar distintas métricas que indican el avance del proceso de entrenamiento:

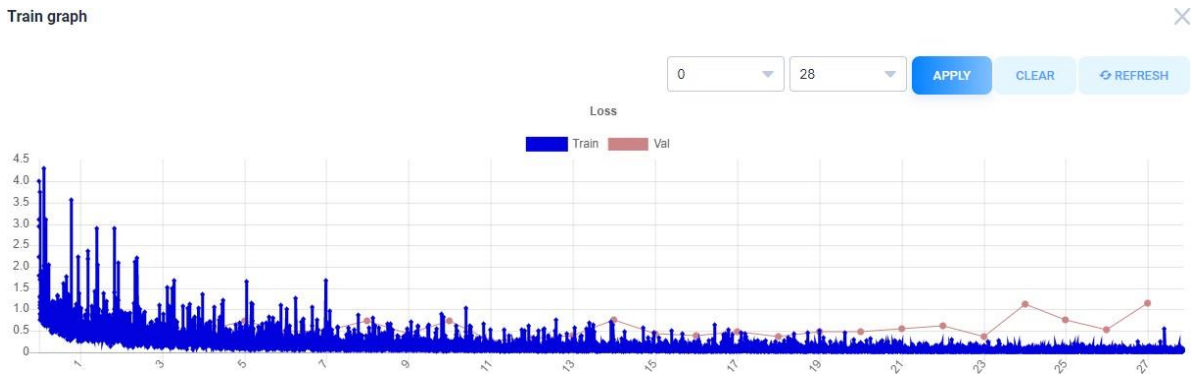


Figura 5.4-2 - Gráfico de "Loss" generado por Supervisely.

Una vez finalizado el entrenamiento, se genera un "checkpoint" o punto de control para cada una de las épocas en las que se entrenó. Se puede, entonces, exportar de alguna de ellas la red neuronal entrenada en ese estado:

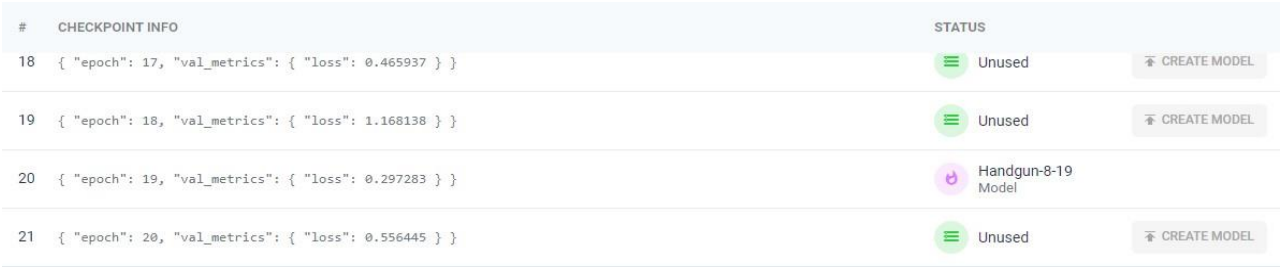


Figura 5.4-3 - Listado de checkpoints generados durante la corrida de entrenamiento en Supervisely.

En la imagen se observa que se exportó el modelo Handgun-8-19, donde 8 corresponde a la 8va corrida de entrenamiento, y 19 a la época dentro de dicha corrida.

Para cada una de las múltiples corridas, se generaron entre 1 y 3 modelos que tengan valores de "loss" (pérdida) bajos.

Supervisely luego agrupa estos modelos o redes neuronales entrenadas en una sección conjunta, donde se puede fácilmente acceder a ellas para re-entrenar, testearlas con otro dataset y ver sus parámetros de configuración:

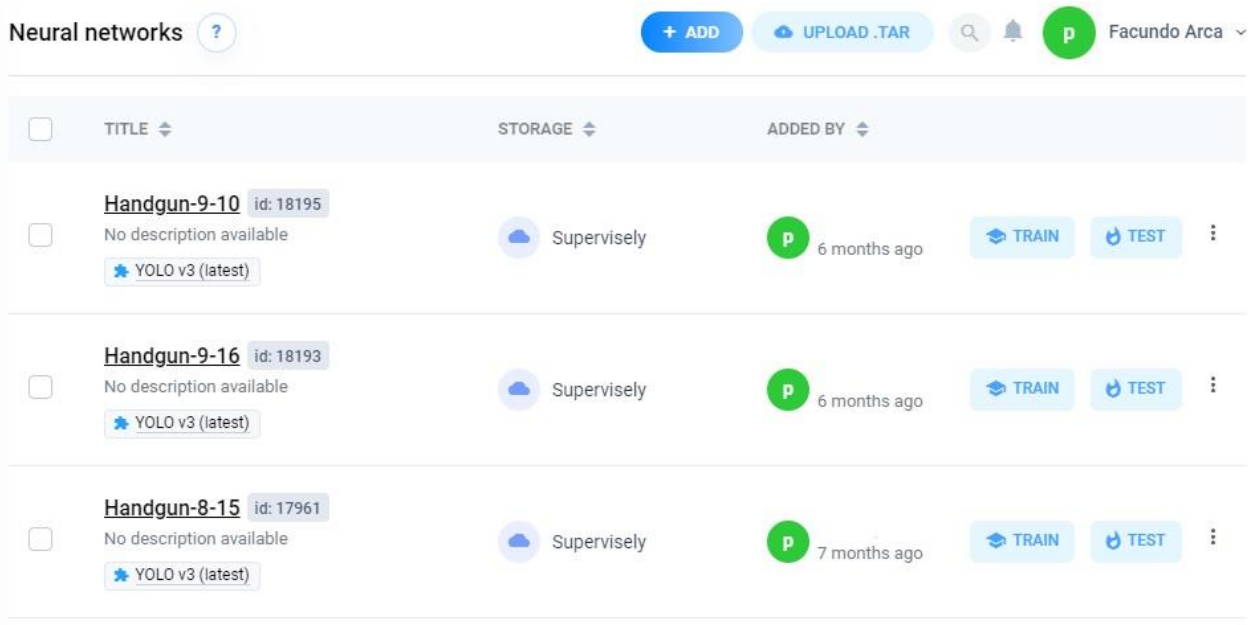


Figura 5.4-4 - Listado de redes neuronales generadas en Supervisely.

5.4.2 DTL & Data augmentation

DTL (Data transformation language - "Lenguaje de transformación de datos") es un lenguaje interno provisto por Supervisely para estandarizar el proceso de realizar un amplio espectro de operaciones sobre datasets y sus respectivas imágenes.

Este lenguaje permite correr todo tipo de scripts sobre los datasets generados. Para este proyecto, se utilizó principalmente para realizar la división en datasets de training y validation mediante etiquetado aleatorio, y para aplicar técnicas de **data augmentation.**

Data augmentation, en Machine Learning, consiste en el conjunto de técnicas aplicadas sobre un dataset para aumentar la cantidad y diversidad de información que puede aportar al proceso de entrenamiento, mediante transformaciones sobre los objetos del mismo **[14]**.

En resumen, agrega elementos al dataset derivados de los que ya existen en él.

Las "transformaciones" que se pueden aplicar a los elementos son muchas, el lenguaje DTL de Supervisely ofrece una amplia variedad para aplicar sobre imágenes:

rotaciones, desenfoque, cambios de contraste/brillo/colores, recortar, rasterizar, invertir sobre ambos ejes, entre otros.

En computer vision, la técnica de data augmentation se convirtió en un estándar de regularización, y también para mejorar la performance y combatir el overfitting en redes convolucionales.

DTL define un lenguaje de scripting unificado orientado a capas de datos y de transformación. Las capas de datos son las que permiten realizar operaciones de lectura, asignación de variables, guardado, exportación sobre los datasets, mientras que las capas de transformación aplican data augmentation sobre los mismos.

Se define mediante un archivo JSON compuesto por una lista de elementos, donde cada elemento corresponde a una capa (de datos o transformación). Cada elemento contiene una acción (action), un origen (src), un destino (dst) y configuraciones (settings).

```
{
  "action": "flip",
  "src": [
    "$data"
  ],
  "dst": "$data_flip_V",
  "settings": {
    "axis": "vertical"
  }
},
```

Figura 5.4-5 - Ejemplo capa DTL en JSON

Respetando sus orígenes y destinos, estos elementos se concatenan formando un flujo direccional de operaciones a aplicar sobre el dataset.

En este proyecto, las capas de transformación que se aplicaron al dataset fueron las siguientes:

- Espejado sobre el eje vertical:

Efecto de espejo, el resultado obtenido permite visualizar la misma arma en dos posiciones diferentes.

Se eliminó la posibilidad de realizar rotaciones en el eje horizontal, ya que el resultado es un arma con el cañón hacia abajo y cargador hacia arriba lo cual no es una situación representativa de la realidad, lo que añadiría un factor de error.

Figura 5.4-6 - Ejemplo de rotaciones sobre ejes

- Rotaciones a 90° y 270°:

Figura 5.4-7 - Ejemplo de rotaciones en 90° y 270°

Se incorpora así una mayor cantidad de imágenes de armas en posiciones verticales, ya que el grueso del dataset está compuesto por armas en posiciones horizontales.

El resultado de las capas de rotación y espejado es una expansión del conjunto inicial de imágenes, aumentando el tamaño del dataset.

- Resize:

Las imágenes se convierten a un tamaño de 704 x 704, que coincide con el input size de la red en YOLOv3. Se optó por tomar una resolución más grande de la sugerida porque se busca detectar objetos relativamente pequeños, y una mayor resolución mejora la precisión de la herramienta en estos casos.

Se eligió esta resolución puntualmente porque se acerca a valores de resolución HD (720p), que se asume como valor estándar para las fuentes de videos de cámaras de seguridad.

Valores por encima de éste hubiesen implicado una red más precisa, pero muy condicionada a entrenarse y ejecutarse en hardware de muy elevados costos para mantener tiempos de inferencia aceptables, dados los objetivos de este proyecto.

- Sliding window:

Esta capa (ventana deslizante) se utiliza para recortar parte de la imagen con sus anotaciones, deslizando la ventana de izquierda a derecha, de arriba a abajo.

Esto es muy útil durante la detección de objetos que representan un porcentaje muy pequeño del total de la imagen, como es nuestro caso.

Aquí se establece el tamaño de la ventana a deslizar y el mínimo tamaño de superposición que existirá entre cada ventana.

El resultado de esto es un conjunto de imágenes en las cuales el arma es el objeto principal de las mismas y se la puede observar en su totalidad o con partes cortadas.

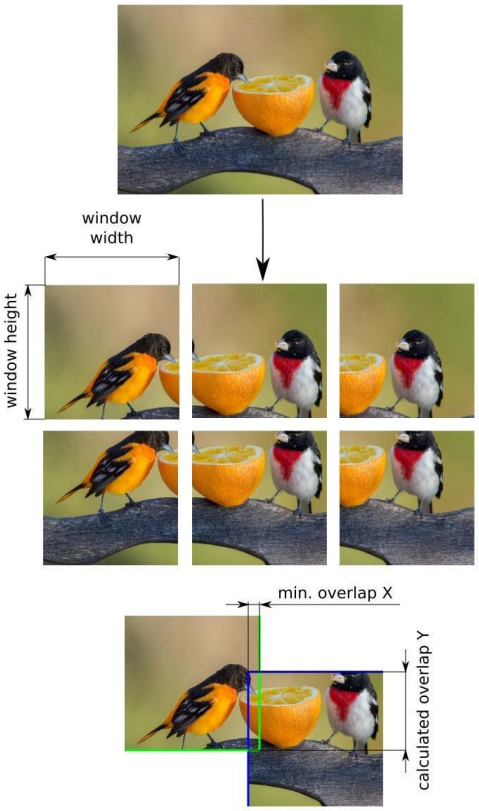


Figura 5.4-8 - Ejemplo de aplicación de sliding window – Extraído de [50].

Adicionalmente, se incluyen dos capas del tipo IF que permiten evaluar distribuciones probabilísticas para asignar los elementos al dataset de training y validation (se optó por una relación de 90/10), así como también para evaluar si se debe descartar una imagen que no tenga elementos luego de aplicar sliding window.

La aplicación de este DTL aumenta considerablemente la cantidad y variedad de imágenes disponibles a la hora de entrenar. En el anexo se incluye la definición en formato JSON del DTL utilizado.

El siguiente diagrama muestra una versión gráfica del DTL aplicado al dataset de armas de fuego:

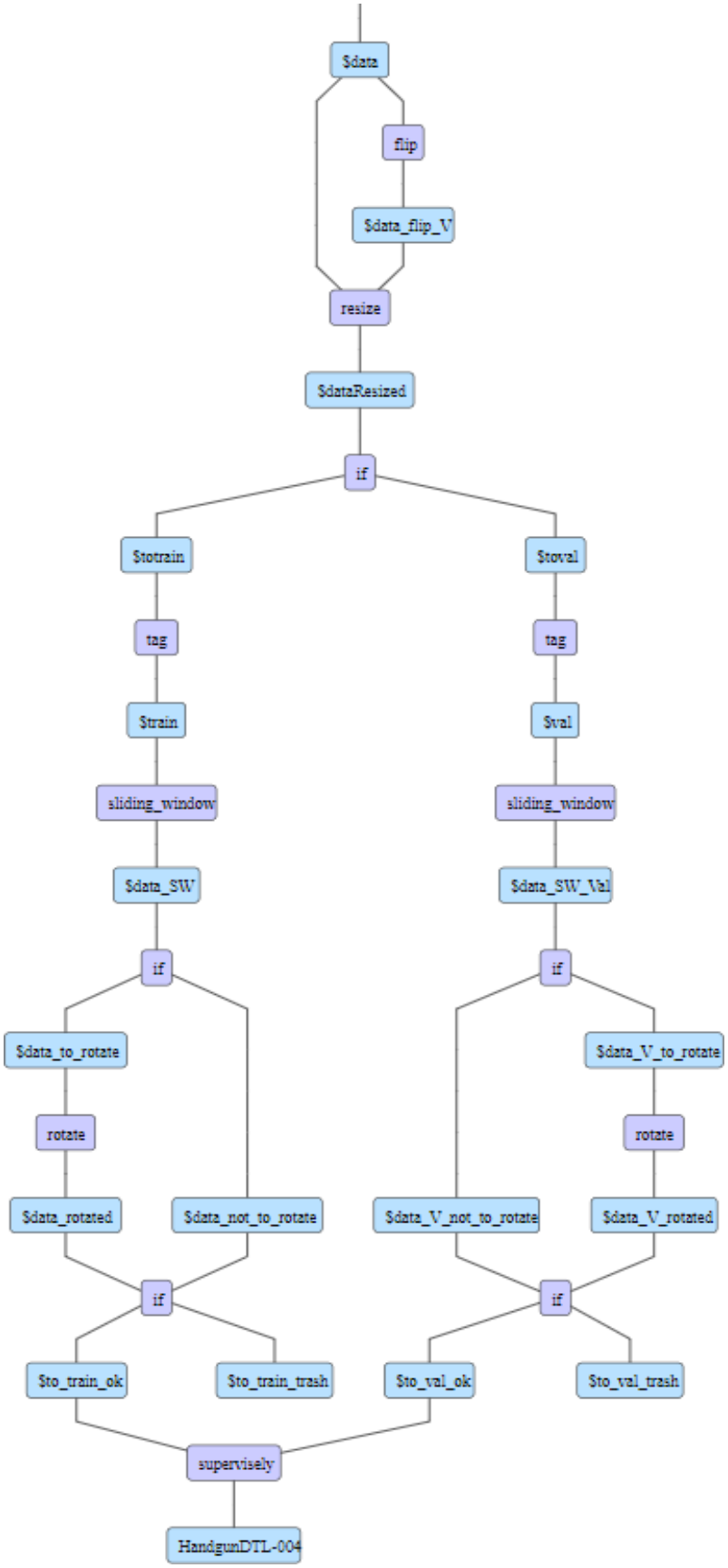


Figura 5.4-9 - Diagrama de flujo del DTL utilizado en Supervisely.

5.4.3 Configuración de parámetros de la red

Para cada corrida de entrenamiento, se modificaron distintos parámetros de la red, para evaluar qué impacto tenían en el resultado final. Para cada una de las distintas corridas se modificó el tamaño de lote (batch size) y subdivisiones de entrenamiento.

El tamaño de lote o batch size representa la cantidad de imágenes que la red observa antes de realizar un ajuste en sus pesos.

Las subdivisiones particionan el lote en mini-lotes de procesamiento simultáneo, su utilidad es evitar que la GPU se quede sin memoria intentando procesar demasiadas imágenes en paralelo.

Se tomaron batch sizes de 1, 2, 4 y 8, y valores de subdivisiones acorde a la capacidad de las GPU disponibles.

Se optó por mantener el learning rate (tasa de aprendizaje) constante a lo largo del proceso de entrenamiento, así como el input size de entrada en 704 x 704, y entrenar un total de 50 épocas por corrida.

5.5 Selección

Luego de haber generado las redes neuronales resultantes en el proceso de entrenamiento, se tuvo que elegir una en particular para exportar e integrar a la herramienta de detección.

Para ello se hizo uso del dataset de testing definido anteriormente: Se corrieron inferencias en el mismo haciendo uso de los múltiples modelos generados, y se comparó su capacidad de detección sobre el mismo.

5.5.1 Comparación de redes resultantes

En la siguiente tabla se resumen los valores calculados para cada uno de los modelos generados. Se incluyen a su vez los parámetros de configuración de batch size y subdivisiones de entrenamiento que se utilizaron en cada una de las corridas de entrenamiento.

Luego de realizar las inferencias sobre el dataset de testing, para cada uno de los modelos probados se realizó un análisis manual mediante los cuales se identificó la cantidad de falsos positivos y falsos negativos. Con dichos valores se calcularon las métricas restantes:

Modelo	Batch Size	Sub-divisions	# Detecciones	# Falsos positivos	# Positivos verdaderos	% Positivos verdaderos / detecciones	% Positivos verdaderos / total	mAP [%] @ IoU 0.5
H-9-16	4	2	144	6	138	95,83	44,66	34,42
H-9-10	4	2	155	9	146	94,19	47,25	34,30
H-8-19	4	2	135	5	130	96,30	42,07	34,33
H-8-15	4	2	137	7	130	94,89	42,07	34,59
H-8-11	4	2	152	12	140	92,11	45,31	33,29
H-8-8	4	2	146	9	137	93,84	44,34	32,76
H-7-26	8	4	145	7	138	95,17	44,66	33,42
H-7-23	8	4	164	9	155	94,51	50,16	40,65
H-7-22	8	4	152	8	144	94,74	46,60	33,53
H-6-14	4	2	151	7	144	95,36	46,60	33,78
H-6-11	4	2	143	8	135	94,41	43,69	33,19
H-5	2	1	72	6	66	91,67	21,36	16,71
H-3-23	2	1	98	5	93	94,90	30,10	23,19
H-2	2	1	131	9	122	93,13	39,48	24,67

Tabla 5.5-1 - Tabla comparativa de las redes generadas en Supervisely.

Adicionalmente, para cada una de las inferencias obtenidas, se calcula el mAP [%] con un IoU de 0.5, mediante el uso del plug-in mAP de Supervisely.

Suponiendo que A es el cuadro del arma en el dataset etiquetado, y que B es el cuadro del arma generado en la inferencia, este cálculo se realiza evaluando el cociente del área de la **intersección A ∩ B** sobre el área de la **unión A ∪ B**.

Cuando este cociente se encuentra por encima de un valor umbral (en este caso 0.5) se considera que la predicción es acertada, caso contrario, no lo es.

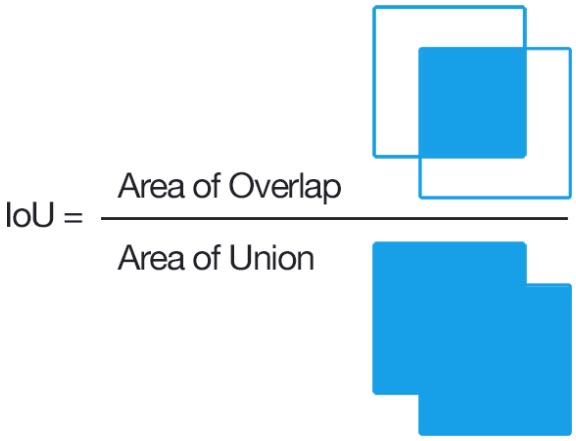


Figura 5.5-1 - Representación gráfica IoU

5.5.2 Criterio de selección

La selección del modelo a utilizar se reduce al modelo que mayor precisión presente, debido a que la velocidad de procesamiento es idéntica en todos, por ser distintas variaciones de la misma red neuronal (YOLOv3 @ 704 x 704 px).

Un fuerte indicador es la métrica mAP [%] @ IoU 0.5 calculada, ya que es un estándar comúnmente aceptado para medir la eficacia y precisión de un modelo de computer vision.

Esta métrica indica claramente un candidato puntual, **el modelo H-7-23**, con un mAP de 40,65%, bastante por encima de los demás modelos.

Este valor es aceptable y consistente con los números que se suelen observar en otras inferencias realizadas con modelos en YOLOv3, con inputs size similares. Por ejemplo, YOLOv3 @ 608 x 608 px en el dataset MS COCO obtiene un mAP de 33,0%.

Adicionalmente, se tuvieron en cuenta los valores calculados manualmente y, debido a que éstos coinciden en que este modelo es el óptimo, se lo selecciona como el adecuado para utilizar en la herramienta de software implementada.

Este modelo es luego exportado desde Supervisely, respetando el formato estándar de YOLOv3, para ser utilizado en cualquier contexto, de manera local o remota. De esta manera, el modelo no queda sujeto a Supervisely en sí.

El modelo exportado se compone de un archivo .cfg que contiene información de la estructura de la red, un archivo .weights que contiene los pesos de cada nodo de la red y un archivo config.json con los parámetros de entrenamiento:

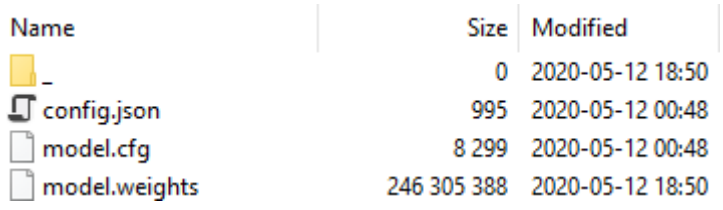


Figura 5.5-2 - Archivos generados en la exportación de la red neuronal.

6 Herramienta de detección de armas de fuego

En esta sección se describe el análisis y diseño de la herramienta de software prototipo la cual emplea la red neuronal. La misma está orientada a detectar la presencia de armas de fuego en un contexto de hechos de inseguridad, y puede ser usada en variados escenarios y situaciones con el objetivo de mejorar la seguridad ciudadana.

La metodología empleada para el desarrollo fue Ágil, debido a:

- El tamaño del equipo de desarrollo.
- Los tiempos disponibles.
- La prioridad del proyecto era una entrega pronta.

6.1 Requerimientos

El análisis de requerimientos tiene dos objetivos: encontrar los verdaderos requisitos, aquellos que cuando se implementen añadirán el valor esperado por los usuarios, y representarlos de un modo adecuado para los usuarios, clientes y desarrolladores.

6.1.1 Requerimientos funcionales (RF)

Los requisitos funcionales son declaraciones de los servicios que se espera que preste el sistema, en la forma en que reacciona a determinadas entradas (inputs de usuarios, interacciones con otros sistemas, respuestas automáticas, procesos predefinidos, etc).

- El sistema deberá ser capaz de detectar la presencia de armas de fuego en videos.
- Si se utilizan archivos de video, el sistema debe contar con una interfaz desde la cual se seleccionará el video a procesar.
- El sistema debe contar con una forma de introducir el valor mínimo de confianza con el cual se realizarán las detecciones (valores de confianza menores aumentarán el número de detecciones, pero puede aumentar consiguientemente el número de falsos positivos).
- Ante la posible presencia de un arma, el sistema deberá mostrar una ventana de alerta emergente al usuario, donde al mismo se le presentará una imagen con una detección y en la cual deberá clickear “SI” si hay un arma presente, o “NO” si no la hay.

- De haberse seleccionado “SI” en la ventana emergente, el sistema guardará la imagen y dará la voz de alarma.
- De haberse seleccionado “NO”, el sistema continuará con la ejecución, sin tomar acciones.

6.1.2 Requerimientos no funcionales (RNF)

Se trata de requisitos que no se refieren directamente a las funciones específicas suministradas por el sistema (características de usuario), sino a las propiedades del sistema: rendimiento, seguridad, disponibilidad. Si los RF definen “qué” hace el sistema, los RNF definen “cómo” lo hace.

- Para minimizar la dependencia a plataformas externas el sistema debe operar de manera completamente independiente
- Debido a los recursos disponibles, el alcance del proyecto se ha limitado a detectar armas de fuego de mano, pero el sistema debe poder ser expandido para detectar un rango mayor de elementos.
- Debido a la naturaleza de la información procesada, se debe minimizar la información compartida, por lo que el sistema debe correr de manera local y sin conexión a internet.
- El sistema debe poseer un alto grado de confiabilidad, debido a la naturaleza de lo que se desea detectar, se debe minimizar el número de falsos negativos (porcentaje de armas presentes en video no detectadas por el sistema).
- El sistema debe ser lo suficientemente intuitivo para ser utilizado por un usuario final que no posee conocimientos técnicos.
- Para evitar violar la privacidad de las personas registradas, se debe evitar persistir cualquier registro si no ocurre un hecho de inseguridad.
- El desarrollo de la herramienta debe ser modular, teniendo en cuenta que la misma podrá evolucionar y/o integrarse dentro de otros productos de software en proyectos futuros.

6.2 Diseño de solución

En esta sección se explica el proceso de definición de la arquitectura, componentes, interfaces y otras características del sistema resultante.

Al momento de comenzar el diseño de la herramienta que emplearía la red neuronal, se realizó una investigación de implementaciones similares.

Se encontró una implementación básica de detección que mediante la consola de Windows permite cargar los archivos de configuración de la red neuronal, alimentar una imagen y retornar la detección resultante de aplicar dicha red.

El diseño de esta herramienta parte de una implementación preexistente de YOLOv3 en Python y Tensorflow obtenida del repositorio https://github.com/YunYang1994/TensorFlow2.0-Examples **[48].** Esto permitió reutilizar los componentes fundamentales de YOLOv3, que ejecutan las tareas de detección y generación de los cuadros predictivos, y son transversales a cualquier modelo que se ejecute.

Para esto, se realizó un fork (bifurcación) del repositorio original. Un fork es una copia de un repositorio. El fork de un repositorio permite a los desarrolladores experimentar libremente con los cambios sin afectar el proyecto original.

Por lo general, las bifurcaciones se utilizan para proponer cambios en el proyecto de otra persona o para utilizar el proyecto de otra persona como punto de partida para una idea propia construyendo un proyecto completamente independiente.

6.2.1 Diagrama de caso de uso

Debido a que esta implementación sirve principalmente como un prototipo para demostrar las capacidades de la red neuronal generada, se identifica un único caso de uso para la misma:

"El actor (usuario) interactúa con la herramienta de software mediante un aplicativo, el cual demuestra el funcionamiento de la red neuronal."

Tradicionalmente, se define un diagrama de casos de uso que permite documentar las distintas interacciones independientes que puede tener un agente externo con el sistema. Este agente externo puede ser un usuario (considerando una variedad de roles), el tiempo (para tareas programadas) u otro sistema.

Para este proyecto, al existir un único rol de usuario y un único caso de uso, el diagrama es el siguiente:

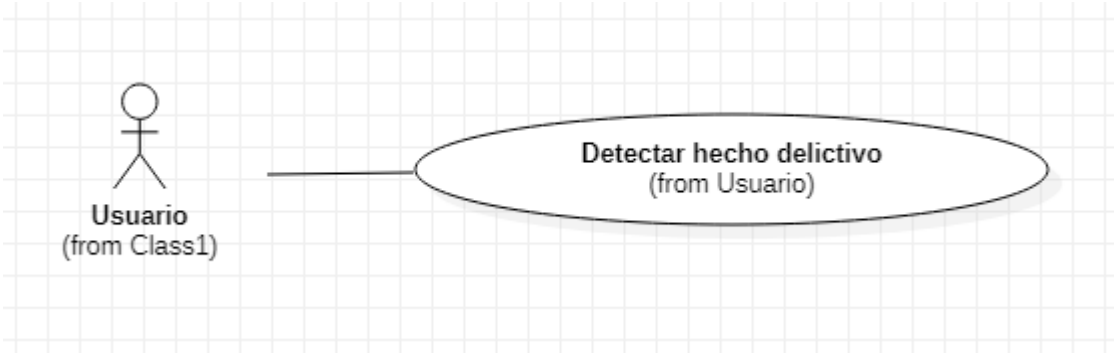


Figura 6.2-1 - Diagrama de casos de uso de la herramienta.

Para especificar en qué consiste el caso de uso "Detectar hecho delictivo" se define la especificación del caso de uso.

El propósito de la misma es definir cuál es el objetivo del caso de uso, qué actores participan, las precondiciones necesarias para que este pueda ejecutarse, y su flujo de operación (principal, para los comportamientos esperados, y alternativo, para los desvíos o comportamientos inesperados)

Especificación de casos de uso:

Proyecto: “Herramienta de software para la detección automática de armas mediante técnicas basadas en aprendizaje profundo”	
Nombre de CU: Detectar hecho delictivo	CU: 1
Objetivo: Detectar la presencia de un hecho de inseguridad a partir del procesamiento de imágenes	
Actores: Cliente	
Precondiciones: -	
Prioridad: ALTA	
Flujo principal	Flujo alternativo
1. La herramienta recibe un input de video para analizar. 2. Se procesa el video utilizando los algoritmos establecidos. 3. Mediante el procesamiento anterior se detecta una probabilidad de presencia de arma de fuego. 4. La herramienta notifica al usuario y espera por la confirmación de una de las opciones 5. De seleccionar SI, la herramienta disparara una alarma y salvará la imagen en el dispositivo 6. Se continúa analizando el fragmento de video restante en búsqueda de presencia de armas de fuego.	3.1 Las probabilidades detectadas no son suficientes para corresponder a la presencia de un arma de fuego. Se continúa con el punto 6 del flujo principal. 4.1 De seleccionar NO se continúa con el punto 6 del flujo principal

Tabla 6.2-1 - Especificación de “Detectar hecho delictivo”.

6.2.2 Diagrama de clases

El siguiente diagrama de clases modela los componentes (clases) que forman parte de la herramienta de detección: sus métodos, atributos e interacciones entre sí:

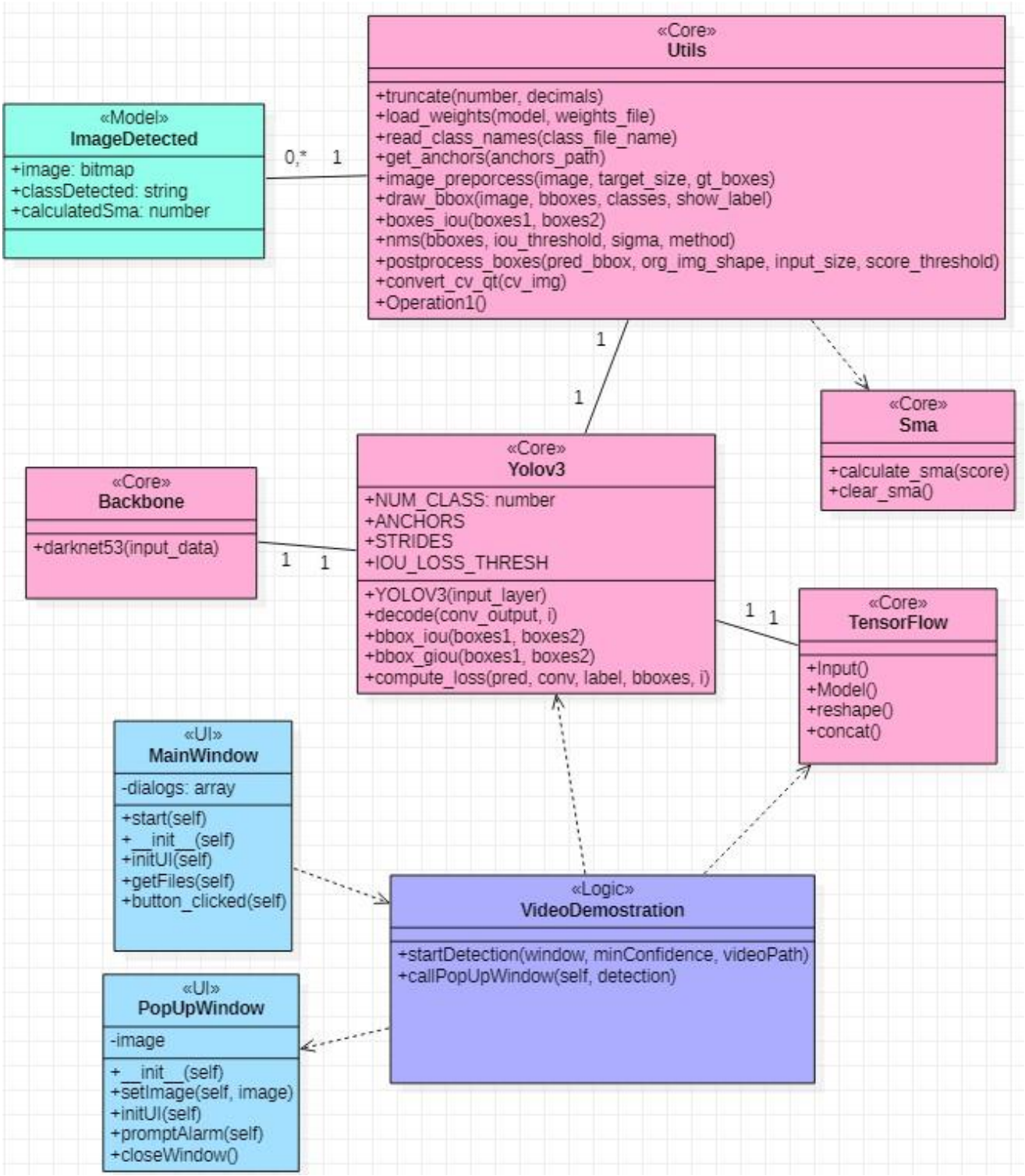


Figura 6.2-2 - Diagrama de clases de la herramienta

Dentro de los elementos que intervienen, se pueden identificar:

- Core:
  - Backbone: clase que define Darknet-53, la cual es una red neuronal convolucional que actúa como columna vertebral para el enfoque de detección de objetos YOLOv3.
  - Yolov3: Clase encargada de las operaciones con bounding boxes y el cálculo de la pérdida.
  - TensorFlow: Clase que exporta las operaciones a realizar con tensores (concat, shape, etc.).
  - Utils: Clase que exporta métodos de utilidad general a lo largo del proyecto, como el dibujado de los bounding box, la carga del archivo weights, lectura de los nombres de clases, entre otros.
  - Sma: Clase encargada de almacenar los valores de confianza de cada detección, para luego realizar el cálculo del simple moving average.
- UI:
  - MainWindow: Clase que renderiza la pantalla principal de la aplicación e incluye la lógica de operación de la misma (carga de video, inicio del procesamiento, seteo de confidence).
  - PopUpWindow: Clase que renderiza la ventana de popUp con la detección a confirmar o rechazar y la lógica asociada.
- Logic:
  - VideoDemostration: Clase encargada de recibir el archivo de video, la confianza y comenzar el procesamiento. Es la clase principal que invoca a todas las demás.
- Model:
  - ImageDetected: Clase de modelado que define las propiedades del objeto imagen detectadas (qué clase se detectó, con qué puntaje, y la imagen en sí).

6.2.3 Diagrama de secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. El diagrama de secuencia es un tipo de diagrama usado para modelar la interacción dinámica entre objetos en un sistema según UML.

Es útil para complementar un diagrama de clases, pues el diagrama de secuencia se podría describir de manera informal como "el diagrama de clases en movimiento", por lo que ambos deben estar relacionados entre sí (mismas clases, métodos, atributos, etc.).

A continuación, se adjunta el diagrama de secuencia para el único caso de uso pertinente de la herramienta.

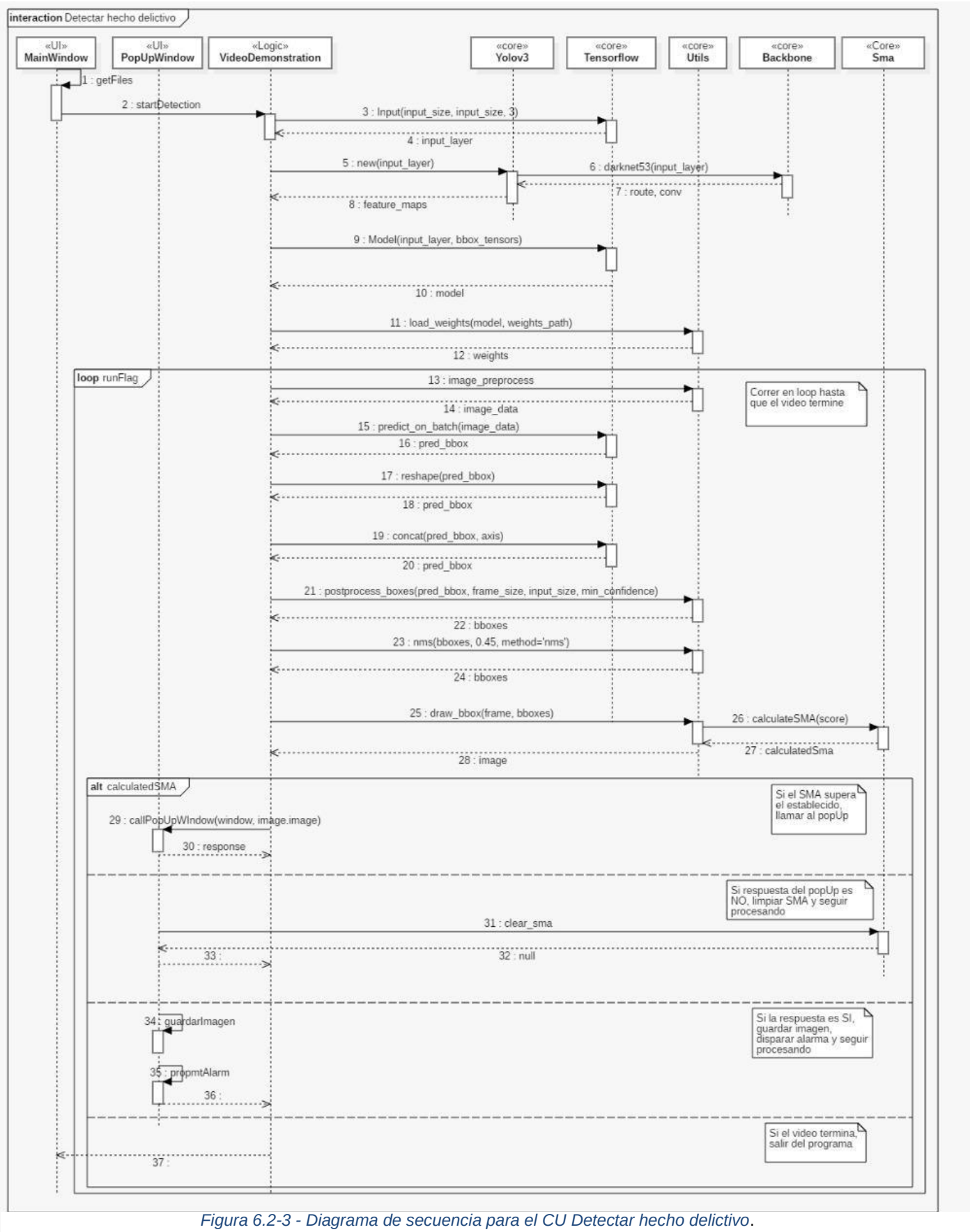


Figura 6.2-3 - Diagrama de secuencia para el CU Detectar hecho delictivo.

6.3 Implementación

6.3.1 Versionado de la aplicación

El control de versiones de software es el proceso de asignar nombres de versión únicos o números de versión únicos a estados únicos del software.

Estos números se asignan en orden creciente y corresponden a nuevos desarrollos en el software.

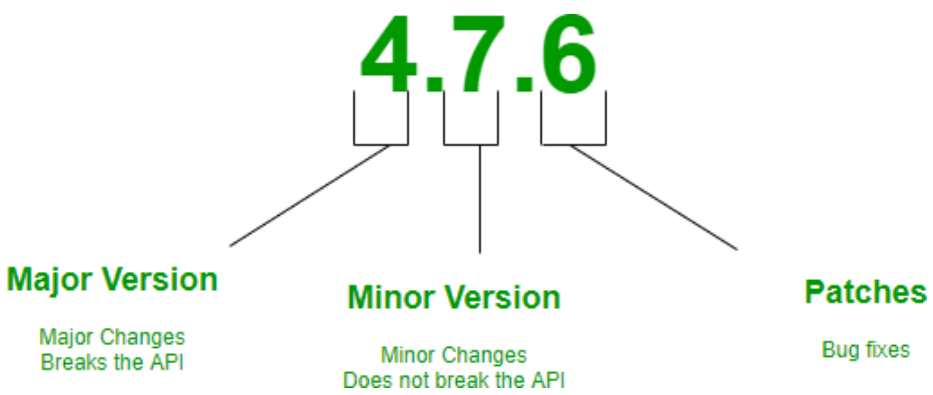


Figura 6.3-1 - Versionado de la aplicación.

El esquema de versionado más común (y el que se optó por utilizar) es el que se indica en la imagen, donde:

- Major: el software sufre grandes cambios y mejoras, por ejemplo, se añaden funcionalidades o se implementan cambios que pueden romper la compatibilidad con ciertas dependencias.
- Minor: el software sufre pequeños cambios y/o correcciones de errores, por ejemplo, se implementan una o varias funcionalidades más pequeñas.
- Patches: se aplica una corrección al software, y a su vez sufre pocos cambios.

Dentro del ciclo de vida de nuestra aplicación, desde su concepción hasta la primer release, se obtuvo el siguiente esquema de versionado:

- *Versión 1.1.1*
    - *Interfaces traducidas al inglés*
    - *Seteo de confidence por defecto*
    - *Modificaciones menores visuales*
- *Versión 1.1.0*
    - Implementado SMA
- *Versión 1.0.0*
    - Se recibe respuesta desde el popup que indica una detección:
        - SI: Disparar alarma, guardar imagen de la detección localmente
        - NO: Continuar ejecución normalmente, reiniciar SMA
- *Versión 0.4.0*
    - Utilización con el modelo entrenado para la detección de armas
    - Comunicación de continuar con la ejecución al cerrar el popup
- *Versión 0.3.0*
    - Pasar imagen con detección a popup
    - Hacer display de un popup por sobre la ventana principal para obtener feedback del usuario
- *Versión 0.2.0*
    - Implementar visor opencv en la ventana principal para mostrar el video
    - Implementar búsqueda de archivos locales
        - Pasar la ruta del archivo al campo para ver cual se seleccionó
    - Pasar el valor del texto de entrada a la funcion de deteccion
- *Version 0.1.0*
    - Estructura general de la ventana principal
    - Estructura de directorios para organización de las clases/funciones
    - Conexión entre la GUI y la función de detección

6.3.2 SMA (Simple moving average)

Un promedio móvil simple (SMA o Simple Moving Average) es el promedio móvil aritmético utilizado en estadística, el cual es calculado sumando valores recientes y luego dividiendo esa cifra por el número de períodos de tiempo en el promedio de cálculo.

Por ejemplo, se podría sumar el precio de cierre de un valor durante varios períodos de tiempo y luego dividir este total por ese mismo número de períodos.

Los promedios a corto plazo responden rápidamente a los cambios en el valor, mientras que los promedios a largo plazo reaccionan más lentamente.

La fórmula para el cálculo del SMA es:

$$\text{SMA} = \frac{A_1 + A_2 + \ldots + A_n}{n}$$

Donde:

- An = valor medido en el periodo n
- n = número total de periodos

Debido a la naturaleza de las redes neuronales, aún las más entrenadas presentan casos donde pueden detectar falsos positivos. En el caso de este proyecto, esto significa que la red neuronal detecta un objeto que no es un arma.

Adicionalmente, la red neuronal obtenida es una versión inicial, por lo que, al configurar valores de confianza muy bajos (< 30%) se observa mayor cantidad de falsos positivos.

El Simple Moving Average (SMA) asiste en suavizar la curva de detecciones para disminuir la probabilidad de que la alarma se dispare porun falso positivo disperso, considerando que un positivo real será detectado múltiples veces en cuadros relativamente consecutivos.

Figura 6.3-2 - Ejemplo de suavizado de curva mediante SMA[6]

Como se puede observar en el gráfico, a mayor cantidad de períodos considerados para el cálculo, el SMA suaviza cada vez más la curva (curva 200 period SMA). Pero, a medida que el número de períodos disminuye (curva 50 period SMA), el SMA refleja cada vez más la forma real de la curva que se intenta suavizar.

Es decir, cuantos más valores pasados se consideren, el SMA refleja menos los cambios bruscos y se mantiene más suavizada.

Para determinar el número de períodos a utilizar, en nuestro caso se realizó un balance entre tiempo de respuesta y precisión. A mayor número de períodos a analizar, se requiere más tiempo en el cual el arma debe estar presente ante la cámara. Si éste es demasiado alto, el tiempo de respuesta de la alarma desbarata el propósito del proyecto, ya que la intención es una detección temprana.

En cuanto a la precisión, si el número de períodos es muy bajo, el SMA replicará la naturaleza real de la curva, que es la que se intenta suavizar, lo que disparará el número de falsos positivos alertados.

[6] Extraído de https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/sma

Basados en la anterior premisa, en conjunto con la performance de la implementación, se optó por un SMA de 10 períodos para la herramienta en Python.

Se cree que este valor proporciona un balance adecuado entre la sobrecarga que genera el SMA y la protección ante los falsos positivos. Asumiendo que la herramienta se ejecuta en un dispositivo capaz de procesar al menos 30 cuadros por segundo, este cálculo no debería retrasar la detección por más de 333 ms, considerando 10 cuadros consecutivos donde se detecte un arma.

Además, si lograse dispararse la alarma por un falso positivo y el usuario indica que la detección no es un arma en la ventana emergente, el SMA se reinicia y comienza a registrar períodos desde cero nuevamente, para evitar repetir detecciones que el usuario determinó como incorrectas.

6.4 Pruebas y Validación

A continuación, se adjuntan casos de prueba a los que se sometió a la herramienta para demostrar las funcionalidades de la misma.

Debido a la naturaleza de la herramienta, el testeo fue hecho manualmente sin el uso de frameworks externos, creando y ejecutando diferentes casos de prueba.

6.4.1 Creación de la suite de pruebas

Al momento de seleccionar los vídeos que conforman la suite de pruebas, se intentaron seguir las siguientes premisas:

- Videos de naturaleza variada, para evitar sesgos.
- Videos de perspectivas, fondos e iluminaciones variadas.
- Videos de calidades diferentes para analizar el efecto de la misma en la detección.

En la siguiente tabla se indican las pruebas de detección realizadas sobre vídeos, observaciones, fuente y resultados:

#	Título	Observaciones	Video	Resultado esperado
1	Watch Unfazed Cashier Keep His Cool During Terrifying Gunpoint Robbery	Ángulo cercano de cámara de seguridad, calidad intermedia, fondo con múltiples objetos	https://drive.google.com/file/d/1nmAu20ffSb0K86a395bVzSNQnVdUBMVb/view?usp=sharing	Detección al minuto 0:05
2	Best draw from Concealment techniques	Hombre a torso casi completo, cámara a distancia media, calidad buena, fondo claro	https://drive.google.com/file/d/1DvUEZ8qbMlG_r5WMYfSNG8wk4rcUCaUb/view?usp=sharing	Detección al minuto 2:53
3	Shoot Fast & Accurately - EP. 2: The Stance	Hombre a cuerpo completo, calidad buena, fondo oscuro	https://drive.google.com/file/d/1r6U5rxkmi4zioq9L9tWigJ2Me5RX3TKL/view?usp=sharing	Detección al minuto 3:22
4	The Concealed Carry Draw Stroke	Cámara detrás del tirador, fondo con vegetación, calidad buena	https://drive.google.com/file/d/1gVL213i2YCAyL1_cYgwSHF7SI3Y4Su0c/view?usp=sharing	Detección al minuto 5:17

5	Watch: Chase and shootings between police and suspects in Las Vegas	Primera persona desde el tirador, calidad baja, cámara con mucho movimiento	https://drive.google.com/file/d/1F0Xt3H8DnTiuEacrce6v3zMe4p8_A2Wd/view?usp=sharing	Detección al minuto 0:02

Tabla 6.4-1 - Pruebas de detección de la herramienta.

6.4.2 Ejecución de las pruebas

Al momento de ejecutar las pruebas, se dejaron constantes todas las variables, siendo el video el único elemento diferente entre las mismas.

Se estableció un SMA de 10 períodos, con un valor de SMA calculado de 0.96 para disparar la alarma. También se utilizó un umbral de confianza de 80% para considerar una detección.

Se corrieron las pruebas en el orden establecido en la tabla anterior y se observó el comportamiento de la herramienta en cada uno.

6.4.3 Resultados

En esta sección se adjuntan los resultados de las mismas, así como algunas conclusiones al respecto.

- Prueba número 1:

Para la primera prueba se eligió un escenario más representativo del ambiente en el cual se desenvolverá la herramienta, aquí se observa un robo a un local gastronómico, se observan dos personas (asaltante y cajero) en un fondo con varios elementos y una calidad de video aceptable.

La dificultad reside en que en los primeros momentos en los que el arma está visible, lo hace sólo por unos instantes, luego el asaltante la enfunda, para posteriormente sí exhibirla claramente por varios segundos.

El resultado deseado es que la misma sea detectada lo más pronto posible.

Ejecución y conclusiones:

- Detección esperada → 0:05 min
- Detección obtenida → 0:07 min

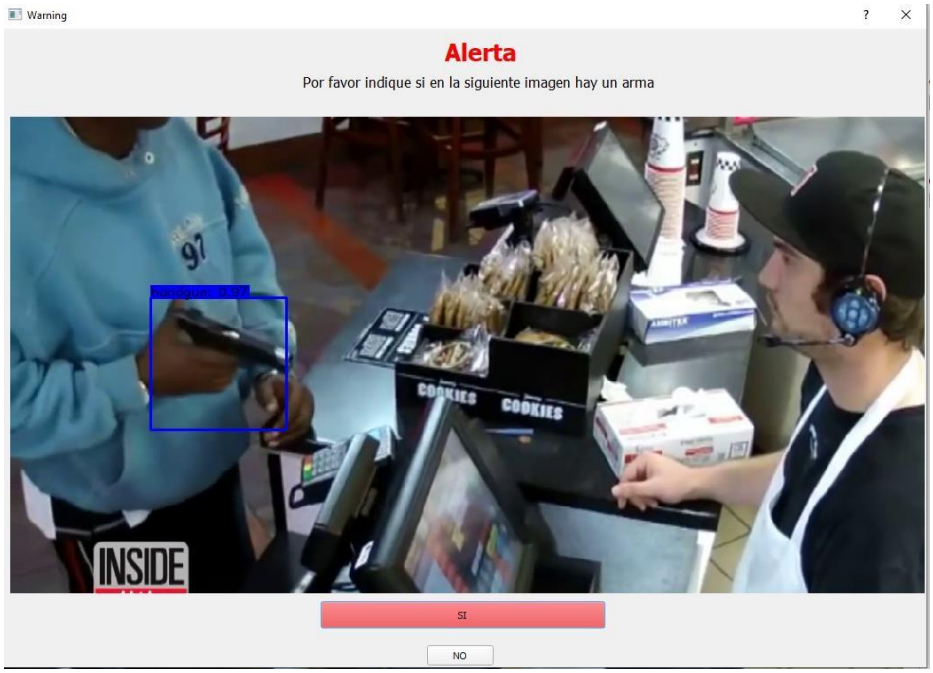


Figura 6.4-1 - Detección prueba N°1

Si bien el resultado presenta una diferencia relativamente leve con respecto al resultado esperado (2 segundos), se analizaron los logs (o registros) de la aplicación para poder obtener más información respecto a su comportamiento.

En la imagen que se ve a continuación, se muestran las detecciones de la herramienta previas al disparo de la alarma:

La columna 'time' indica el número de la detección ordenadas ascendentemente en el tiempo, y la columna "confidence", la confianza con la que se realizó la detección (donde 0 = 0% y 1 = 100%)

Aquí se demuestra el comportamiento descrito en la introducción a la prueba, donde el asaltante muestra el arma inicialmente sólo unos instantes antes de volver a enfundarse (instante time = 2).

```
   time  confidence
0     0    0.872026
Calculated 0
   time  confidence
0     0    0.872026
1     1    0.934048
Calculated 0
   time  confidence
0     0    0.872026
1     1    0.934048
2     2    0.879923
Calculated 0
```

Figura 6.4-2 - Output de detecciones.

Lo que sucede aquí es que la herramienta no consigue obtener las 10 muestras necesarias para calcular el SMA, por lo que debe esperar a que el arma se vuelva a hacer visible.

Figura 6.4-3 - Captura caso de prueba N°1 (arma oculta)

Cuando el asaltante la vuelve a mostrar para amenazar al cajero, la herramienta consigue las muestras restantes y dispara la advertencia, se confirma la presencia de un arma y se dispara la alarma.

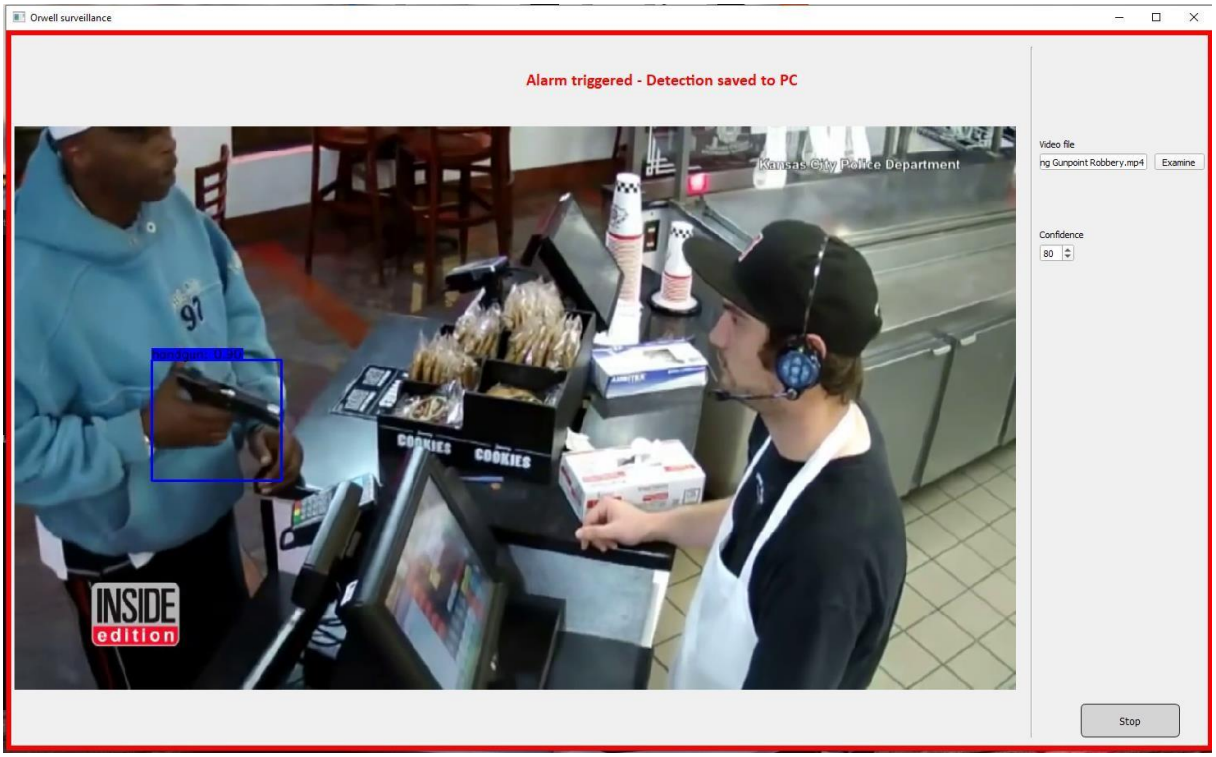


Figura 6.4-4 - Detección confirmada para prueba N°1.

- Prueba número 2

Para esta segunda prueba se toma una cámara un poco más alejada del sujeto armado, aquí el mismo está casi a cuerpo completo. El video posee una calidad muy buena y muy buenas condiciones de iluminación.

En esta prueba, la dificultad reside en que el arma es un porcentaje bastante pequeño del total de la imagen y el gran porcentaje de la misma está cubierta por la mano del tirador.

Ejecución y conclusiones:

- Detección esperada → 2:53 min
- Detección obtenida → 2:54 min

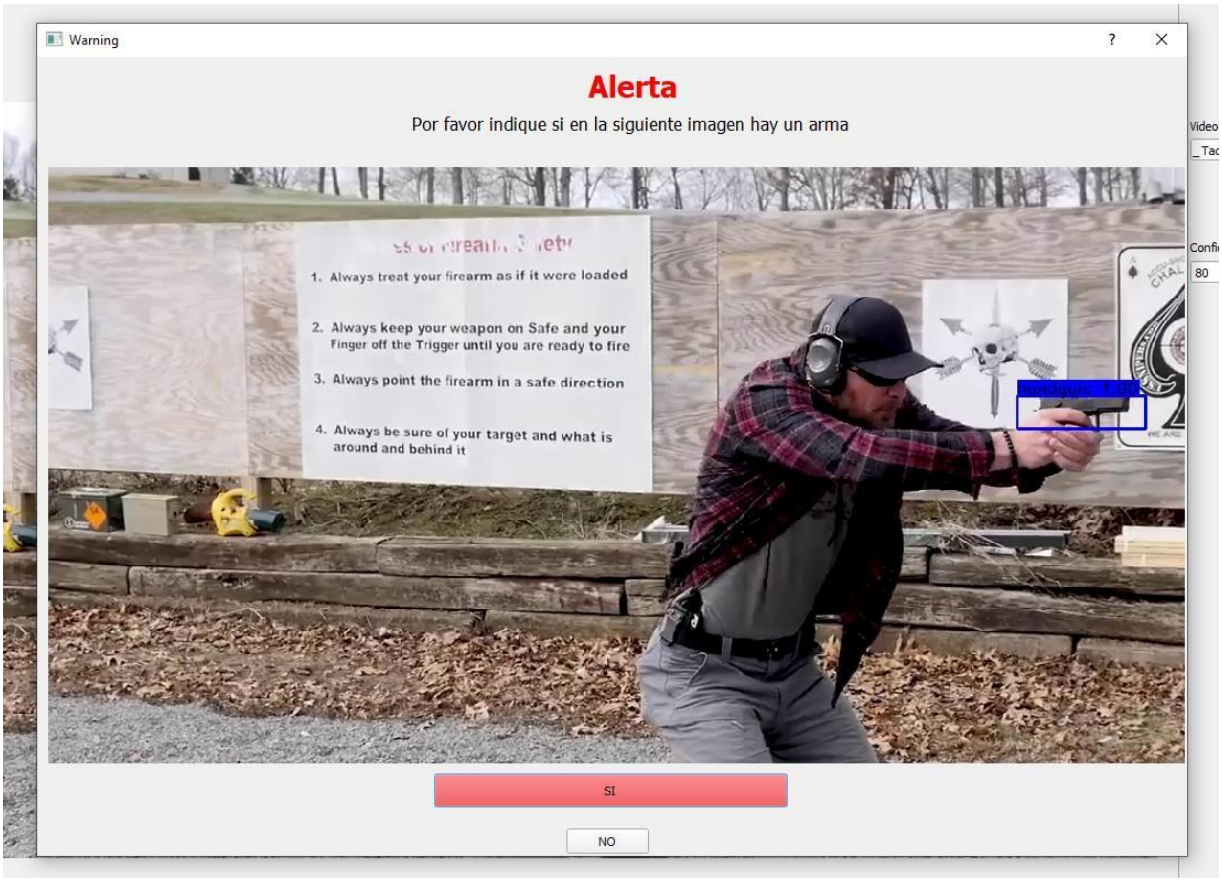


Figura 6.4-5 - Detección en prueba N°2.

Para esta prueba, como puede notarse en el tiempo de respuesta (1 segundo), se demuestra que, aunque el arma sea pequeña con respecto al total de la imagen, las buenas condiciones de luz, imagen y contrastes facilitan la detección.

- Prueba número 3:

Para la tercera prueba se vuelve a cambiar el ángulo de la cámara, aquí está aún más alejada del sujeto. En esta filmación, se observa al tirador a cuerpo completo, de pies a cabeza.

En esta prueba, la mayor dificultad se presenta al momento de detectar un arma de color negro en un fondo de un color similar. Esto dificulta reconocer y diferenciar los bordes que son característicos de un arma, forzando a la red neuronal a eliminar todo el "ruido" de la imagen para poder detectar correctamente el arma.

Ejecución y conclusiones:

- Detección esperada → 3:22 min
- Detección obtenida → 3:27 min

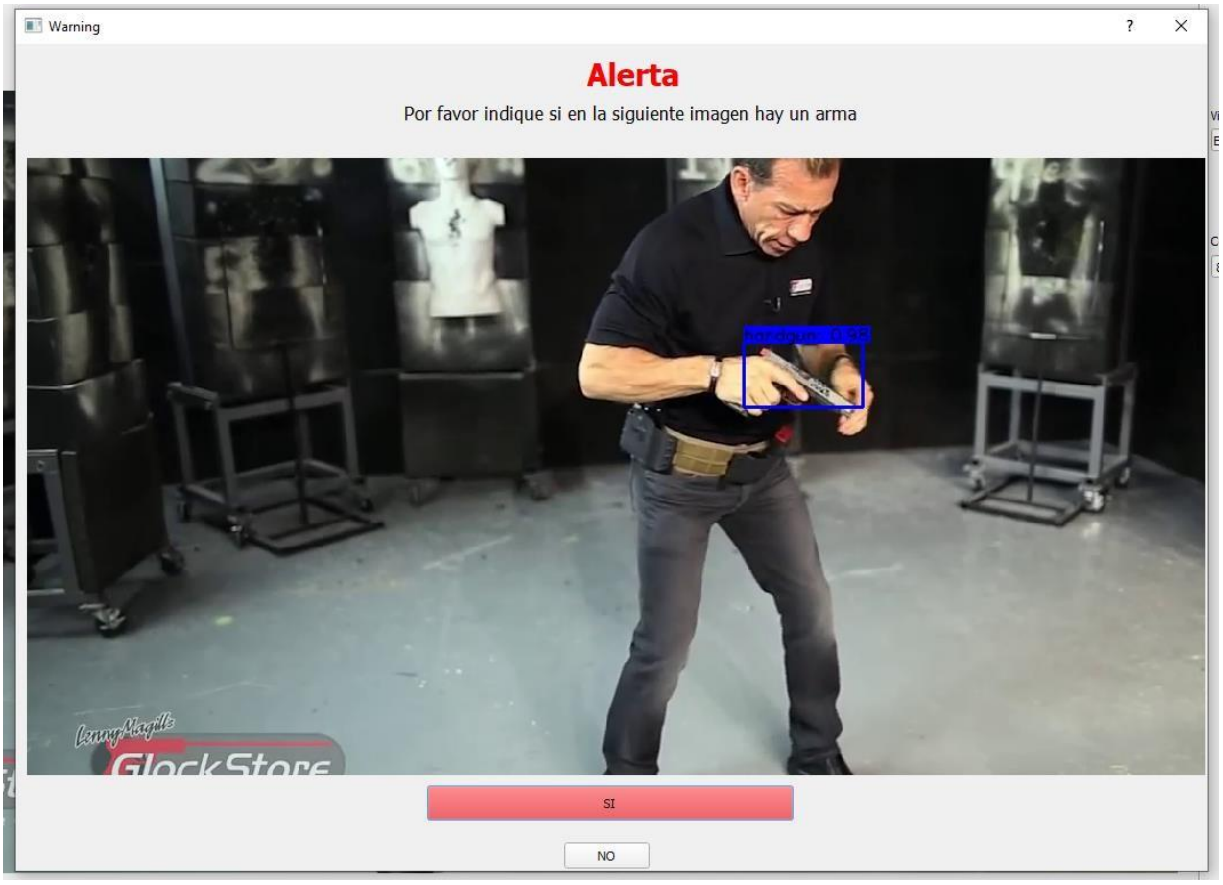


Figura 6.4-6 - Detección en prueba N°3.

En los primeros instantes en los cuales el arma está presente en el período analizado, la misma está apuntando casi directamente a la cámara (cañón al frente). Ésta fue una situación que se consideró no muy representativa de la realidad de las cámaras de seguridad, donde las

mismas en la mayoría de los casos enfocan a un tirador empuñando el arma lateralmente, por lo que dentro del conjunto de imágenes de entrenamiento se incluyeron en su mayoría las de ese último tipo.

Figura 6.4-7 - Movimiento y posición del arma en prueba N°3.

Aquí, la diferencia de tiempo entre la detección deseada y la obtenida es el tiempo que le lleva al tirador mover el arma a una posición lateral, tal como se ve en las imágenes anteriores.

Una vez que el arma se presenta en la posición en la cual la red neuronal puede reconocerla se observa que la alarma se dispara.

- Prueba número 4:

Para esta prueba se utilizó un enfoque diferente, aquí la posición de la cámara y la del sujeto representan una situación que quizás no es la más común considerando casos de cámara de seguridad: la cámara se encuentra detrás del tirador, por sobre su hombro derecho con un fondo muy verde, de mucha vegetación.

La intención de incluir esta prueba es para demostrar la robustez de la red neuronal, es decir, la capacidad de la misma de adaptarse a situaciones para las cuales no fue específicamente entrenada.

La detección es exitosa, ya que la red fue entrenada mayormente con imágenes donde el tirador está orientado lateralmente a la cámara.

Ejecución y conclusiones:

- Detección esperada → 5:17 min
- Detección obtenida → 5:19 min

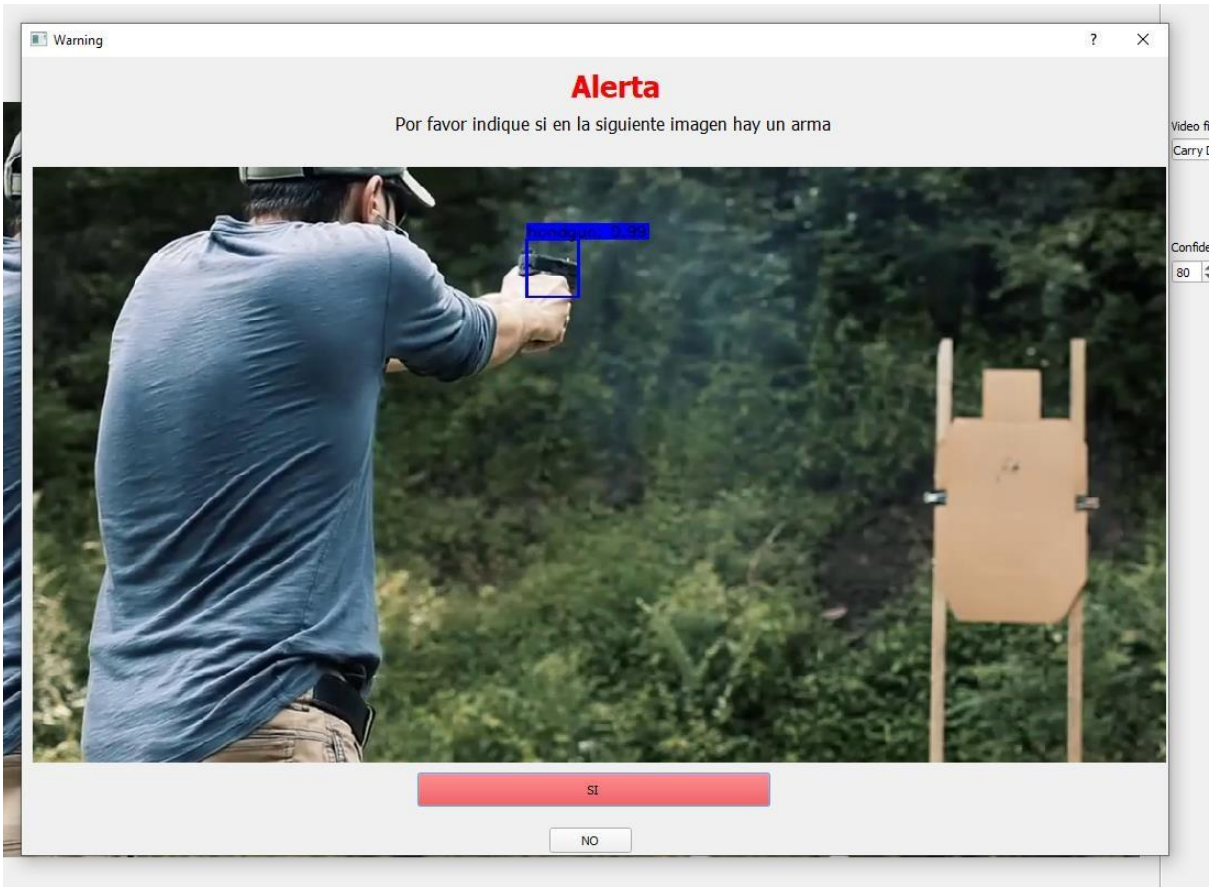


Figura 6.4-9 - Detección caso de prueba N°4.

La herramienta ha demostrado poder generalizar adecuadamente la detección de armas, independientemente del contexto en el cual se utilice.

Esto permite afirmar que el modelo entrenado puede ser utilizado no sólo dentro de un conjunto acotado de situaciones de la vida real, sino que también puede adaptarse a situaciones nuevas y proveer resultados adecuados.

La buena calidad del video y de la iluminación ayuda a que esta detección se realice con precisión muy elevada.

	time	confidence	SMA_10
0	0	0.994863	NaN
1	1	0.996632	NaN
2	2	0.994408	NaN
3	3	0.994389	NaN
4	4	0.999129	NaN
5	5	0.999780	NaN
6	6	0.999746	NaN
7	7	0.996497	NaN
8	8	0.996632	NaN
9	9	0.991313	1.0
Calculated		1.0	

Figura 6.4-8 - Output prueba N°4

- Prueba número 5:

Para la última prueba se eligió otro video para el cual la red neuronal no había sido entrenada. Aquí la calidad del video es mucho menor a los de las pruebas anteriores, y el ángulo de la cámara está en primera persona con respecto al tirador.

Adicionalmente, existe una dificultad propia de los movimientos violentos que causa la manera agresiva de conducir.

Aquí se pone a prueba la capacidad de generalización de la herramienta.

Ejecución y conclusiones:

- Detección esperada → 0:02 min
- Detección obtenida → 0:09 min

Figura 6.4-10 - Detección caso de prueba N°5.

Al observar los resultados de esta prueba, se confirman varias hipótesis supuestas anteriormente.

Aquí, la calidad del video, en conjunto con los movimientos de la cámara, un arma que se no se distingue bien con el fondo sobre el cual se muestra, y un punto de vista diferente hacen que la herramienta tenga que esforzarse más para detectar el arma, lo cual se refleja en el tiempo de detección (7 segundos).

	time	confidence	SMA_10
0	0	0.835373	NaN
1	1	0.983382	NaN
2	2	0.951887	NaN
3	3	0.995567	NaN
4	4	0.998173	NaN
5	5	0.908964	NaN
6	6	0.994704	NaN
7	7	0.950261	NaN
8	8	0.993110	NaN
9	9	0.961194	1.0

Calculated 1.0

Figura 6.4-11 - Output de prueba N°5.

Aun así, como se observa en la imagen, una vez que la herramienta detecta un arma, lo hace con una precisión muy alta (mayor al 90%).

También cabe mencionar algunos casos como el que se muestra en la imagen a continuación:

Figura 6.4-12 - Falso positivo en caso de prueba N°5.

Aquí se obtiene claramente un falso positivo, donde la red neuronal, equivocadamente, indica el espejo retrovisor del vehículo como un arma.

Esto puede ocurrir por varias razones:

- Similitudes físicas entre el objeto que realmente se debe detectar y el falso positivo.
- Baja calidad del video que no permite que los objetos tengan bien delimitados sus bordes y formas.
- El dataset utilizado para entrenar el modelo no tenía el tamaño suficiente.
- El modelo no se entrenó por el suficiente tiempo.

Estas situaciones no deben ser pasadas por alto ya que, si el riesgo de que ocurran no se contempla, en un ambiente real pueden conllevar a situaciones de tensión innecesarias.

Aquí se justifica la incorporación del Simple Moving Average definido anteriormente. De no estar implementado el mismo, esta detección errónea hubiera disparado la alarma. Con esta implementación, la detección errónea se pondera con las demás y sólo dispara la alarma si pasa el umbral, lo cual no ocurre aquí y la alarma se dispara posteriormente con una detección genuina de un arma.

7 Conclusiones

Durante la realización de este proyecto final de carrera se logró adecuadamente desarrollar la herramienta de software propuesta, cumpliendo con los requerimientos definidos.

Como era de esperarse, la realización de este proyecto representó un desafío para los integrantes, ya que implicó apartarse de las tecnologías en las que estábamos más familiarizados en pos de introducirnos en un tópico nuevo que nos atrae y genera curiosidad.

La realización de este proyecto nos permitió cultivar un nuevo conjunto de habilidades y conocimientos técnicos los cuales nos asisten en la resolución de problemas que quizás no hubiésemos podido resolver previamente; y también nos abre nuevas puertas para futuras oportunidades laborales.

A continuación, se concluye sobre la planificación, el componente de software implementado, el impacto del proyecto, y finalmente se delinean trabajos futuros.

7.1 Sobre la planificación:

Considerando el plan de proyecto presentado inicialmente, se concluye que fue una estimación optimista. Se establecieron cargas horarias semanales que no tuvieron en cuenta la posibilidad de que alguno de los integrantes tenga un incremento en las horas laborales, lo que efectivamente sucedió. Ambos integrantes comenzamos a trabajar jornada completa durante la realización de este proyecto, por lo que la carga horaria semanal no pudo ser cumplida.

Este desvío en la cantidad de horas disponibles semanalmente conllevó a que la fecha de entrega propuesta no pueda cumplirse.

Esto puede vincularse directamente con la gestión de riesgos asociada al proyecto. A continuación se adjunta una tabla con los riesgos definidos en el plan de proyecto que se presentaron durante la realización del mismo, detallando estimación de probabilidad (p), estimación de impacto (i) y exposición al riesgo (ER):

#	Riesgo	p	i	ER
2	Se realizan estimaciones inexactas respecto a la cantidad de horas requeridas para realizar el proyecto	5	2	10
4	Uno de los miembros del grupo debe ausentarse temporalmente por razones personales	2	3	6
9	No se cumplen las fechas pactadas para la entrega de los hitos	3	5	15
12	El trabajo con un entorno software desconocido causa problemas no previstos	3	2	6

Tabla 7.1-1 - Riesgos identificados durante la ejecución del proyecto.

Aquí se destaca la necesidad de definir un plan de gestión de riesgos previo al comienzo de un proyecto. Al haberlos definido previamente y establecer una acción de respuesta, pudimos saber cómo gestionarlos cuando ocurrieron y continuar con la realización del proyecto.

7.2 Sobre el componente de software:

Se trató de generar una implementación lo más desacoplada posible con respecto a la naturaleza de la red neuronal y los objetos que ésta busca detectar puntualmente. Se puede observar que la herramienta toma como entrada un video, un valor de confianza, y una red neuronal (mediante sus archivos de configuración).

Cualquier proyecto futuro podría reemplazar los archivos de configuración de la red neuronal con uno cuyo objetivo sea detectar objetos distintos y, junto a algunas modificaciones menores, seguir brindando las mismas funcionalidades.

7.3 Sobre el impacto del proyecto:

Tener un impacto positivo en la seguridad ciudadana fue la razón que justificó la concepción de este proyecto. Se cree que la herramienta presentada logra adecuadamente esto: auxiliar en la detección de un hecho de inseguridad mediante la detección de un arma de fuego en tiempo real.

Segundos luego de que el evento ocurra, le permite a quienes estén a cargo del monitoreo (entidades gubernamentales, seguridad privada o la ciudadanía en general) tomar decisiones rápidamente para mitigar el hecho y potencialmente salvar vidas y preservar bienes materiales.

7.4 Sobre trabajos futuros:

La modularidad con la que fue construida esta herramienta permite que la misma sea adaptada para realizar funciones diferentes en base a una detección.

Actualmente, se configuró para guardar la imagen localmente y disparar la alarma, pero esto puede ser fácilmente modificado para realizar la acción de quien lo implementa desee (enviar la imagen a un servidor, enviar notificaciones a celulares, etc.).

También se provee el dataset original de este proyecto con todas las imágenes utilizadas para el entrenamiento del mismo, esto permite a proyectos futuros:

- Expandir el conjunto inicial de imágenes, sumando nuevas de la misma naturaleza para así aumentar la efectividad de la herramienta. Se ha demostrado que el tamaño del dataset es directamente proporcional a la efectividad que luego presentará la herramienta, por lo que con más recursos asignados se obtendrán precisiones mucho mayores.

- Expandir el conjunto inicial para detectar otro tipo de armas de fuego. Actualmente se centraron los esfuerzos disponibles en detectar pistolas dentro del amplio espectro de armas de fuego existentes. Se podrían sumar más imágenes y clases al dataset para que la red neuronal resultante pueda no sólo detectar una mayor variedad de tipos de armas, sino que también pueda discernir de qué tipo se trata y tomar decisiones según cada tipo.

- Expandir para incluir en el espectro de detección un conjunto de armas blancas, ya que las mismas están presentes en una gran cantidad de hechos delictivos, siendo de vital importancia poder detectarlas y actuar en consecuencia.

Dentro del aspecto arquitectónico, un posible proyecto futuro podría convertir la implementación Desktop actual en una de tipo cliente-servidor, montando el procesamiento de imágenes en un servidor expuesto a través de una API (Application Programming Interface o Interfaz de programación de aplicaciones) a la cual se pueda acceder desde cualquier sitio web, abriendo un nuevo conjunto de posibilidades sobre la portabilidad de la misma.

8 Referencias

[1] Boyle, T. (2020, 10 febrero). Dealing with Imbalanced Data - Towards Data Science. Recuperado de https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18

[2] Brownlee, J. (2019, 7 octubre). How to Perform Object Detection With YOLOv3 in Keras. Recuperado de https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/

[3] Brownlee, J. (2020a, abril 16). How Do Convolutional Layers Work in Deep Learning Neural Networks? Recuperado de https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[4] Brownlee, J. (2020b, agosto 14). What is the Difference Between Test and Validation Datasets? Recuperado de https://machinelearningmastery.com/difference-test-validation-datasets/

[5] Burkov, A. (2019). *The Hundred-Page Machine Learning Book*. Quebec, Canada: Andriy Burkov.

[6] Dai, J. (2016, 20 mayo). R-FCN: Object Detection via Region-based Fully Convolutional Networks. Recuperado de https://arxiv.org/abs/1605.06409

[7] Davies, E. R. (2012). *Computer and Machine Vision: Theory, Algorithms, Practicalities* (4.ª ed.). London, United Kingdom: Academic Press.

[8] Dirección Nacional de Estadística Criminal & Ministerio de Seguridad de la República Argentina. (2019). *ESTADÍSTICAS CRIMINALES República Argentina - 2019*. Recuperado de https://estadisticascriminales.minseg.gob.ar/reports/Informe%20Nacional%20Estadisticas%20Criminales%202019.pdf

[9] Gandhi, R. (2018a, julio 5). Support Vector Machine — Introduction to Machine Learning Algorithms. Recuperado de https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[10] Gandhi, R. (2018b, diciembre 3). R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. Recuperado de https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[11] Gao, H. (2018, 21 junio). Faster R-CNN Explained - Hao Gao. Recuperado de https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

[12] GeeksforGeeks. (2020, 1 marzo). R-CNN vs Fast R-CNN vs Faster R-CNN | ML. Recuperado de https://www.geeksforgeeks.org/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-ml/

[13] Gonfalonieri, A. (2019, 4 noviembre). How to Build A Data Set For Your Machine Learning Project. Recuperado de https://towardsdatascience.com/how-to-build-a-data-set-for-your-machine-learning-project-5b3b871881ac

[14] Great Learning Team. (2020, 11 agosto). Understanding Data Augmentation | What is Data Augmentation & how it works? Recuperado de https://www.mygreatlearning.com/blog/understanding-data-augmentation/

[15] Huang, J. (2016, 30 noviembre). Speed/accuracy trade-offs for modern convolutional object detectors. Recuperado de https://arxiv.org/abs/1611.10012

[16] Hui, J. (2019, 27 agosto). Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3. Recuperado de https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[17] Hui, J. (2020b, enero 15). Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). Recuperado de https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

[18] Hui, J. (2020c, febrero 7). mAP (mean Average Precision) for Object Detection - Jonathan Hui. Recuperado de https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[19] Image detection, recognition and image classification with machine learning. (2019, 31 julio). Recuperado de https://medium.com/ai-techsystems/image-detection-recognition-and-image-classification-with-machine-learning-92226ea5f595

[20] Jaitman, L., Caprirolo, D., & Granguillhome Ochoa, R. (2017). The Costs of Crime and Violence: New Evidence and Insights in Latin America and the Caribbean. *IADB*, 21-28. https://doi.org/10.18235/0000615

[21] Jay, P. (2018, 20 junio). The intuition behind RetinaNet - Prakash Jay. Recuperado de https://medium.com/@14prakash/the-intuition-behind-retinanet-eb636755607d

[22] Jordan, J. (2020, 29 agosto). Setting the learning rate of your neural network. Recuperado de https://www.jeremyjordan.me/nn-learning-rate/

[23] Kan, E. (2018, 11 diciembre). Quick ML Concepts: Tensors - Towards Data Science. Recuperado de https://towardsdatascience.com/quick-ml-concepts-tensors-eb1330d7760f

[24] Klette, R. (2014). *Concise Computer Vision: An Introduction into Theory and Algorithms (Undergraduate Topics in Computer Science)* (2014.ª ed.). London, United Kingdom: Springer.

[25] La provincia está instalando otras 600 cámaras de videovigilancia. (2015, 28 agosto). *Santa Fe Provincia*. Recuperado de https://www.santafe.gob.ar/noticias/noticia/215401/

[26] Lhessani, S. (2020, 18 septiembre). What is the difference between training and test dataset? Recuperado de https://medium.com/@lhessani.sa/what-is-the-difference-between-training-and-test-dataset-91308080a4e8

[27] Lin, T. (2017, 7 agosto). Focal Loss for Dense Object Detection. Recuperado de https://arxiv.org/abs/1708.02002

[28] Liu, W. (2015, 8 diciembre). SSD: Single Shot MultiBox Detector. Recuperado de https://arxiv.org/abs/1512.02325

[29] Mahendran, V. (2019, 15 noviembre). Custom object training and detection with YOLOv3, Darknet and OpenCV. Recuperado de https://blog.francium.tech/custom-object-training-and-detection-with-yolov3-darknet-and-opencv-41542f2ff44e

[30] Muehlemann, A. (2019, 18 noviembre). How to train your own YOLOv3 detector from scratch - Insight. Recuperado de https://blog.insightdatascience.com/how-to-train-your-own-yolov3-detector-from-scratch-224d10e55de2

[31] Nayak, S. (2019, 6 marzo). Training YOLOv3 : Deep Learning based Custom Object Detector. Recuperado de https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/

[32] Nayak, S. (2020, 6 junio). Deep Learning based Object Detection using YOLOv3 with OpenCV (Python / C++). Recuperado de https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/

[33] Norvig, S. R. (2020). *Artificial Intelligence: A Modern Approach, Global Edition* (3rd edition). Hallbergmoos, Alemania: Pearson.

[34] Olafenwa, M. (2020, 7 agosto). Object Detection with 10 lines of code - Towards Data Science. Recuperado de https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606

[35] PLOS ONE. (2019). YOLOv3 architecture. Recuperado de https://plos.figshare.com/articles/YOLOv3_architecture_/8322632/1

[36] PyQt5 tutorial 2020: Create a GUI with Python and Qt. (s. f.). Recuperado de https://build-system.fman.io/pyqt5-tutorial

[37] Redmond, J. (2018, 8 abril). YOLOv3: An Incremental Improvement. Recuperado de https://arxiv.org/abs/1804.02767

[38] Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137-1149. https://doi.org/10.1109/tpami.2016.2577031

[39] Shtym, T. (2020, 16 junio). YOLOv3 vs Gaussian YOLOv3 - Tanya Shtym. Recuperado de https://medium.com/@tanyashtym/yolov3-vs-gaussian-yolov3-d4cc886f9e17

[40] Sinopsis: Seguridad Ciudadana | PNUD. (s. f.). Recuperado de https://www.undp.org/content/undp/es/home/librarypage/crisis-prevention-and-recovery/IssueBriefCitizenSecurity.html#:%7E:text=La%20seguridad%20ciudadana%20es%20el,una%20coexistencia%20segura%20y%20pac%C3%ADfica.

[41] Szeliski, R. (2010). *Computer Vision: Algorithms and Applications (Texts in Computer Science) by Richard Szeliski, Springer*. London, United Kingdom: Springer.

[42] Tsang, S. (2019a, marzo 26). Review: RetinaNet — Focal Loss (Object Detection) - Towards Data Science. Recuperado de https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4

[43] Tsang, S. (2019b, noviembre 11). Review: SSD — Single Shot Detector (Object Detection). Recuperado de https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11

[44] Yelisetty, A. (2020, 14 julio). Understanding Fast R-CNN and Faster R-CNN for Object Detection. Recuperado de https://towardsdatascience.com/understanding-fast-r-cnn-and-faster-r-cnn-for-object-detection-adbb55653d97

[45] Videos utilizados para testing - Disponibles en: https://drive.google.com/drive/folders/1umEt7CS0iKjGS2zBc0XRtrDboXuYbLtb?usp=sharing

[46] Repositorio de Github del proyecto: https://github.com/santoo32/PFC-2020-FRSF

[47] Repositorio implementación original de YOLOv3 (fork): https://github.com/YunYang1994/TensorFlow2.0-Examples

[48] Dataset de training/validation: https://drive.google.com/file/d/1E6Sm4oV6rveJo-VkQxJk2xitap3rJ5td/view?usp=sharing

[49] Dataset de testing: https://drive.google.com/file/d/1pkD6HNhV34TkBtp0FU5T31109aMlsu36/view?usp=sharing

[50] Documentación de Supervisely: https://docs.supervisely.com/

9 Anexo

9.1 Archivo JSON que genera el diagrama DTL:

```
[
  {
    "action": "data",
    "src": [
    "HandgunDataset/*"
    ],
    "dst": "$data",
    "settings": {
      "classes_mapping": "default"
    }
  },
  {
    "action": "flip",
    "src": [
      "$data"
    ],
    "dst": "$data_flip_V",
    "settings": {
      "axis": "vertical"
    }
  },
  {
    "action": "resize",
    "src": [
      "$data",
      "$data_flip_V"
    ],
    "dst": "$dataResized",
    "settings": {
      "width": -1,
```

```
    "height": 720,
    "aspect_ratio": {
    "keep": true
    }
  }
},
{
  "dst": [
    "$totrain",
    "$toval"
  ],
  "src": [
    "$dataResized"
  ],
  "action": "if",
  "settings": {
    "condition": {
      "probability": 0.9
    }
  }
},
{
  "dst": "$train",
  "src": [
  "$totrain"
  ],
  "action": "tag",
  "settings": {
  "tag": "train",
    "action": "add"
  }
},
{
```

```
  "dst": "$data_SW",
  "src": [
    "$train"
  ],
  "action": "sliding_window",
  "settings": {
    "window": {
      "width": 800,
      "height": 540
    },
    "min_overlap": {
      "x": 320,
      "y": 180
    }
  }
},
{
  "action": "if",
  "src": [
  "$data_SW"
  ],
  "dst": [
    "$data_to_rotate",
    "$data_not_to_rotate"
  ],
  "settings": {
    "condition": {
      "probability": 0.2
    }
  }
},
{
  "action": "rotate",
```

```
  "src": [
    "$data_to_rotate"
  ],
  "dst": "$data_rotated",
  "settings": {
  "rotate_angles": {
      "min_degrees": 70,
      "max_degrees": 110
    },
    "black_regions": {
      "mode": "keep"
    }
  }
},
{
  "action": "if",
  "src": [
  "$data_rotated",
    "$data_not_to_rotate"
  ],
  "dst": [
    "$to_train_ok",
    "$to_train_trash"
  ],
  "settings": {
    "condition": {
      "min_objects_count": 1
    }
  }
},
{
  "dst": "$val",
  "src": [
```

```
    "$toval"
  ],
  "action": "tag",
  "settings": {
  "tag": "val",
    "action": "add"
  }
},
{
  "dst": "$data_SW_Val",
  "src": [
    "$val"
  ],
  "action": "sliding_window",
  "settings": {
    "window": {
      "width": 800,
      "height": 540
    },
    "min_overlap": {
      "x": 320,
      "y": 180
    }
  }
},
{
  "action": "if",
  "src": [
  "$data_SW_Val"
  ],
  "dst": [
    "$data_V_to_rotate",
    "$data_V_not_to_rotate"
```

```
  ],
  "settings": {
    "condition": {
      "probability": 0.2
    }
  }
},
{
  "action": "rotate",
  "src": [
  "$data_V_to_rotate"
  ],
  "dst": "$data_V_rotated",
  "settings": {
  "rotate_angles": {
      "min_degrees": 70,
      "max_degrees": 110
    },
    "black_regions": {
      "mode": "keep"
    }
  }
},
{
  "action": "if",
  "src": [
    "$data_V_not_to_rotate",
    "$data_V_rotated"
  ],
  "dst": [
    "$to_val_ok",
    "$to_val_trash"
  ],
```

```
    "settings": {
      "condition": {
        "min_objects_count": 1
      }
    }
  },
  {
    "dst": "HandgunDTL-004",
    "src": [
      "$to_train_ok",
      "$to_val_ok"
    ],
    "action": "supervisely",
    "settings": {}
  }
]
```