

## CARRERA DE ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL

MEMORIA DEL TRABAJO FINAL

### Estimación de la captura realizada en buques pesqueros mediante visión artificial

**Autor:**  
**Lic. Nicolás Eduardo Horro**

Director:  
PhD. Félix Ramón Rojo LaPalma (INVAP S.E.)

Jurados:  
Esp. Ing. Santiago Salamandri (FIUBA)  
Mg. Ing. Pablo Slavkin (FIUBA)  
Mg. Ing. Franco Bucafusco (FIUBA)

*Este trabajo fue realizado en la Ciudad de San Carlos de Bariloche,  
entre abril de 2021 y octubre de 2021.*



## *Resumen*

En el presente documento se describe el desarrollo del software de un prototipo para estimar la captura y el descarte en buques pesqueros mediante cámaras de video y técnicas de visión artificial. Este sistema fue desarrollado en INVAP S.E. como una alternativa de mejora en la explotación de los recursos pesqueros.

Los conocimientos necesarios para su implementación incluyen el ciclo completo de desarrollo de modelos de aprendizaje automático. En particular, se utilizan los algoritmos YOLOv4 y DeepSORT para detección y seguimiento de objetos. El desarrollo incluye otros componentes para la adquisición de video y registro de la información procesada en las bases de datos InfluxDB y ElasticSearch que están orientadas a series temporales y eventos respectivamente.



# Índice general

<b>Resumen</b>	<b>I</b>
<b>1. Introducción general</b>	<b>1</b>
1.1. La pesca industrial . . . . .	1
1.2. Marco de la propuesta . . . . .	3
1.3. Estado del arte . . . . .	4
1.3.1. Análisis de contenido en video . . . . .	4
1.3.2. Video analítico para pesca . . . . .	5
1.4. Motivación y alcance . . . . .	6
1.4.1. Requerimientos . . . . .	8
<b>2. Introducción específica</b>	<b>11</b>
2.1. Técnicas de visión por computadora e inteligencia artificial . . . . .	11
2.2. Aprendizaje supervisado y no supervisado . . . . .	12
2.2.1. Problema de regresión . . . . .	12
2.2.2. Problema de clasificación . . . . .	12
Evaluación de un clasificador binario . . . . .	13
Accuracy . . . . .	13
Precision . . . . .	14
Recall . . . . .	14
Curva ROC . . . . .	14
F1-Score . . . . .	14
Matriz de confusión . . . . .	15
2.3. Redes neuronales con capas convolucionales . . . . .	15
2.4. Detección de objetos . . . . .	15
2.4.1. Modelo de caja negra . . . . .	16
2.4.2. Evaluación de modelos . . . . .	17
2.4.3. Intersección sobre unión . . . . .	17
2.4.4. Métricas de evaluación . . . . .	17
<i>Mean average precision (mAP)</i> . . . . .	18
2.4.5. Arquitecturas de detectores . . . . .	20
2.4.6. El algoritmo YOLO . . . . .	20
Supresión de no máximos . . . . .	20
2.4.7. La detección de objetos como un problema de aprendizaje multitarea . . . . .	21
Función de costo de clasificación . . . . .	22
Función de costo de localización . . . . .	22
Función de costo de presencia de objeto . . . . .	22
Función de costo total . . . . .	23
2.4.8. Evolución de la familia de algoritmos YOLO . . . . .	23
YOLOv2 . . . . .	23
YOLOv3 . . . . .	23
YOLOv4 . . . . .	24

2.5.	Seguimiento de objetos . . . . .	25
2.5.1.	Técnicas de seguimiento de objetos . . . . .	25
2.5.2.	Seguimiento de múltiples objetos con SORT y DeepSORT . . . . .	26
	Modelo de movimiento y filtro de Kalman . . . . .	26
	Modelo de apariencia . . . . .	27
	Problema de asociación y algoritmo húngaro . . . . .	27
<b>3.</b>	<b>Diseño e implementación</b>	<b>29</b>
3.1.	Arquitectura del sistema . . . . .	29
3.2.	Desarrollo de modelos de aprendizaje automático . . . . .	33
3.3.	Desarrollo de modelos de detección . . . . .	35
3.3.1.	Entrenamiento para detección de presas con dataset de Kaggle . . . . .	36
	Obtención de los datos . . . . .	37
	Exploración de los datos . . . . .	37
	Etiquetado y preparación de datos para Darknet . . . . .	39
	Ajuste de hiperparámetros . . . . .	40
	Entrenamiento en ambiente Darknet . . . . .	40
	Evaluación . . . . .	40
	Exportación de modelos a Tensorflow . . . . .	41
3.3.2.	Entrenamiento para detección de redes, operadores y descarte en pesca de langostinos . . . . .	41
3.3.3.	Preparación de datos y aumentado . . . . .	42
3.4.	Desarrollo de un modelo de clasificación de actividades . . . . .	43
3.4.1.	Etiquetado manual y preparación de datos . . . . .	44
	Identificación de actividades . . . . .	44
	Ánalysis exploratorio e ingeniería de características . . . . .	47
	Selección de características . . . . .	48
3.4.2.	Sistematización del método . . . . .	49
3.5.	Desarrollo de modelos de extracción de características . . . . .	49
3.5.1.	Redes siamesas . . . . .	50
	Entrenamiento con <i>triplet loss</i> . . . . .	50
3.5.2.	Arquitecturas de extractores . . . . .	53
	Extractor basado en Resnet50 . . . . .	53
3.6.	Desarrollo de un framework modular . . . . .	55
3.6.1.	El paradigma de tuberías . . . . .	55
3.6.2.	Fuentes . . . . .	57
3.6.3.	Sumideros . . . . .	58
	Detección de objetos . . . . .	58
	Seguimiento de objetos . . . . .	59
	Definición de regiones de interés . . . . .	60
3.7.	Despliegue con microservicios e integración con bases de datos . . . . .	62
3.7.1.	InfluxDB y TICK . . . . .	63
3.7.2.	ElasticSearch y ELK . . . . .	63
3.7.3.	Configuración con Docker Compose . . . . .	65
<b>4.</b>	<b>Ensayos y resultados</b>	<b>69</b>
4.1.	Aplicaciones . . . . .	69
4.2.	Aplicación de detección de objetos . . . . .	70
4.2.1.	Resultados para modelo de capturas . . . . .	73
4.2.2.	Resultados para modelo para aplicación de langostinos . . . . .	75

4.3.	Aplicación de seguimiento de objetos . . . . .	76
4.3.1.	Caso de uso: conteo con ROIs . . . . .	78
4.3.2.	Configuración de SORT y DeepSORT . . . . .	78
4.4.	Monitoreo e integración con bases de datos . . . . .	80
4.4.1.	Requisitos de cómputo . . . . .	81
4.5.	Aplicación de clasificación de actividades . . . . .	82
<b>5.</b>	<b>Conclusiones</b>	<b>85</b>
5.1.	Resultados obtenidos . . . . .	85
5.2.	Próximos pasos . . . . .	86
	<b>Bibliografía</b>	<b>87</b>



# Índice de figuras

1.1.	Ordenación del recursos pesqueros. . . . .	2
1.2.	Esquema de un sistema de monitoreo electrónico . . . . .	3
1.3.	Ejemplos de escenas en el interior de buques. . . . .	6
2.1.	Diagrama simplificado de una red neuronal con capa convolucional. . . . .	16
2.2.	Clasificación, localización, detección y segmentado de objetos . . . . .	16
2.3.	Definición de TP,FP y FN para detección de objetos. . . . .	18
2.4.	Curva de <i>precision-recall</i> para cálculo de mAP en detección de objetos antes y después de suavizado. . . . .	19
2.5.	Grilla y supresión de no máximos en algoritmo YOLO. . . . .	21
2.6.	Comparativa entre YOLOv4 y otros detectores de objetos en abril de 2020. . . . .	24
3.1.	Diagrama en bloques del sistema. . . . .	30
3.2.	Tipos de productos y gestión de ambientes. . . . .	31
3.3.	Flujo de trabajo para un problema de aprendizaje automático. . . . .	34
3.4.	Especies en conjunto de datos de competencia de Kaggle. . . . .	37
3.5.	Ejemplos de los datos de la competencia de Kaggle. . . . .	38
3.6.	Distribución de clases en los datos de la competencia de Kaggle. . . . .	39
3.7.	Ejemplo de detección para modelo de detector entrenado con datos de Kaggle. . . . .	41
3.8.	Captura descargada en cubierta en buque tangonero. . . . .	42
3.9.	Muestras luego del proceso de aumentado. . . . .	43
3.10.	Levado de redes (1/3). . . . .	45
3.11.	Redes levantadas. . . . .	46
3.12.	Descarga de redes. . . . .	46
3.13.	Clasificación de capturas. . . . .	46
3.14.	Evolución en el tiempo de media de posición horizontal y desvío estándar y oblicuidad de posición vertical de operadores. . . . .	47
3.15.	Histogramas para desvío estándar de posición horizontal de operadores en distintos momentos del video. . . . .	48
3.16.	Representatividad de las clases en el problema . . . . .	48
3.17.	Proceso completo para generar un clasificador de actividades. . . . .	49
3.18.	Arquitectura de red siamesa para <i>triplet loss</i> . . . . .	51
3.19.	Ejemplo de tubería para clasificación basada en trayectorias . . . . .	56
3.20.	Diagrama de paquetes (simplificado) . . . . .	57
3.21.	Diagrama de clases de componentes de detección en Videoanalytics. . . . .	59
3.22.	Conteo de objetos por ROI utilizando salida de detector de objetos. . . . .	60
3.23.	Estimación de movimiento por ROI utilizando substractor de fondo. . . . .	61
3.24.	Componentes de TICK. . . . .	64
3.25.	Configuración de microservicios con Docker Compose (diagrama simplificado). . . . .	65

4.1. Visualización del grafo para procesamiento de detección de objetos.	72
4.2. Resultados de la aplicación de detección sobre videos de dominio público.	74
4.3. Resultados de la aplicación de detección de operadores, redes y pescados sobre videos de dominio público.	76
4.4. Visualización del grafo para procesamiento de seguimiento de objetos.	78
4.5. Aplicación con seguimiento y ROIs para conteo de atunes.	79
4.6. Integración con InfluxDB y Grafana en aplicación de conteo.	80
4.7. Integración con ElasticSearch y Kibana en aplicación de registro de detecciones.	81
4.8. Tiempo de cómputo por cada componente para una aplicación con detección, seguimiento y publicación en bases de datos.	81
4.9. Test de información mutua, test de ANOVA y test de correlación de Kendall.	82
4.10. Desempeño de los modelos de clasificación (AUC).	83
4.11. Tiempo de inferencia de cada modelo.	83
4.12. Tiempo de inferencia de cada modelo, excluyendo KNN.	84

# Índice de tablas

1.1.	Productos para video analítico . . . . .	5
1.2.	Especies y arte de pesca asociada. . . . .	7
3.1.	Momentos identificados en video de pesca de langostino. . . . .	45
3.2.	Entornos de inferencia para modelos de detección de objetos. . . .	59
4.1.	Evaluación del detector para caso de uso de presas . . . . .	73
4.2.	Evaluación del detector para el caso de uso de langostinos. . . .	75



# Capítulo 1

## Introducción general

En este capítulo se presenta una breve reseña sobre la pesca industrial en Argentina y la tendencia mundial a incorporar en esta actividad sistemas de monitoreo electrónico. Luego se trata el estado del arte de la visión artificial y el análisis de contenido en video para este campo de aplicación. Al final del capítulo se describen el alcance, los objetivos y la motivación de este trabajo.

### 1.1. La pesca industrial

La pesca industrial es un tipo de pesca que tiene como objetivo obtener un gran número de capturas. En Argentina el sector primario pesquero (aquel que se ocupa de la captura) se compone de una flota de buques fresqueros de altura, de costeros grandes y costeros chicos y una flota de buques procesadores. La flota fresquera de altura está conformada por barcos arrastreros con bodegas refrigeradas que cuentan con equipamiento de navegación y detección y utilizan redes de arrastre [1].

La figura 1.1 expone un diagrama simplificado de los tipos de embarcaciones y métodos de captura de acuerdo al objetivo y la profundidad.

A fin de garantizar que la pesca a esta escala sea sostenible, el Consejo Federal Pesquero [2] tiene como función establecer la política pesquera nacional. Esto incluye la planificación del desarrollo pesquero nacional, las políticas de investigación del recurso, la determinación de la Captura Máxima Permisible (CMP) por especie, la aprobación de los permisos de pesca comercial y experimental y el establecimiento de cánones para el ejercicio de la pesca, entre otros [1].

Dado que una pesquería es un sistema complejo de factores interdependientes entre los que se cuentan el estado del recurso biológico, limitaciones sociales e institucionales, condiciones económicas y convicciones culturales, es importante generar información estadística e indicadores que permitan el análisis de la actividad de las pesquerías para poder establecer de manera precisa su marco regulatorio [3].

El monitoreo y la vigilancia deben permitir conocer las características del esfuerzo en la actividad pesquera y asegurar que las capturas se realicen dentro de los cánones admitidos. Existen también regulaciones que establecen el modo en que debe emplearse la técnica, por ejemplo, exigiendo una permanencia mínima de las redes en el fondo para disminuir la captura incidental de mamíferos marinos [4].

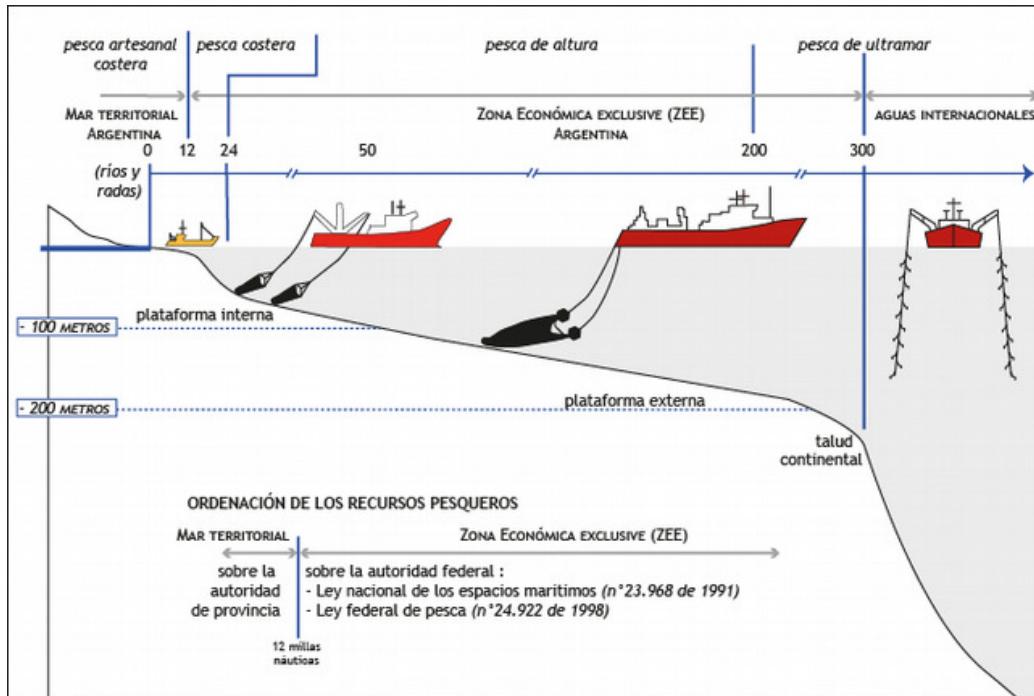


FIGURA 1.1. Tipos de embarcaciones según el tipo de pesca <sup>1</sup>.

Tradicionalmente, la principal manera de recopilar información independiente acerca de las actividades y la captura de los buques ha sido mediante observadores a bordo, que reportaban en un informe manuscrito la cantidad de capturas y descarte de cada especie. A este informe se le denomina parte de pesca, y dependiendo del tipo de captura puede incluir información del día o hasta cuarenta días.

Actualmente existe una creciente tendencia a la utilización de sistemas electrónicos de seguimiento (SE) y un mayor grado de automatización [5], que representan una alternativa más eficiente y rentable que la elaboración de reportes manuscritos. No obstante, el volumen de datos generados, la manipulación de sus medios de almacenamiento y la dependencia de observadores calificados para extraer conclusiones e indicadores útiles sigue siendo un procedimiento costoso y propenso a errores [6].

Las nuevas técnicas en las áreas de visión por computadora e inteligencia artificial hallan un posible campo de aplicación en la optimización de estas tareas de análisis, como lo demuestran algunos trabajos con resultados alentadores [7, 8, 9, 10, 11, 12] y la competencia de clasificación de capturas en buques del conocido sitio de competencias de aprendizaje automático Kaggle [13].

La figura 1.2 muestra un esquema de un sistema de seguimiento electrónico típico y algunos ejemplos de sus usos.

<sup>1</sup>Imagen obtenida de <http://revistas.ubiobio.cl/index.php/TYE/article/view/3982/3837>. Accedido: 05/09/2021.



FIGURA 1.2. a) Sistema de Seguimiento Electrónico. b) Sistema de CCTV que apunta a redes de arrastre. c) Registro de especímenes por observadores calificados. d) Cámara captando descarte de especie protegida, probablemente no declarada <sup>2</sup>.

El presente trabajo incorpora algunas de estas técnicas a un sistema existente de monitoreo electrónico para lograr un mayor grado de automatismo. Esto lo diferencia de otros sistemas similares en los que se requiere de mayor participación de personal calificado.

## 1.2. Marco de la propuesta

INVAP S.E. [14] es una empresa del Gobierno de Río Negro dedicada al diseño y construcción de sistemas tecnológicos complejos entre los que se incluyen satélites y reactores nucleares. Dentro del contexto de la búsqueda de nuevos negocios e incorporación de nuevas tecnologías, se realizan trabajos de investigación y prototipos, a menudo mediante convenios con universidades y otras organizaciones. Estos trabajos pueden eventualmente evolucionar e incorporarse en sistemas de mayor complejidad. Este trabajo es parte de un sistema que utiliza cámaras de video y computadoras o dispositivos auxiliares para el control de la pesca.

El principal cliente interesado es el Estado Argentino y, en particular, la Subsecretaría de Pesca y Acuicultura perteneciente a la Secretaría de Agricultura, Ganadería y Pesca del Ministerio de Agricultura, Ganadería y Pesca.

Si bien no son clientes directos, es también importante mencionar la participación del Consejo Federal Pesquero y de INIDEP [15], organismos involucrados en el seguimiento y regulación de la explotación del recurso y promotores de la modernización de los sistemas de monitoreo y seguimiento. También pueden ser potenciales usuarios centros de investigación o entes privados que lo utilicen para registro propio.

El trabajo realizado es un prototipo de software cuyo objetivo es el procesamiento y envío del parte de pesca en tiempo real y la automatización de algunas tareas de análisis.

<sup>2</sup>Imagen adaptada de: <https://www.pewtrusts.org/es/research-and-analysis/issue-briefs/2019/09/electronic-monitoring-a-key-tool-for-global-fisheries>. Accedido 10/09/2021.

### 1.3. Estado del arte

Si bien el interés por automatizar tareas relacionadas con la elaboración de partes electrónicos de pesca a partir de videos a bordo es relativamente reciente, los sistemas de vigilancia de circuito cerrado de televisión (CCTV, acrónimo usado también en inglés por *Closed Circuit TeleVision*) fueron inventados en el año 1942 y comenzaron a comercializarse por la empresa Vericon en el año 1949 [16].

Desde entonces han estado evolucionando, incluyendo al principio capacidades modestas de detección de movimiento hasta incorporar las técnicas más avanzadas de visión por computadora, inteligencia artificial y aprendizaje profundo que incluyen identificación de rostro, detección y seguimiento de múltiples tipos de objetos, reconocimiento de poses y patrones de conducta en personas, cumplimiento de normas de seguridad en operadores, y otros. Una de sus aplicaciones es la vigilancia de espacios públicos como plazas y centros comerciales.

#### 1.3.1. Análisis de contenido en video

A la capacidad de analizar video para detectar y determinar eventos espaciales y temporales se le denomina “análisis de contenido en video” o “video analítico”, y consiste en combinar múltiples técnicas de visión por computadora, inteligencia artificial y aprendizaje profundo para extraer información útil de una secuencia de cuadros de video. A continuación se describen las más relevantes para este trabajo:

- Enmascarado dinámico: bloquear una parte de una señal de video, como por ejemplo para reducir intencionalmente la resolución de los rostros. Aún cuando esta función no se use para extraer información, puede ser requerida en una solución de video analítico que tenga que aplicarse a una escena en la que participen personas y existan normativas de privacidad.
- Detección de movimiento: determinar la presencia de algún movimiento relevante en una escena. A modo de ejemplo, esto puede incluir la discriminación de movimientos de objetos o personas de variaciones en la imagen por sombras, iluminación, movimiento del barco por las olas, u otros factores que no aporten información al problema.
- Detección de objetos: determinar la presencia y localización de un tipo de objeto o entidad, como por ejemplo una persona, red, capturas (pescados y otras presas), etc.
- Reconocimiento e identificación: reconocer un rostro o la matrícula de un vehículo en una imagen y asociarla a una identidad.
- Detección de bloqueo: determinar si la señal de la cámara está siendo tapada o afectada intencionalmente por otro método (por ejemplo, encadilada).
- Seguimiento: determinar la posición de objetos en una señal de video, posiblemente respecto a una grilla de referencia externa.
- Conteo y registro de eventos: activar alarmas o registrar eventos cuando un objeto ingresa, egresa, o permanece un tiempo determinado en una región de la imagen.

- Clasificación: indicar si una imagen o fragmento de la señal de video se corresponde con una categoría. Esto requiere que previamente se haya definido un conjunto de categorías y exista un algoritmo que permita reconocerlas. A modo de ejemplo, si en la imagen está ocurriendo una actividad, que en el contexto de la pesca podría ser “descarga de redes”, “operadores en tiempo de descanso” o “clasificación de capturas”, entre otros.
- Búsqueda en video: tener la capacidad de obtener los momentos del video en los que ocurre algún evento de interés utilizando un lenguaje de consulta de bases de datos como SQL o de obtener las imágenes del video que satisfacen algún criterio de búsqueda, como por ejemplo aquellas en las que es visible una red de pesca (paso requerido, para luego sobre estas imágenes utilizar algoritmos de estimación volumétrica, por ejemplo).

Dentro del software de video analítico que está disponible en el mercado existen productos que implementan estas funciones, por lo general destinados a aplicaciones de vigilancia y seguridad tanto en ambientes civiles como industriales. También existen proyectos de código abierto de características similares. En algunos casos, cuando la aplicación es muy específica, estos productos deben ser extendidos y requieren el desarrollo de componentes adicionales o una configuración avanzada que debe ser realizada por expertos.

El término *Commercial off-the-shelf* (COTS) hace referencia a un producto que está listo para su uso y en principio no requiere configuración. Cuando un producto de software puede ser extendido por desarrolladores el fabricante suele indicar que se dispone de un kit de desarrollo de software (SDK, por sus siglas en inglés *Software Development Kit*). Pueden existir motivos por los que sea conveniente implementar desde cero una aplicación de procesamiento de video, como por ejemplo no depender de tecnología propietaria, con fines de investigación, o para tener control completo sobre el procesamiento y tratamiento de la información. Para esos casos existen bibliotecas o *frameworks* orientados a facilitar el desarrollo de aplicaciones de procesamiento de video distribuido.

En la tabla 1.1 se mencionan algunas soluciones comerciales y otras de código abierto que entran en la categoría de COTS. También se consideran *frameworks*.

TABLA 1.1. COTS y frameworks para soluciones de video analítico y procesamiento distribuido de video

Producto	Tipo	Licencia
Samsung SDS [17]	COTS	Comercial
Bosh Security Video Analytics [18]	COTS	Comercial
OpenDataCam [19]	COTS	Código abierto
NVIDIA Deepstream SDK [20]	Framework	Gratis/propietario
Videoflow [21]	Framework	Código abierto
VidGear [22]	Framework	Código abierto

### 1.3.2. Video analítico para pesca

La distorsión de lente de pez que suelen tener las cámaras de vigilancia, las condiciones de iluminación variables (zonas de escaso contraste por la fuerte saturación de luces fluorescentes y otras casi sin iluminar) y la dificultad para detectar

una estructura por la forma y disposición de las presas hacen que resulte difícil garantizar una correcta detección por un método automático.

La figura 1.3 muestra ejemplos de los tipos de escenas del interior del barco en las cuales se apunta a realizar la detección. En otros casos, las escenas corresponden a la cubierta del barco, presentando otro tipo de desafíos.

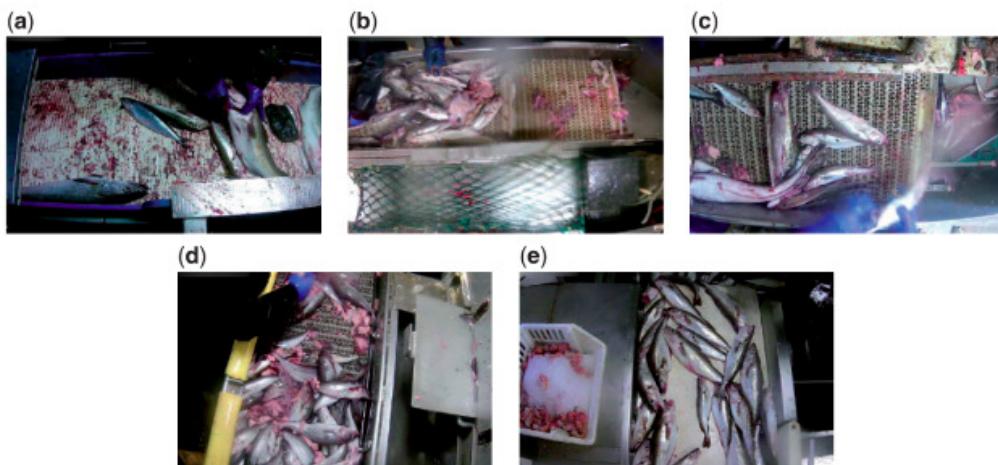


FIGURA 1.3. Ejemplos de imágenes de cámaras en el interior de distintos buques<sup>3</sup>.

La literatura consultada muestra que se pueden obtener resultados aceptables utilizando alguna variante de red convolucional (CNN, por sus siglas en inglés *Convolutional Neural Network*) [10]. Si bien no existe una restricción en la cantidad de neuronas mínima que deba tener una CNN, dado que existe una relación entre la cantidad de neuronas o “profundidad” de una red y su capacidad de reconocimiento de patrones en escenas complejas, es usual que las CNN utilizadas para este tipo de tareas cuenten con múltiples capas, cada una destinada a extraer patrones de un nivel de abstracción creciente. El inconveniente de las redes neuronales profundas es que requieren una gran cantidad de datos de entrenamiento, que en este caso (a diferencia de como ocurre en otros dominios, como el reconocimiento de personas, caras, vehículos, carteles, etc.) es difícil (o costoso) obtener.

## 1.4. Motivación y alcance

El resultado del relevamiento de aplicaciones de video analítico para pesca fue que en todos los casos se trataba de iniciativas de carácter académico o de prototipos pero que no están disponibles comercialmente, ni adecuados a las necesidades particulares de cada cliente que pueden incluir especies, características del barco según zona de pesca, tipo de presa, entre otras. La innovación de este trabajo reside en la aplicación del estado del arte de algunas de las técnicas mencionadas en un dominio para el cual no existen productos o soluciones en el mercado.

---

<sup>3</sup>Imagen adaptada de: <https://academic.oup.com/view-large/figure/206594116/fsz149f1.tif>. Accedido 10/09/2021.

En la tabla 1.2 se lista una selección de especies que se capturan en el mar argentino que han sido consideradas para este trabajo. Dependiendo de la temporada y la CMP de captura para cada barco las piezas que pueden ser admitidas en un caso pueden estar prohibidas en otro (por ejemplo, para permitir la reproducción). Cuando una especie no está permitida se debe contabilizar el descarte. Esto ocurre particularmente cuando el arte de pesca empleado no es selectivo, como ocurre con las redes de arrastre, que pueden contener capturas de especies protegidas.

La técnica de conteo y estimación a aplicar está fuertemente ligada al tipo de pieza. Para presas grandes, una solución puede ser utilizar un detector de objetos combinado con un algoritmo de seguimiento para contar las piezas que son depositadas en un recipiente, cinta transportadora, o arrojadas desde la cubierta, a modo de ejemplo. Para capturas pequeñas como langostinos o peces pequeños, puede ser conveniente utilizar un algoritmo regresivo que estime el tamaño de la red o, si esto resulta inviable, identifique los momentos en el video que requieren inspección de un experto.

TABLA 1.2. El tipo de arte asociada a la captura de cada especie sirve para determinar qué tipo de técnicas utilizar tanto para estimar la captura como el descarte. Esta información puede ser complementada con la de la flota de buques. Fuentes: INIDEP y FAO - Resumen Informativo sobre la Pesca por Países - Perfil de Argentina<sup>4</sup>.

Especie	Arte de pesca	Técnicas
Bonito	Red de cerco con y sin jareta.	Detección y conteo de piezas.
Centolla y centollón	Trampas y arrastre.	Detección y conteo de piezas.
Gatuzo	Red de arrastre de fondo.	Detección y conteo de piezas.
Langostino	Red de arrastre de fondo.	Estimación de volumen, detección de redes, búsqueda en video.

Se estableció como objetivo del trabajo sistematizar la solución de punto a punto para estimar la cantidad de piezas capturadas y descartadas en el caso de piezas de mayor tamaño, y de identificar los momentos del video en que aparece la red o se realizan tareas de clasificación cuando se trata de piezas pequeñas. Esto último porque dado que las cámaras pueden estar operativas durante días, tanto si se debe realizar una inspección manual como si se realiza mediante un algoritmo, identificar los intervalos de interés es un paso previo necesario. La estimación de volumen, no obstante, escapa al alcance de este trabajo.

<sup>4</sup>INIDEP. Sitio Oficial: <https://www.argentina.gob.ar/inidep> y FAO - Resumen Informativo sobre la Pesca por Países - Perfil de Argentina: <http://www.fao.org/fi/oldsite/FCP/es/ARG/profle.htm>. Accedidos 05/02/2021.

Como se trata de un desarrollo experimental pero que debe tener continuidad, se otorgó especial prioridad a dos aspectos que deben considerarse en el modo en que se implementa el software:

1. Debe ser posible presentar demostraciones técnicas desde las primeras semanas desde el inicio del proyecto, tanto para el cliente como para potenciales interesados. No es un requisito que sean realizadas con los datos finales, pero sí representativos del problema.
2. El software deberá estar organizado y diseñado de manera que sea escalable, mantenable, y pueda modificarse y extenderse para cada caso particular por desarrolladores y analistas de datos, tomando como ejemplo el flujo de trabajo propuesto por la librería Scikit-Learn [23].

#### 1.4.1. Requerimientos

A continuación se describen los requerimientos principales que se establecieron para que la solución pueda cumplir con el alcance propuesto:

1. Requerimientos generales
  - a) El sistema debe complementar una solución existente de múltiples cámaras de video para automatizar operaciones de conteo actualmente realizadas por operadores humanos.
2. Requerimientos funcionales
  - a) El sistema debe detectar regiones de la imagen que contengan objetos o entidades de interés, pudiendo ser ejemplos presas (pescados, mariscos y otros animales), operadores, redes de captura o instrumentos que puedan utilizarse para inferir alguna actividad como malacates, cintas o recipientes, entre otros. El tipo de objetos a detectar depende de cómo se haya configurado y entrenado el detector en cada caso, pudiendo ser una opción de implementación contar con múltiples modelos de detección y que por configuración se pueda elegir cual aplicar, dependiendo del objetivo.
  - b) El sistema debe poder computar la trayectoria de un objeto detectado, con el propósito de asociar estas trayectorias a eventos. Por ejemplo, incrementar contadores, o indicar que una captura o descarte ingresó a una zona de la imagen o pudiendo utilizarse esta información para determinar si un operador está manipulando una pieza.
  - c) El sistema debe registrar todos los eventos de interés en una base de datos para su posterior utilización con fines estadísticos. Estos eventos se definen mediante la detección de un tipo de entidad en escena y que la misma ingrese, transite, o egrese de una región de la imagen previamente establecida (las cámaras siempre estarán en una posición fija). De este modo pueden definirse eventos como:
    - Si una pieza está presente en una cinta transportadora.
    - Si una pieza es arrojada por un lado del barco (indicando un descarte).

- El tiempo que permanece una pieza en un recipiente de la embarcación.
- La cantidad de operadores en el barco, si están realizando una maniobra, etc.

### 3. Requerimientos de desempeño

- a) El procesamiento de video no puede estar por debajo de 1 cuadro procesado por segundo.
- b) El desempeño del detector debe ser equivalente o superior al de un operador humano medio, o en su defecto la detección debe cumplir como mínimo con un 70 % de mAP sobre un conjunto de datos de evaluación.

### 4. Requerimientos de interfaz

- a) No se requiere una interfaz gráfica, pero sí la adhesión a protocolos estandar para poder acceder a la información, por ejemplos: REST [24], XML [25] o JSON [26], entre otros.
- b) El sistema debe permitir indicar las regiones para cada cámara operativa en un archivo de configuración.

### 5. Requerimientos de ambiente

- a) Los componentes de procesamiento deberán ser servicios o microservicios y deben poder ejecutarse en un entorno Linux.
- b) La solución debe poder ejecutarse en una estación de trabajo, y con modificaciones menores poder correr, aunque sea con un desempeño degradado, en una plataforma embebida o *edge* [27]. Un ejemplo son los procesadores de la familia NVIDIA Jetson Xavier [28].

### 6. Restricciones de diseño/implementación

- a) El diseño debe ser modular y los componentes deben estar desacoplados, para poder realizar variaciones sobre la configuración según cada escenario.



## Capítulo 2

# Introducción específica

En este capítulo se presenta una reseña de las técnicas de visión por computadora e inteligencia artificial que han sido utilizadas en este trabajo, poniendo énfasis en las que influenciaron decisiones de diseño o que guardan mayor relación con el análisis de los ensayos. La primera parte del capítulo presenta algunos conceptos como tipos de aprendizaje y redes convolucionales, que son requeridos para una descripción de los algoritmos más avanzados sobre los que se apoya este trabajo: YOLO y DeepSORT.

### 2.1. Técnicas de visión por computadora e inteligencia artificial

La detección de objetos en la que se apoya el sistema propuesto es un problema de aprendizaje supervisado que combina clasificación y localización. La localización en imágenes es un problema de regresión donde la variable a predecir son las coordenadas del rectángulo que encierra el objeto de interés.

Las redes neuronales y en particular las redes convolucionales son utilizadas en casi todas las tareas de visión por computadora. Existen arquitecturas de redes especializadas para distintas tareas, como por ejemplo:

- Extracción de características.
- Evaluar similitud entre imágenes.
- Clasificación de imágenes.

La mayoría de los detectores de objetos basados en redes neuronales, incluyendo YOLO [29], tienen capas de convolución en su arquitectura.

Los algoritmos de seguimiento como DeepSORT [30] utilizan redes convolucionales para extraer un vector de características que permite agrupar por similaridad las salidas del detector de objetos, y así asociarlas a la trayectorias del objeto en seguimiento más probable.

A su vez, el entrenamiento de este extractor de características utiliza una arquitectura de redes neuronales denominada “redes siamesas” en la que se define una función de costo para dar prioridad al aprendizaje de las similitudes (o diferencias) entre imágenes [31, 32].

El algoritmo DeepSORT es una mejora de SORT [33]. Ambos utilizan un estimador de movimiento con el filtro de Kalman para obtener trayectorias y resuelven

el problema de asociación de detecciones entre cuadros mediante el algoritmo húngaro. Existen variaciones y algoritmos de seguimiento posteriores que realizan la estimación con redes neuronales recurrentes y redes *long short-term memory* (LSTM) [34, 35]. Estas variaciones pueden mejorar la estimación, pero requieren entrenar modelos supervisados, para lo que, entre otras cosas, se debe disponer de suficientes datos etiquetados.

## 2.2. Aprendizaje supervisado y no supervisado

Excluyendo el aprendizaje por refuerzo, el aprendizaje automático puede dividirse en dos grandes tipos: supervisado y no supervisado [36]. En ambos grupos se dispone de datos de entrada, pero sólo en el aprendizaje supervisado se cuenta además con las salidas correspondientes a cada ejemplo que pueden ser variables categóricas en un problema de clasificación, numéricas en un problema de regresión, o una combinación de ambas en un problema de aprendizaje múltiple, como se describe en 2.4.7.

En un problema de aprendizaje no supervisado, al no contar con una variable de salida conocida, el objetivo de entrenamiento de un modelo es poder agrupar los datos por afinidad, hallar asociaciones entre ellos o encontrar representaciones compactas mediante reducción de dimensiones.

En este trabajo se han desarrollado distintos modelos de aprendizaje supervisado para la clasificación de actividades y se han entrenado detectores de objetos de arquitectura YOLO, que son un ejemplo de aprendizaje supervisado de regresión, clasificación binaria y clasificación multiclas.

También se han desarrollado modelos de aprendizaje no supervisado de extracción de características, cuyo objetivo es hallar una representación compacta de una imagen que, utilizando la función de distancia coseno, exprese qué tanto se parecen (o difieren) dos imágenes.

### 2.2.1. Problema de regresión

Los problemas de regresión consisten en determinar, dada una muestra y una o más variables numéricas de salida, los valores que mejor se ajustan a los que se observaron en los ejemplos. En localización de objetos, por ejemplo, dada una imagen conteniendo un objeto en alguna región rectangular, la variable de salida es un vector que representa el centro y las dimensiones de la caja que lo contiene:  $[x_c, y_c, w, h]$ . Si bien se verá más adelante que no alcanza con describir únicamente la región de la imagen en que se halla un objeto para poder localizarlo, este aspecto del problema puede ser modelado como un problema de regresión.

### 2.2.2. Problema de clasificación

Los problemas de clasificación consisten en determinar, dada una muestra y un conjunto de clases que pueden representarla, cuál es la clase que mejor la representa.

Un caso particular es la clasificación binaria en el cuál la salida puede tomar sólo dos valores posibles. Se suele adoptar la convención de llamar a la primera clase “cero” o “falso” y a la segunda clase “uno” o “verdadero”.

### Evaluación de un clasificador binario

En los problemas de clasificación binaria se suelen usar los indicadores de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos (TP, FP, TN y FN, por sus siglas en inglés) para comparar los resultados de un modelo con los resultados esperados [36]. En la mayoría de los casos, la salida de un clasificador es una probabilidad indicando la pertenencia a una clase. Se debe, por lo tanto, definir un valor arbitrario  $\alpha$  de modo que:

$$C_{pred} = \begin{cases} 0 & \text{si } P(X = 1) < \alpha \\ 1 & \text{si } P(X = 1) \geq \alpha \end{cases} \quad (2.1)$$

Si  $C_{pred}$  es la clase predicha y  $C_{GT}$  es la clase esperada entonces:

- Un verdadero positivo es cuando:  $C_{pred} = C_{GT} = 1$ .
- Un verdadero negativo es cuando:  $C_{pred} = C_{GT} = 0$ .
- Un falso positivo es cuando:  $C_{pred} = 1, C_{GT} = 0$ .
- Un falso negativo es cuando:  $C_{pred} = 0, C_{GT} = 1$ .

A partir de estos indicadores, se definen [36]:

- Accuracy
- Precision
- Recall
- Curva ROC
- AUC
- F1-Score
- Matriz de confusión

En algunos casos estas métricas y representaciones dependen de haber establecido el valor  $\alpha$  y en otros casos la métrica es independiente del mismo.

### Accuracy

Es una métrica asociada a la tasa de observaciones correctas sobre total de observaciones [36]. Está dada por:

$$acc = \frac{TP + TN}{TP + FP + FN + TN} \quad (2.2)$$

Esta métrica es sensible a la proporción de las clases, por lo que si el conjunto de datos de evaluación no está balanceado, como ocurre en algunos de los utilizados en este trabajo, es probable que se realice una interpretación errónea del desempeño del algoritmo. Por este motivo no se ha utilizado esta métrica.

### Precision

Es una métrica asociada a una baja tasa de falsos positivos [36]. Está dada por:

$$\text{precision} = \frac{TP}{TP + FP} \quad (2.3)$$

Es utilizada para construir una de las métricas más utilizadas en detección de objetos: *mean average precision* (mAP).

### Recall

Esta métrica mide la cantidad de predicciones correctas para cada clase [36]. Por ejemplo: de todos los casos de presencia de una especie protegida, ¿cuántos fueron correctamente identificados?

$$\text{recall} = \frac{TP}{TP + FN} \quad (2.4)$$

Debe ser considerada en la etapa de selección de algoritmos, cuando no es un requerimiento reducir al mínimo posible los falsos negativos. Al igual que la anterior, es utilizada para construir la métrica mAP.

### Curva ROC

La curva ROC (acrónimo de *Receiver Operating Characteristic*) indica qué tan capaz es un modelo de distinguir clases relacionando las métricas de *precision* y *recall* para un intervalo que incluye todos los valores de  $\alpha$  [36].

La métrica AUC (por sus siglas en inglés *Area Under Curve* es el área bajo la curva ROC, y es un indicador de qué tan bueno es un clasificador independientemente del umbral de clasificación elegido.

### F1-Score

Esta métrica es un promedio entre *precision* y *recall* [36]:

$$f1 = 2 \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \quad (2.5)$$

### Matriz de confusión

Es una forma de representar visualmente para cada clase TP, TN, FP y FN en un problema de clasificación binaria, o cuantas instancias correctas e incorrectas fueron halladas para cada clase en un problema de clasificación multiclas [36].

## 2.3. Redes neuronales con capas convolucionales

Una red neuronal artificial prealimentada o *feed-forward neural network* es un grupo interconectado de nodos en los que la información fluye desde una entrada hacia una salida, sin ciclos. Cada nodo recibe múltiples entradas a las cuales asigna un peso y luego aplica una función, llamada de activación. La salida de cada neurona es luego propagada a neuronas de capas posteriores. El nombre se debe a que el concepto tiene semejanza con las neuronas biológicas [37].

El objetivo de entrenamiento de una red neuronal es hallar los pesos óptimos que minimicen un objetivo de entrenamiento que se expresa a partir de la minimización de una función de costo asociada a una tarea de aprendizaje automático, por ejemplo clasificación o regresión.

En una red prealimentada las neuronas se organizan en capas sucesivas desde la entrada hacia la salida. Las capas pueden contener nodos que ejecutan funciones de activación orientadas para cada propósito. Funciones típicas son sigmoidal, tangente hiperbólica, rectificador (RELU), operaciones de convolución y reducción (*pooling*), éstas últimas de especial utilidad para reeconomicimiento de patrones en imágenes y otras señales de una, dos o más dimensiones.

A las redes que contienen capas con nodos que realizan operaciones de convolución como la que se muestra en 2.1 se las denomina *Convolutional Neural Networks* (CNN). Cada nodo funciona como un filtro sensible a características o patrones de la entrada como bordes o esquinas. Los coeficientes que se aprenden, en este caso, son los pesos de cada filtro. A las capas convolucionales suelen suceder las operaciones de *pooling* que reducen la dimensión de las salidas, para también reducir la cantidad de nodos de las capas posteriores de la red.

## 2.4. Detección de objetos

La detección de objetos es la tarea de localizar y clasificar objetos en una imagen, delimitándolos con una región poligonal (si bien en la mayoría de los casos se utiliza un rectángulo) [38].

Es importante establecer la diferencia entre clasificación, localización, detección y distintos tipos de segmentado, que se ejemplifican en la figura 2.2.

- Clasificación: dado un conjunto finito de etiquetas (clases), asignar a una imagen la etiqueta de la clase que mejor la representa.

---

<sup>1</sup>Imagen obtenida de <https://www.codificandobits.com/blog/tutorial-clasificacion-imagenes-redes-convolucionales-python/>. Accedido 15/09/2021.

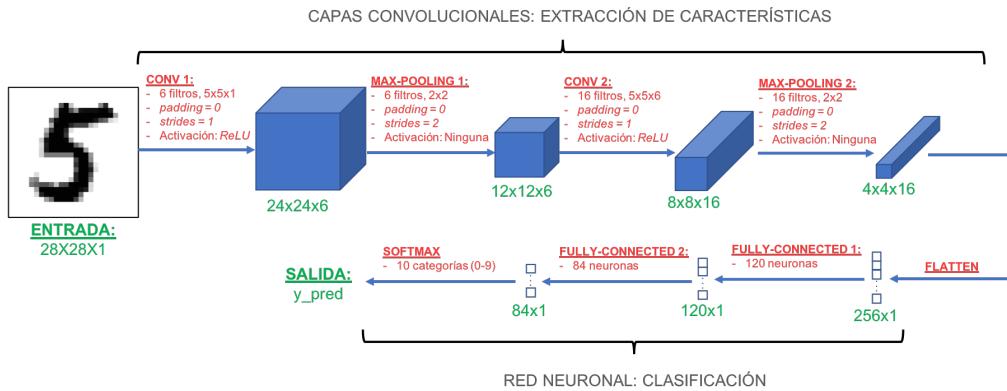


FIGURA 2.1. Diagrama simplificado de una red neuronal con capa convolucional. Los filtros aprendidos por la red extraen patrones de la imagen que luego pueden ser representados en menos dimensiones, reduciendo la cantidad de neuronas necesarias en capas posteriores <sup>1</sup>.

- Clasificación y localización: delimitar en una imagen la región rectangular en la que aparece un objeto de una clase.
- Detección de objetos: se utiliza este término para el mismo problema anterior, pero en este caso hallando múltiples objetos en una imagen, cada uno indicado por una región rectangular y su probabilidad de pertenencia a una clase.
- Segmentado de instancias: identificar en una imagen instancias de objetos y asignarlas a la clase de ese objeto. A diferencia del caso anterior, la asignación es a nivel de píxel en lugar de delimitar los objetos con regiones rectangulares.
- Segmentado semántico: identificar en una imagen tipos de objetos y asignarles una clase. La asignación también es a nivel de píxel como en el caso anterior, pero en este caso no se distinguen las instancias de los objetos. Es decir, si por ejemplo en una imagen hay múltiples vehículos, a todos ellos se asigna la clase “vehículo” en lugar de “vehículo1, vehículo2, ...”.



FIGURA 2.2. Ejemplos tareas de clasificación, localización, detección y segmentado de objetos <sup>2</sup>.

#### 2.4.1. Modelo de caja negra

Independientemente de la técnica utilizada para la detección de objetos, es posible -y útil- plantear un modelo de caja negra, en el que dada una imagen de

entrada de color RGB expresada como un tensor de dimensiones  $(W, H, 3)$ , se obtiene como salida una tupla de objetos detectados, donde para cada detección se indica:

- Probabilidad de pertenencia a cada clase.
- Coordenadas de la región rectangular que contiene el objeto.

#### 2.4.2. Evaluación de modelos

Es de interés establecer métricas para comparar el desempeño de distintos modelos de detección de objetos independientemente de su funcionamiento interno.

Dado que la salida de un detector de objetos es una tupla con detecciones indicadas por regiones rectangulares y la clase correspondiente a cada una, conociendo las salidas esperadas se puede plantear la diferencia entre las regiones rectangulares predichas por el modelo y las regiones rectangulares esperadas.

#### 2.4.3. Intersección sobre unión

Si  $A_{pred}$  y  $A_{GT}$  son las áreas correspondientes a la predicción del modelo y la región real respectivamente ( $GT$  por las siglas en inglés para *Ground Truth*), se define la intersección sobre unión  $IoU$  (por sus siglas en inglés *Intersection over Union*) como el cociente entre el área superpuesta de ambas regiones sobre su área total [38].

$$IoU = \frac{A_{pred} \cap A_{GT}}{A_{pred} \cup A_{GT}} \quad (2.6)$$

En el momento de inferencia cuando  $IoU$  es cero significa que no hay ninguna intersección y por lo tanto el modelo falló completamente en predecir la presencia de ese objeto, y cuando  $IoU$  es uno significa que el objeto fue detectado con la máxima precisión posible.

#### 2.4.4. Métricas de evaluación

En el apartado 2.2.2 se definieron indicadores y métricas para especificar el desempeño de un clasificador binario.

Estos indicadores pueden aplicarse para el problema de detección de objetos utilizando las salidas del detector y los valores esperados. Se definen:

- $C_{pred}$ : clase predicha por el detector.
- $C_{GT}$ : clase esperada.
- $A_{pred}, A_{GT}, IoU$ : área de la predicción, área real, e intersección sobre unión de ambas.

---

<sup>2</sup>Imagen adaptada de: [https://www.researchgate.net/figure/Comparison-of-semantic-segmentation-classification-and-localization-object-detection\\_fig1\\_334363440](https://www.researchgate.net/figure/Comparison-of-semantic-segmentation-classification-and-localization-object-detection_fig1_334363440). Accedido 13/09/2021.

Se atribuye una detección como correcta (verdadero positivo) si hay coincidencia entre la región de la predicción y la de resultados esperados para la misma clase, y si  $IoU$  supera un umbral  $\alpha$  con un valor arbitrario, como por ejemplo  $\alpha = 0,5$ .

- Un verdadero positivo (TP) es cuando:

$$C_{pred} = C_{GT} \text{ y } IoU > \alpha \quad (2.7)$$

- Un falso positivo (FP) es cuando:

$$C_{pred} \neq C_{GT} \text{ } IoU \geq \alpha \quad (2.8)$$

- Un falso negativo (FN) es cuando:

$$IoU < \alpha \quad (2.9)$$

La figura 2.3 muestra visualmente ejemplos de los tres casos. En detección de objetos no se utiliza verdadero negativo (TN) porque existen infinitas posibilidades.

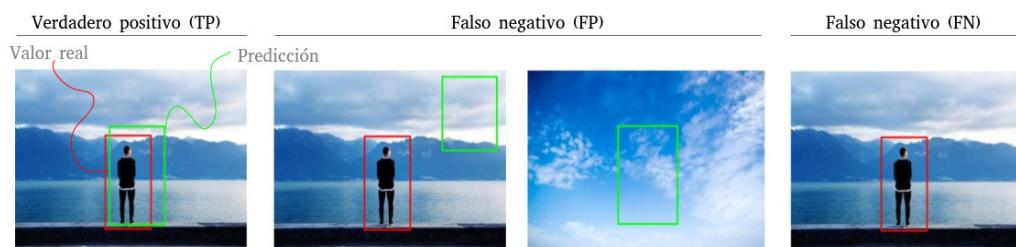


FIGURA 2.3. Definición de TP, FP y FN para detección de objetos <sup>3</sup>.

Con estas definiciones es posible construir las métricas *precision* y *recall* mencionadas en el apartado 2.2.2, que en detección de objetos pueden interpretarse del siguiente modo:

- Un valor elevado de *recall* pero bajo de *precision* implica que todos los objetos reales fueron detectados, pero con muchas detecciones incorrectas (alto porcentaje de falsos positivos).
- Un valor bajo de *recall* pero elevado de *precision* indica que muchos de los objetos reales fueron detectados correctamente, pero hubo otros que no fueron detectados (alto porcentaje de falsos negativos).

Un valor elevado de *recall* y de *precision* indica que todos los objetos fueron detectados correctamente, siendo este el objetivo de un detector ideal.

### *Mean average precision (mAP)*

En un problema de detección de objetos deben evaluarse simultáneamente la capacidad del modelo de localizar y de clasificar. Una de las métricas utilizadas

---

<sup>3</sup>Imagen adaptada de: <https://manaleaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html>. Accedido 13/09/2021.

para evaluar el desempeño de un detector de objetos es *mean average precision* (mAP) [39] que se obtiene calculando primero AP para cada clase con el siguiente procedimiento:

1. Se establece un valor fijo de  $\alpha$  para determinar si se toma una detección como válida a partir de su *IoU*.
2. Se calculan *precision* y *recall* para cada una de las muestras.
3. Se ordenan en sentido decreciente de confianza en el resultado, es decir, de mayor a menor *precision* y con *recall* incrementándose. La figura 2.4 es un ejemplo de la curva resultante.
4. La curva suele adquirir una forma dentada, consecuencia de que algunas de las predicciones sean correctas (*recall* creciente) y otras incorrectas (*precision* descendente). Para evitar la forma dentada se suele tomar el valor más alto hacia la derecha y extenderlo hasta el flanco descendente anterior.
5. Finalmente, AP es el área bajo la curva, es decir:  $AP = \int_0^1 p(r)dr$ .

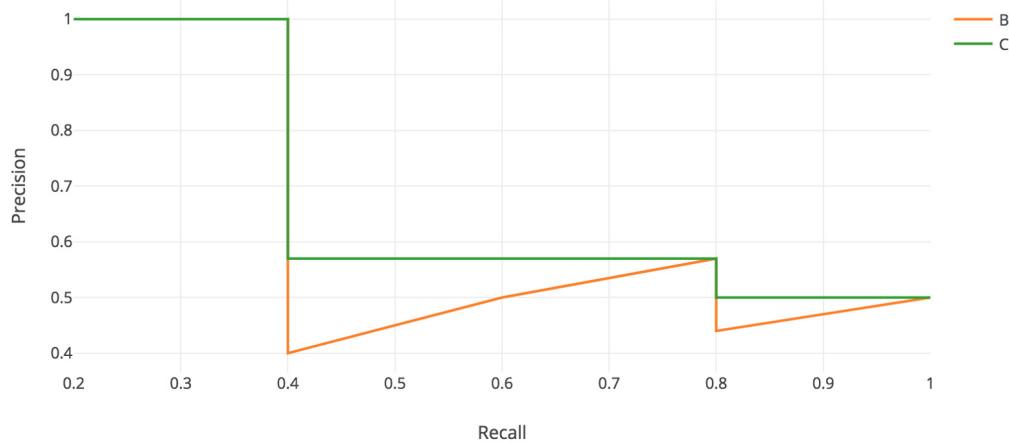


FIGURA 2.4. Curva de *precision-recall* para cálculo de mAP en detección de objetos antes y después de suavizado <sup>4</sup>.

La métrica mAP se obtiene de promediar el valor de AP obtenido para cada imagen y clase. Existen diferentes criterios para la elección del umbral de IoU, la cantidad de puntos a utilizar para interpolar en el suavizado de la curva, o como se realiza el promedio final. Algunos ejemplos son las que se utilizan en PascalVOC [40] y COCO [41].

### 2.4.5. Arquitecturas de detectores

Los detectores de objetos pueden agruparse en dos familias de algoritmos [42]:

- De dos etapas: los que realizan una búsqueda previa de regiones de interés y luego aplican un clasificador para cada una de esas regiones. Ejemplos son la familia de algoritmos publicados con el título *Rich feature hierarchies for accurate object detection and semantic segmentation* (R-CNN) [43], *Fast R-CNN* [44] y *Faster R-CNN* [45], entre otros.
- De una etapa: los que analizan la imagen de entrada en una única pasada. Ejemplos son *Single Shot MultiBox Detector* (SSD) [46] y la familia de algoritmos YOLO (acrónimo de *You Only Look Once*) [29].

### 2.4.6. El algoritmo YOLO

El algoritmo YOLO es un detector de objetos de una etapa que fue evolucionando con el tiempo, siendo su cuarta versión la elegida para este trabajo. Cada versión incorpora una mejora sobre la anterior, pero conservando su principio de funcionamiento.

En términos generales, el algoritmo YOLOv1 aplica las siguientes ideas:

1. Dividir la imagen de entrada en  $S \times S$  celdas. Para cada celda se debe predecir si existe un objeto representado en ella. Este puede ser más grande que la celda y cubrir otras celdas, pero no puede haber más de un objeto por celda. Para cada celda hay una cantidad finita de rectángulos  $B$  (por el término en inglés *bounding box*) candidatos a contener un objeto representado por esa celda.
2. Para cada uno de los rectángulos, existen cuatro valores  $(x_c, y_c, w, h)$  que representan la posición de su centro, ancho y alto. La posición del centro se expresa de manera relativa a la posición de la celda, no de la imagen.
3. Para cada rectángulo además existe un puntaje  $Pr(Obj)$  que refleja qué tan probable es que contenga un objeto (en la publicación se le denomina *objectness*). Durante el entrenamiento, este valor se calcula como  $Pr(Obj) \cdot IoU$ , donde  $IoU$  es la intersección sobre la unión entre la celda y cada rectángulo de los datos de entrenamiento.

#### Supresión de no máximos

Si un único objeto ocupa más de una celda, es posible que se tengan múltiples regiones para el mismo objeto, por lo tanto, se debe elegir la que mejor lo representa. Para ello, se sigue el siguiente procedimiento:

1. Se descartan todas las celdas en las que la probabilidad de presencia de un objeto sea menor que un umbral (típicamente 0.6)

---

<sup>4</sup>Imagen adaptada de: <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>. Accedido 16/09/2021.

2. Se consideran todas las celdas con la probabilidad más alta de incluir un objeto.
3. Se toman los regiones que tienen mayor puntaje y se remueven los que tienen  $IoU$  mayor que un umbral determinado entre sí (tipicamente 0.5). Eso elimina las regiones (*bounding boxes*) similares.

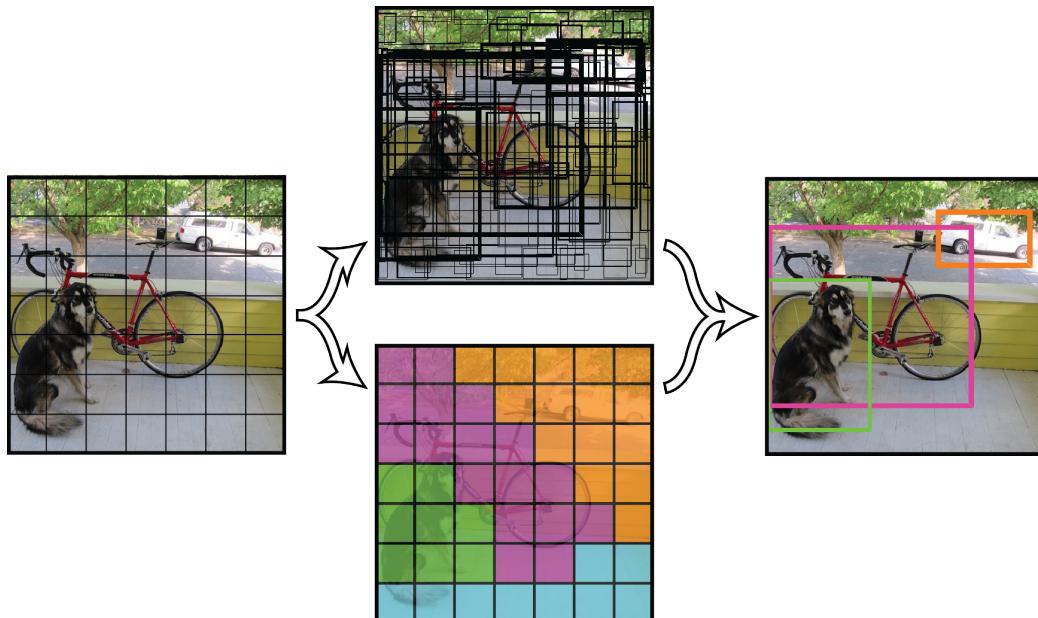


FIGURA 2.5. En el algoritmo YOLO, una imagen se divide en grillas y por cada celda se clasifica el objeto que mejor representa y sus regiones rectangulares asociadas. En el ejemplo las regiones con menor puntaje son eliminadas, quedando finalmente tres detecciones<sup>5</sup>.

#### 2.4.7. La detección de objetos como un problema de aprendizaje multitarea

Se definió en el apartado 2.4 que la salida esperada para un problema de detección es una tupla que debe contener el resultado de la localización y de la detección para cada objeto. En YOLO, esa información se extresa con una tupla donde cada detección tiene tres componentes [38, 47]:

- Una variable binaria indicando si se ha detectado un objeto. Si su valor es verdadero, son válidas la región y vector indicando probabilidad de clase. De lo contrario deben ignorarse para esa entrada.
- Una región rectangular  $(x_c, y_c, w, h)$  que encierra el objeto detectado.
- Un vector de  $n$  elementos, siendo  $n$  el número de clases a reconocer, indicando la probabilidad de que el objeto detectado pertenezca a cada una de las  $n$  clases. La suma de los elementos de este vector debe ser 1.

Esta salida puede plantearse como un problema compuesto:

<sup>5</sup>Página de YOLOv1 en sitio oficial de Darknet. <https://pjreddie.com/darknet/yolov1/>. Accedido 13/09/2021.

- Regresión logística: determinar si hay un objeto presente.
- Clasificación multiclasa: la probabilidad de que el objeto pertenezca a la clase  $C_i$ .
- Regresión lineal: hallar la región rectangular para esa detección.

A continuación se describirán con mayor detalle las funciones de costo (*loss*) asociadas a estos tres objetivos de entrenamiento.

### Función de costo de clasificación

Esta función está dada por:

$$\sum_{i=0}^{S^2} I_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (2.10)$$

donde:

- $I_i^{obj}$  indica si existe un objeto en la celda  $i$ . 1=Sí, 0=No.
- $\hat{p}_i(c)$  probabilidad de pertenencia a la clase  $c$  de la celda  $i$ .

### Función de costo de localización

Esta función mide el error entre las regiones rectangulares que se predicen respecto a las que están dadas en los datos de entrenamiento, y está dada por:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B I_i^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B I_i^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \quad (2.11)$$

donde:

- $S$  es la cantidad de divisiones de la imagen de entrada, siendo  $S^2$  la cantidad total de celdas.
- $B$  es la cantidad de rectángulos por celda.

### Función de costo de presencia de objeto

Esta función mide el error al decidir si un objeto está presente en una celda.

$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{noobj} (C_i - \hat{C}_i)^2 \quad (2.12)$$

donde:

- $\hat{C}_i$  es la confianza en la predicción para la celda  $j$  en la celda  $i$ .

- $I_{ij}^{obj}$  es 1 si el  $j$ -ésimo rectángulo de la celda  $i$  contiene un objeto. No obstante, como en la mayoría de los casos la celda no contiene objetos, es importante que el algoritmo no aprenda a identificar el fondo como un objeto. Para eso se agrega el término  $\lambda_{noobj}$  y se utiliza  $I_{ij}^{noobj}$  en lugar de  $I_{ij}^{obj}$ .

### Función de costo total

Combinando las ecuaciones 2.10, 2.11 y 2.12 se obtiene:

$$\begin{aligned}
 L = & \sum_{i=0}^{S^2} I_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B I_i^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \\
 & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B I_i^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{noobj} (C_i - \hat{C}_i)^2
 \end{aligned} \tag{2.13}$$

### 2.4.8. Evolución de la familia de algoritmos YOLO

El algoritmo YOLO fue mejorando sus capacidades de detección desde su publicación original hasta las más recientes, penalizando en algunos casos el tiempo de inferencia. Sin entrar en todos los detalles específicos de cada versión, se consideran para este trabajo dos aspectos. El primero es la capacidad de manejar objetos pequeños, superpuestos y apilados, y el segundo es cómo se ven afectados los tiempos de inferencia en las versiones más nuevas.

#### YOLOv2

YOLOv2 (también conocido como YOLO9000) [48], mejora la detección de objetos pequeños o muy próximos, que es una de las limitaciones de YOLOv1.

Este problema se resuelve parcialmente con la introducción del concepto de *anchor boxes*, que son rectángulos asociados a las detecciones, inicializados a partir de un archivo de configuración en lugar de ser aprendidos desde cero. De este modo, los rectángulos hallados se expresan como desviaciones de los rectángulos iniciales. Los autores recomiendan elegir los *anchor boxes* de acuerdo a las dimensiones del objeto que se quiere predecir según el tipo de problema.

Otras mejoras del algoritmo incluyen cambios en la estructura de la red.

Pese a incluir los *anchor boxes*, esta versión sigue presentando dificultades en detectar objetos pequeños, comparada con otros algoritmos como los de la familia R-CNN.

#### YOLOv3

YOLOv3 [49] incorpora una arquitectura de red más profunda e introduce mejoras significativas en el desempeño, pero penalizando el tiempo de inferencia. Una

de particular interés para este trabajo, en que se deben reconocer objetos pequeños que pueden estar apilados, es que la definición de *anchor boxes* se amplía para objetos de distinta escala, además de permitir detectar múltiples objetos en una misma celda.

### YOLOv4

YOLOv4 [50] se publicó en abril de 2020. Entre sus principales ventajas está una mejora de hasta un 10 % en mAP respecto a la versión anterior y en algunas configuraciones, como por ejemplo si se utiliza un tamaño de celda de entrada de 416 o 512 píxeles, una tasa de cuadros por segundo (FPS) por encima de los treinta cuadros con una GPU RTX2070 o similar. La figura 2.6 compara su mAP y FPS con otros algoritmos del estado del arte en el momento de publicación del trabajo.

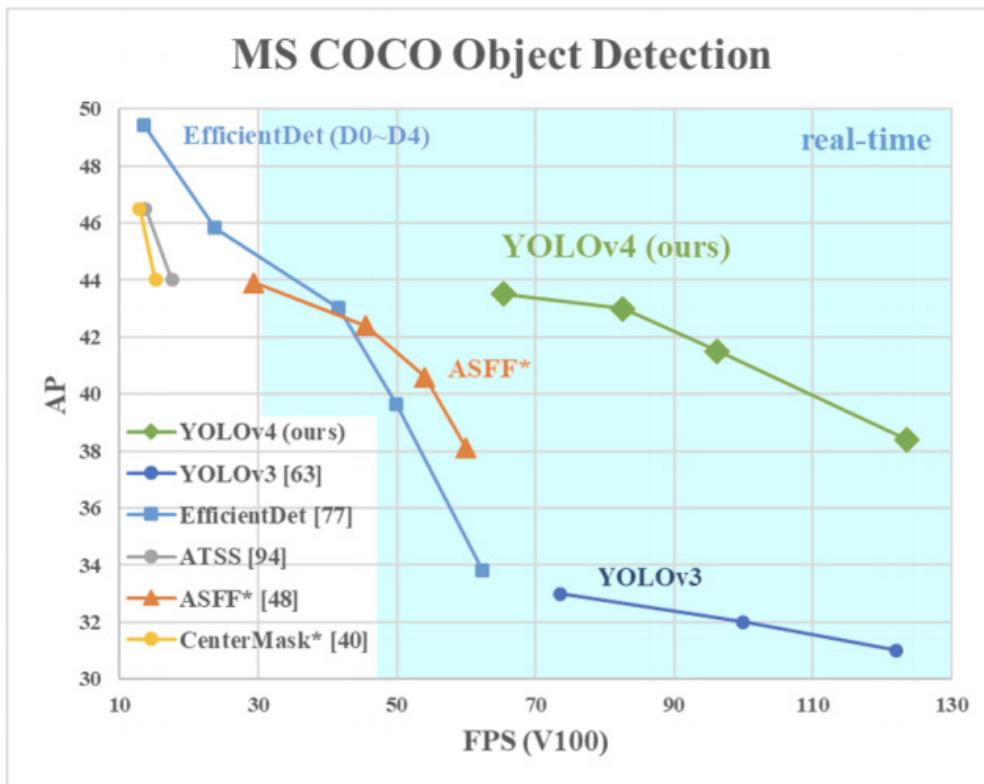


FIGURA 2.6. Comparativa entre YOLOv4 y otros detectores de objetos en abril de 2020. YOLOv4 tiene una mejora de desempeño de aproximadamente un 10 % respecto a YOLOv3, y sin dejar de ser apto para tiempo real<sup>6</sup>.

<sup>6</sup>Imagen obtenida de la publicación original del autor: <https://arxiv.org/pdf/2004.10934.pdf>. Accedido 13/09/2021.

## 2.5. Seguimiento de objetos

El seguimiento de objetos es el proceso de localizar objetos que se mueven en un video, manteniendo sus identidades entre los sucesivos cuadros para poder generar sus trayectorias [51]. No obstante, además de la generación de trayectorias, existen otros motivos por los cuales puede ser de utilidad implementar seguimiento de objetos.

Una restricción de los detectores de objetos es que pueden ser computacionalmente exigentes. Si interesa conocer la posición de uno o más objetos en todo momento pero no se dispone de suficientes recursos de cómputo para ejecutar el detector en cada cuadro de video, se puede reducir la frecuencia de uso del detector a una vez cada cierta cantidad de cuadros y recurrir al seguimiento para los cuadros intermedios.

Otro motivo es que los detectores dependen de la apariencia del objeto, que puede presentar dificultades en algunos cuadros, siendo algunos motivos:

- Oclusión: un objeto está total o parcialmente visible porque está detrás de otro.
- Cambio de identidad: dos objetos similares se entrecruzan. El detector no puede establecer cual es cual luego del intercambio.
- Cambio de punto de vista: un objeto tiene un aspecto muy diferente desde un determinado ángulo y el detector no lo reconoce.
- Imagen borrosa: una distorsión vuelve al objeto irreconocible durante un período de tiempo o en una zona de la imagen (por ejemplo, si el lente de la cámara fue impactado por gotas de agua).
- Cambio de escala: un objeto que se aproxima o aleja mucho del campo de visión puede dejar de ser reconocido por el detector.
- Confusión con el fondo: alguna característica del fondo puede confundirse con el objeto y el detector ser incapaz de distinguirlo.
- Variaciones de iluminación: un exceso o falta de iluminación puede hacer difícil de distinguir un objeto en determinados momentos o zonas de una imagen.
- Baja resolución: un objeto puede ser visualmente difícil de reconocer excepto por su movimiento.

### 2.5.1. Técnicas de seguimiento de objetos

Existen múltiples técnicas de seguimiento de objetos. Una forma de agruparlas es según su alcance y condiciones de operación:

- Por cantidad de objetos (abreviadas SOT y MOT por sus siglas en inglés *single/multiple object tracking*).
- Por el modo de procesamiento, que puede ser en vivo (*online*) o en diferido (*offline*). En el primer caso solo se puede utilizar información del cuadro actual y los anteriores mientras que en el segundo se puede acceder a la secuencia completa de cuadros.

- Por su dependencia de un detector de objetos: abreviadas DBT y DFT por sus siglas en inglés *detection-based tracking* y *detection-free tracking*. En el segundo caso el seguimiento está restringido a una cantidad fija de objetos establecida en la inicialización del algoritmo, mientras que los métodos basados en detectores pueden descubrir nuevos objetos que aparecen o adecuarse a objetos que desaparecen, pero requieren que un detector sea entrenado para reconocerlos.

Para este trabajo interesan únicamente las técnicas de seguimiento de múltiples objetos cuya cantidad pueda variar en el tiempo y que puedan aplicarse en tiempo real.

### 2.5.2. Seguimiento de múltiples objetos con SORT y DeepSORT

En la solución de un problema de seguimiento de múltiples objetos, dos grandes aspectos deben ser considerados. Uno de ellos es medir la similaridad entre objetos de distintos cuadros y el otro es poder recuperar la información de la identidad usando la similaridad entre objetos. El primer problema involucra el modelado de la apariencia, el movimiento, la interacción, la exclusión, y la ocultación. El segundo es un problema de inferencia.

No todos los algoritmos modelan todos los aspectos del primer problema. El algoritmo SORT, por ejemplo, sólo incorpora un modelo de movimiento, mientras que el algoritmo DeepSORT lo extiende con un modelo de apariencia que recurre a un extracto de características con una red convolucional.

#### Modelo de movimiento y filtro de Kalman

El modelo de movimiento debe representar el comportamiento dinámico de un objeto, permitiendo estimar su posición en cuadros futuros. En la mayoría de los casos, se asume que los objetos se desplazan suavemente en el mundo real y por lo tanto en el espacio de la imagen. También es relevante si los objetos se desplazan sobre un fondo fijo o no. Para las aplicaciones que se consideran en este trabajo siempre se asume que la cámara tiene una posición y orientación fija.

Los objetos que se presentan en una imagen son en realidad proyecciones de objetos que se desplazan en un espacio tridimensional. Para los algoritmos que se describen se tratan las proyecciones de los objetos en el plano como objetos en sí mismos que quedarán representados por un rectángulo que tiene una posición, ancho, alto y velocidad. La unidad de distancia adoptada es el *píxel*.

Tanto SORT como DeepSORT utilizan un filtro de Kalman con un modelo lineal de la velocidad entre cuadros. Cuando se asocia una detección con un objeto en seguimiento, se utiliza la detección para actualizar su estado. Si no existe una detección, entonces el estado se actualiza con la predicción sin aplicar ninguna corrección.

### Modelo de apariencia

El modelo de apariencia debe representar el aspecto visual de un objeto, capturando aquellas características que permitan distinguirlo de otros. Debe existir también una forma que permita medir la apariencia entre los objetos en seguimiento y determinar de manera cuantitativa cuales se parecen más entre sí.

### Problema de asociación y algoritmo húngaro

El segundo problema del seguimiento de múltiples objetos es asignar las observaciones actuales a trayectorias anteriores de objetos en seguimiento.

Idealmente, se quiere asignar cada nueva observación a la trayectoria más probable de las existentes, o iniciar una nueva trayectoria si la posición de la nueva detección está suficientemente alejada de las existentes y es atribuible a la apariación de un nuevo objeto en escena.

Este concepto puede abordarse definiendo un criterio de costo que debe ser mínimo cuando la posición de la detección coincide con la trayectoria esperada, y elevado cuando no guarde similitud.

Una estrategia de asignación óptima asociaría una objeto en seguimiento a cada nueva observación procurando minimizar el costo total, y estableciendo un costo máximo admisible para cada asignación. Si ese costo se supera, entonces se trata a la nueva observación como un objeto nuevo o un error.

Una forma de resolver el problema de asignación, que utilizan SORT y Deep-SORT, es mediante el algoritmo húngaro [52], que modela el problema como una matriz de costo  $C$  de  $n \times m$  donde las filas y las columnas representan los elementos a asignar, es decir, cada celda  $c_{ij}$  representa el costo de asignar el elemento  $i$  al  $j$ .

El costo es una función que se define a partir de la distancia en *pixels*, en el caso de SORT, y a la que se suma la distancia en apariencia, en el caso de DeepSORT.



## Capítulo 3

# Diseño e implementación

En este capítulo se resumen las decisiones de diseño del sistema, comenzando por un esquema de alto nivel que lo presenta en términos de bloques funcionales y tipos de productos, ambientes y procesos involucrados en su desarrollo. Luego se describen los procesos de generación de modelos de aprendizaje automático y finalmente se presenta el diseño del *framework* de componentes que permite integrar los modelos en la solución final, al que se llamó Videoanalytics.

### 3.1. Arquitectura del sistema

Para cumplir con los requerimientos de la propuesta y habiendo orientado la solución a capitalizar el relevamiento de herramientas y tecnologías referidas en el capítulo anterior, se realizó un diseño preliminar del sistema con los bloques que se muestran en la figura 3.1.

Estos bloques debían cumplir las siguientes funciones:

- Adquisición de video: obtener los cuadros de una cámara o un archivo de video y aplicar una corrección de imagen para reducir las diferencias entre cámaras. Esta corrección puede incluir: contrarrestar distorsión de lente de pez, aplicar transformaciones de coordenadas cromáticas o ecualizar la imagen, entre otras.
- Detección: implementar un detector con el algoritmo YOLOv4 o variantes del mismo.
- Seguimiento: construir trayectorias de los objetos detectados. Como mínimo debía implementarse para el algoritmo DeepSORT. La salida de este algoritmo es una trayectoria de un objeto.
- Incorporación de información de contexto (opcional): este bloque permitiría utilizar información de localización del buque, fecha y hora, actividad de sistemas mecánicos y otros datos tanto para acompañar la salida del clasificador como para mejorarla, dado que hay especies que habitan determinados sectores o se capturan a determinada profundidad e intervienen sistemas de captura específicos para un tipo de objetivo.
- Filtrado, registro y generación de reportes: la salida de la etapa anterior sería procesada aplicándole filtros para extraer sólo los eventos de interés o aumentar la confianza de las predicciones. Un ejemplo es eliminar las predicciones por debajo de un umbral de confianza, o retener únicamente la

actividad de algunos objetos específicos en algunas regiones. Esto produciría un registro con horarios, tipos de capturas o cantidad de descartes, entre otros. Cada reporte sería publicado en una base de datos. Para facilitar el consumo de esta información y permitir combinarla con otros servicios, se utilizaría una base de datos con interfaz REST compatible con herramientas de visualización y consulta.

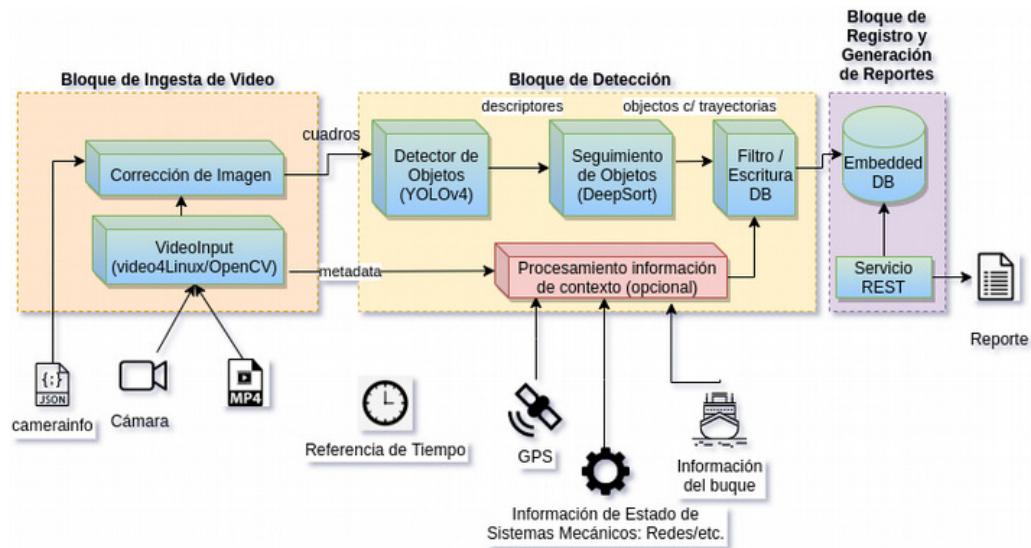


FIGURA 3.1. Descomposición funcional del sistema identificando tecnologías y flujo de datos.

De acuerdo a esta primera definición, se identificaron las tareas a realizar para llevar a cabo este sistema en dos grandes grupos (se omiten las relacionadas con documentación, capacitación, y otras que no sean estrictamente de diseño e implementación):

- Desarrollo de modelos de aprendizaje automático de los detectores y el extractor de características utilizado en el bloque de seguimiento, y eventualmente modelos adicionales para clasificación para el filtro final.
- Diseño e implementación de una arquitectura del sistema completo. Definición de los flujos de trabajo para el desarrollo y puesta en producción de prototipos.

El primer grupo contiene las tareas típicas del ciclo de vida de un problema de aprendizaje automático, mientras que el segundo grupo contempla aspectos de gestión de ambientes, diseño e integración de componentes con bases de datos y herramientas de monitoreo y consulta más propios de la ingeniería de software.

Si bien a nivel conceptual la frontera entre las competencias de ambos grupos está bien definida, en la implementación deben tenerse en cuenta aspectos como las restricciones operativas de los ambientes de desarrollo e inferencia de los modelos, que no necesariamente coinciden y pueden presentar incompatibilidades. Los flujos de trabajo, al involucrar múltiples etapas, también debían considerar la gestión de los productos intermedios. Por último, dado que se debía tener la capacidad de presentar prototipos para demostraciones (entrega continua), resultaba importante reducir el tiempo de puesta del sistema en producción mientras se continuaba su desarrollo.

La estrategia para despliegue de la solución consistió en utilizar una arquitectura de microservicios basada en contenedores Docker [53].

Los servicios incluyen:

- Aplicación que implementa la cadena de procesamiento desde la adquisición del video hasta su publicación en los servicios de bases de datos.
- Servicios de bases de datos: se utilizaron InfluxDB y ElasticSearch para series temporales y eventos respectivamente.
- Servicios de monitoreo, visualización y consulta: se utilizaron Grafana y Kibana.

Durante el desarrollo de aplicaciones debían ensayarse configuraciones de componentes e incluso podía ser requerido mantener en simultáneo distintos tipos de cadenas. Para lograr esto se implementó un *framework* de bloques reutilizables que permite definir cadenas con muy poco código y con un flujo de trabajo similar al que se tiene en Scikit-Learn.

Este *framework* o biblioteca de componentes, llamado “Videoanalytics”, es de código abierto y fue publicado en Github [54], Pypi [55], y Readthedocs [56].

Como la biblioteca debe tener la capacidad de realizar inferencia de los modelos de aprendizaje automático, procesamiento de video, tratamiento de datos y otras funciones complementarias como visualización y estar apoyada en bibliotecas existentes, encontrar una configuración con versiones compatibles y poder reproducir esa configuración es un aspecto que debe tenerse en cuenta. Se optó por utilizar Python 3 como lenguaje de desarrollo y puesta en producción, con ambientes del sistema de paquetes Conda [57] para facilitar la reproducibilidad del ambiente de ejecución.

La figura 3.2 presenta una vista simplificada de la organización de ambientes de ejecución y productos que se utilizó para poder cumplir con este objetivo.

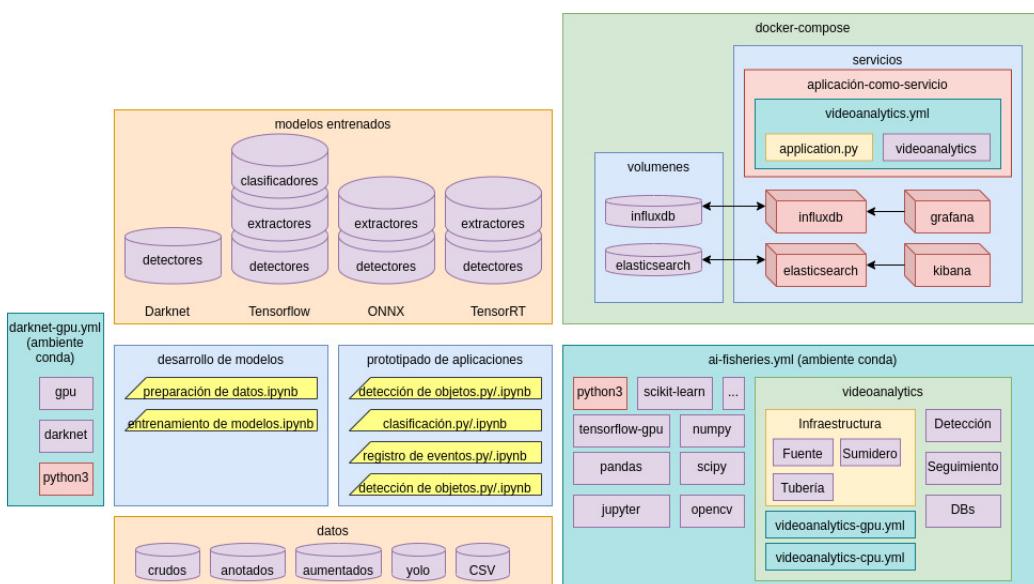


FIGURA 3.2. Diagrama simplificado de productos, ambientes de ejecución, servicios, scripts de procedimientos y aplicaciones y *framework* Videoanalytics.

Se definieron ambientes de Conda para distintos tipos de tareas, siendo algunos de ellos:

- **videoanalytics-gpu.yml**: ambiente mínimo para ejecución del framework videoanalytics en GPU.
- **videoanalytics-cpu.yml**: ambiente mínimo para ejecución del framework videoanalytics en CPU.
- **ai-fisheries.yml**: ambiente que agrega a uno de los anteriores dependencias que se utilizan únicamente durante el desarrollo: JupyterLab [58], librerías de preparación de datos, y otros.
- **darknet-gpu.yml**: ambiente mínimo para entrenamiento de modelos con el framework Darknet [59].

El criterio de separación de ambientes está asociado a los flujos de trabajo y la disponibilidad de recursos para cada uno de ellos, como CPUs y GPUs.

Cada ambiente tiene requerimientos de hardware con distintos tipos de restricciones y habilita la realización de un grupo de tareas. Dado que este proyecto se realizó en múltiples plataformas, era importante disponer del mismo ambiente con las mismas versiones de dependencias en cada computadora con el menor costo posible de migración.

Idealmente, se quisiera disponer de un único ambiente con GPU y compatibilidad entre todas las dependencias, pero dado que no resultó posible se intentó hallar un compromiso entre reducir la cantidad de configuraciones a gestionar y separar lo suficiente esas configuraciones para evitar conflictos de dependencias.

El entrenamiento de modelos también presentaría diferencias dependiendo del tipo de framework utilizado. En el momento en que se inició el proyecto, se consideró a Darknet como la opción más conveniente para entrenar los detectores YOLOv4. Implementar desde cero el lazo de entrenamiento en Tensorflow [60], dada su complejidad, parecía una opción arriesgada y bloquearía el avance de las otras tareas.

Para manipular los datos también se requiere la definición de convenciones y procedimientos si no se dispone de una herramienta que los defina o una práctica establecida de proyectos anteriores. Para este trabajo, los procedimientos incluyeron la obtención de datos crudos, a veces su etiquetado (manual), opcionalmente transformaciones como aumentado o preparación para un *framework* específico (por ejemplo YOLOv4 de Darknet) y otras operaciones de conversión y reorganización.

Hay datos que además son el resultado de alguna cadena de procesamiento, sobre la que se aplican procedimientos de ingeniería de características u otro tipo de preparación. Un ejemplo son las salidas del detector, utilizadas en el clasificador de actividades, en la búsqueda de parámetros óptimos de los algoritmos de seguimiento, o en la preparación de ejemplos positivos y negativos para el extractor de características. Para ordenar y facilitar su gestión, se establecieron convenciones de uso de directorios y nomenclatura, en muchos casos acompañadas de scripts que las automaticen cuando es posible.

Tanto los procedimientos como las aplicaciones se implementaron en Python. Se mantuvieron separados los procedimientos de preparación de datos y desarrollo

de modelos de prototipado de aplicaciones. El entorno utilizado fue JupyterLab para acompañar el código de indicaciones y conclusiones sobre los pasos realizados y tener agilidad en el edición en ambientes donde no se disponía de interfaz de escritorio nativa. Las aplicaciones luego podían exportarse a programas en Python para ejecutarse como un servicio.

La gestión de los modelos requería algunas consideraciones. Se optó por utilizar Tensorflow/Keras como ambiente de inferencia durante el desarrollo porque fue el más utilizado durante el transcurso de la especialización y sobre el que se disponía de mayor experiencia.

Los modelos de detectores entrenados con Darknet no son inmediatamente compatibles con Tensorflow, por lo que también se requirió un procedimiento para convertirlos.

Por último, Tensorflow no es un ambiente óptimo para inferencia, especialmente si se dispone de modelos con redes profundas y recursos restringidos [61]. Por este motivo, para cumplir los requerimientos de reducir tiempos de inferencia los modelos de detección debían exportarse al formato intermedio *Open Neural Network eXchange* (ONNX) [62] para generar un modelo optimizado para puesta en producción.

## 3.2. Desarrollo de modelos de aprendizaje automático

Existe un flujo de trabajo típico para un problema de aprendizaje automático que puede generalizarse en las siguientes etapas (figura 3.3) [63]:

- Identificación de un problema que puede resolverse con técnicas de aprendizaje automático.
- Construcción de un conjunto de datos. Si se utilizan para el desarrollo de modelos de aprendizaje supervisado, las tareas comprendidas en esta etapa también pueden incluir el etiquetado de los datos.
- Preparación de los datos: actividades que pueden incluir su exploración preliminar, manipulación, reorganización, preprocessamiento e ingeniería de características, dependiendo de su tipo. A modo de ejemplo, si los datos están constituidos por variables numéricas o categóricas pueden aplicarse métodos de selección de características o transformaciones de distribución. En cambio, si se trata de datos que no tienen una estructura definida, como imágenes o video, las técnicas a aplicar pueden ser distintas. También se deben separar los datos en particiones destinadas a entrenamiento y evaluación de modelos.
- Desarrollo y entrenamiento de modelos: definición de arquitecturas, hiperparámetros, implementación y ejecución de lazos de entrenamiento si no alcanza con los que se dispone por defecto.
- Evaluación y selección de modelos: obtención de métricas de desempeño del modelo y otras como tiempo de inferencia, consumo de recursos que impactan en decisiones de su puesta en producción.

- Puesta en producción: exportación y optimización para una arquitectura de hardware específica, encapsulamiento en componentes de software para manejo de errores, integración de servicios, y otros.

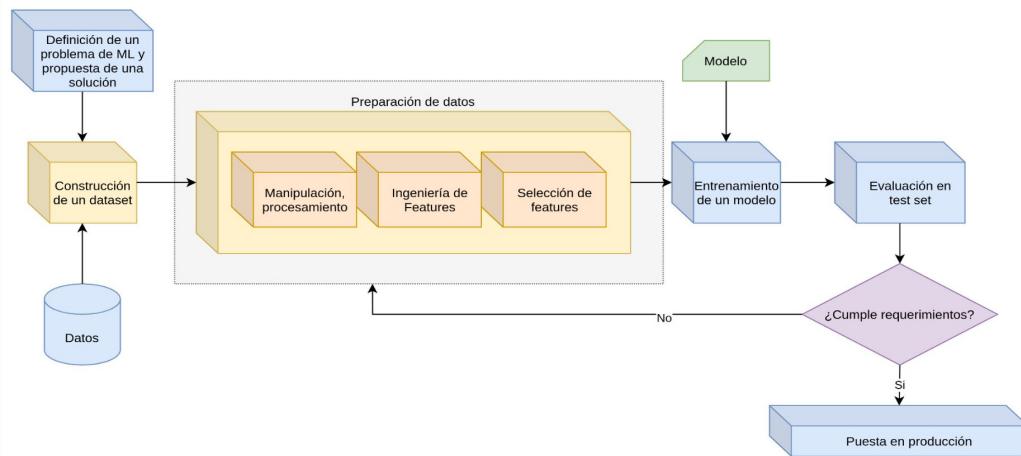


FIGURA 3.3. Flujo de trabajo para un problema de aprendizaje automático.

Para este trabajo se han utilizado modelos que tienen como entrada dos tipos de datos: imágenes de cuadros de video y posiciones de objetos generadas por el detector. En ambos casos se ha omitido la dimensión temporal.

Para la aplicación de detección de operadores, redes y capturas en la pesca de langostinos fue necesario etiquetar datos manualmente, anotando las regiones rectangulares que contenían objetos de interés. Para la aplicación de detección de pescados se dispuso de datos ya etiquetados.

La preparación de datos de imágenes consistió principalmente en realizar procedimientos de reorganización de directorios y particionamiento en entrenamiento y evaluación. En el caso de los datos de pesca de langostinos, por disponer de pocas muestras, además se realizó un aumentado con transformaciones geométricas afines y del espacio cromático.

Para entrenar el modelo de clasificación de actividades, los datos de las posiciones de objetos resultantes de la ejecución del detector sobre un caso de prueba fueron particionados en entrenamiento y evaluación. Su preparación incluyó algunas transformaciones previas de acuerdo a los rangos y distribuciones de las variables de entrada. También se generaron nuevas variables estadísticas, cuya importancia para el modelo fue estudiada posteriormente con métodos de filtro.

Tanto para los modelos de detectores de objetos como para los modelos de clasificadores se utilizaron arquitecturas y lazos de control listos para usar de los frameworks Darknet y Scikit-learn respectivamente. Unicamente se editó la configuración de algunos hiperparámetros.

Para los modelos de extracción de características, en cambio, sí fue necesario implementar un lazo de control utilizando funciones más avanzadas de TensorFlow/Keras.

Todos los modelos obtenidos fueron finalmente evaluados sobre las particiones de evaluación.

### 3.3. Desarrollo de modelos de detección

Este trabajo se apoya en la detección de objetos como una de las etapas necesarias para implementar todas las cadenas de procesamiento propuestas.

Existen múltiples arquitecturas de detectores de objetos, e inclusive existen modelos preentrenados para detectar personas, vehículos, animales y objetos varios. No obstante, para algunas aplicaciones muy específicas, como en este caso, es necesario reconocer objetos que no son habituales o realizar un ajuste fino de una arquitectura existente para lograr mejores resultados en determinadas condiciones de operación.

Puede ocurrir que aún contando con un modelo entrenado para reconocer los objetos de interés, en la escena se presenten en tamaño reducido, apilados o con alta superposición. También puede ocurrir que objetos de distinta clase se confundan por tener una apariencia muy similar y requieran que el modelo sea sensible a algún detalle particular para poder distinguirlos. A modo de ejemplo, los pescados pueden estar apilados entre sí y ser difíciles de diferenciar de algunas zonas de la imagen por el tipo de iluminación de la escena, o los operadores de ambientes industriales pueden estar vestidos con el mismo uniforme y los detalles que los diferencien ser poco visibles (el color de un casco o una banda de color). En el caso de los pescados, algunos de los trabajos consultados utilizan las aletas, la textura de las escamas y otros detalles conocidos por los expertos, si la resolución y condiciones de iluminación con que se tomaron las imágenes permite captarlos.

Además de aplicar conocimiento del dominio del problema, existen técnicas que pueden aplicarse durante el entrenamiento de un detector para mejorar su desempeño, que incluyen el ajuste de parámetros tanto de entrenamiento, como de la arquitectura de las capas internas y la preparación de los ejemplos de entrenamiento para los casos difíciles.

El alcance de este trabajo es la sistematización de un procedimiento de entrenamiento de modelos de la familia YOLOv4 para que sea fácilmente reproducible al disponer de nuevos datos y que gradualmente incluya conocimiento específico del dominio a medida que se adquiere mayor experiencia en la preparación y evaluación de modelos.

La entrada del procedimiento consiste en videos, imágenes o una combinación de ambos. Deben contener los objetos que interesa detectar, preferentemente con distintas disposiciones y afectados por fuentes de iluminación variable.

La salida del procedimiento es un directorio con los siguientes archivos:

- **classes.txt**: nombres de las clases.
- **model.cfg**: arquitectura de la red.
- **model.weights**: pesos de las celdas o neuronas del modelo.
- **./model-tf**: directorio respetando la estructura de modelos exportados para inferencia de Tensorflow (*frozen graph*).
- **model-perf-report.txt**: métricas de desempeño del modelo en la partición de evaluación.

El procedimiento se escribió en forma de cuaderno de Jupyter y fue ejercitado con dos conjuntos de datos:

- Reconocimiento de operadores, redes y pescados en un video de pesca de langostinos.
- Datos de la competencia de kaggle para reconocimiento de pescados. En este caso los ejemplos no son característicos de la zona, pero sí visualmente similares. Por lo tanto, se considera que son representativos para este problema.

El entrenamiento comprende los siguientes pasos siendo los últimos dos de carácter opcional, necesarios sólo para despliegue *edge*:

1. Obtención de los datos.
2. Exploración.
3. Etiquetado (en caso de no haber sido previamente anotados).
4. Preparación para framework Darknet.
5. Aumentado (opcional, dependiendo de la cantidad de muestras disponibles).
6. Ajuste de hiperparámetros de configuración de Darknet.
7. Entrenamiento en ambiente Darknet.
8. Evaluación.
9. Exportación de modelos a Tensorflow.
10. Exportación de modelos a ONNX (opcional).
11. Optimización de modelos para TensorRT (opcional).

Dependiendo del caso, algunos de estos pasos pueden requerir mayor o menor intervención.

### 3.3.1. Entrenamiento para detección de presas con dataset de Kaggle

En el año 2017, el sitio de competencias Kaggle presentó un desafío que consistía en utilizar aprendizaje e inteligencia artificial para fiscalizar la actividad a bordo de buques pesqueros a partir de imágenes de los sistemas de CCTV [13].

Existen algunas diferencias en los tipos de escenas y de las capturas entre los datos de Kaggle y los de Río Negro y Chubut, propias de los sistemas de pesca y especies que se capturan en cada zona, pero los principios para detectarlos son válidos en ambos casos cuando se trata de capturas grandes.

Por no disponer datos regionales en cantidad suficiente, para poder avanzar con este proyecto se entrenó un modelo de detector utilizando los datos de la competencia de Kaggle. Se tiene como objetivo repetir el proceso con datos de la región cuando estén disponibles.

La figura 3.4 muestra las especies de la competencia de Kaggle titulada *The Nature Conservancy Fisheries Monitoring*.



Fish images are not to scale with one another

FIGURA 3.4. Especies para el desafío de aprendizaje automático de Kaggle <sup>1</sup>.

### Obtención de los datos

En este caso los datos consisten en imágenes agrupadas por clase que se descargan con la aplicación de Kaggle. Se menciona este aspecto porque si bien el sistema que se desarrolló está orientado a video, en algunos casos en que no se dispone de videos resulta de interés ensayar cadenas de procesamiento que tengan como entrada lotes de imágenes. Este es uno de los motivos por los que se implementó un componente específico para leer lotes de imágenes.

### Exploración de los datos

La figura 3.5 muestra ejemplos de las imágenes del desafío de Kaggle. En algunos casos como el atún, las especies son muy similares a las que se capturan en Argentina y en principio parece razonable asumir que si el algoritmo funciona para los atunes nórdicos, también lo hará para el pez bonito que se capture en el mar argentino. En los datos de Kaggle se dispone de tres tipos de atún y una especie de tiburón, que puede considerarse, siempre desde el punto de la vista de la detección, similar al cazón que se capture en Argentina. En los datos disponibles hay imágenes diurnas y nocturnas. Un aspecto a considerar, que no fue estudiado, es explotar la correlación entre el horario y el tipo de captura y separar ejemplos diurnos y nocturnos.

<sup>1</sup>Sitio oficial: <https://www.kaggle.com/c/the-nature-conservancy-fisheries-monitoring>. Accedido 09/08/2021.

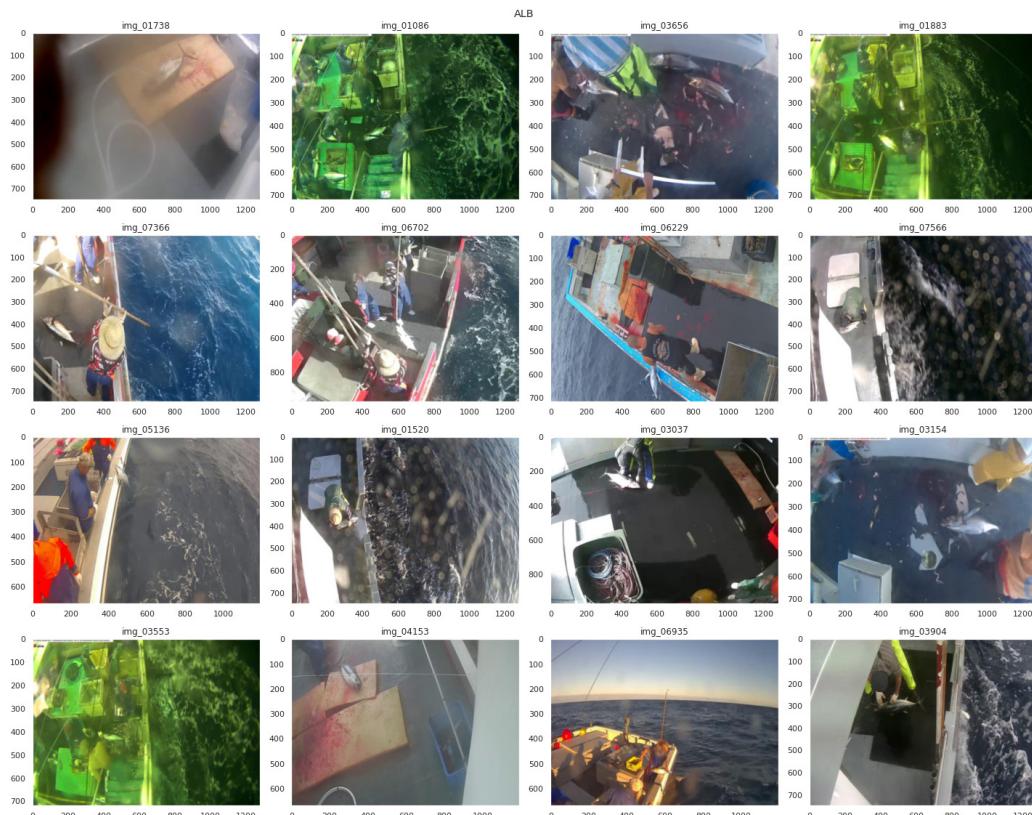


FIGURA 3.5. Selección de muestras elegidas al azar del desafío de aprendizaje automático de Kaggle, en este caso de atún, especie similar al pez bonito que se captura en el mar argentino.

En la figura 3.6 se muestra la distribución de clases, donde se puede apreciar que los datos están desbalanceados y, con excepción del primer caso, muy por debajo de la recomendación de dos mil ejemplos por clase que se indica en las instrucciones de entrenamiento del algoritmo YOLOv4 [50]. No obstante, para los fines de ensayar este prototipo, resultan suficientes.

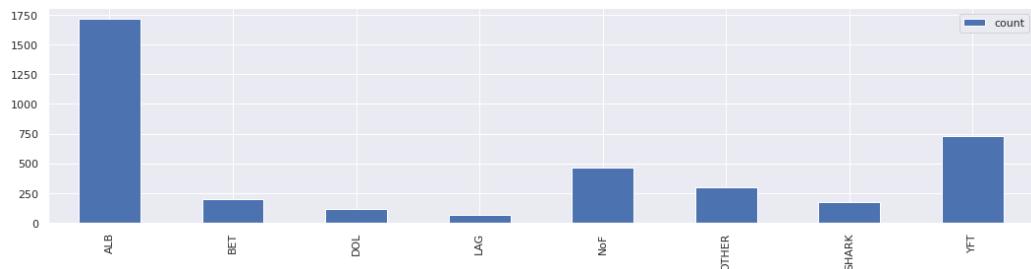


FIGURA 3.6. Distribución de clases en los datos de la competencia de Kaggle. Afortunadamente una de las clases más representada es la especie de atún *Thunnus alalunga* (ALB) y se aproxima a las 2000 muestras, que es la cantidad recomendada para el modelo YOLOv4.

### Etiquetado y preparación de datos para Darknet

Los datos originales de Kaggle no estaban etiquetados para detección de objetos (únicamente estaban organizados en directorios separados por clases), pero existían contribuciones de los usuarios en los foros que agregaban las anotaciones con las regiones rectangulares. El framework Darknet tiene una convención de anotación que utiliza coordenadas normalizadas y el centro de las regiones rectangulares en lugar de los extremos de las regiones, por lo que se escribieron funciones de conversión.

Además, conforme se explica en el instructivo de Darknet, los datos deben disponerse respetando la siguiente estructura:

```
\label{verb:yolo-struct}
./kaggle-fisheries-yolo/
|- data/
|   |- 00000.jpg
|   |- 00000.txt
|   |- ...
|- classes.names
|- train.txt
|- test.txt
|- yolo-fisheries.cfg
|- fisheries.data
```

Donde:

- Un directorio debe contener las imágenes de entrenamiento y validación acompañadas de un archivo de texto del mismo nombre donde se listan los objetos presentes y sus localizaciones (“data”).

- Un archivo de texto debe contener una lista con los nombres de cada clase (“classes.names”).
- Deben indicarse en archivos de texto los nombres de las imágenes que corresponden a entrenamiento y validación (“train.txt” y “text.txt” respectivamente).
- Un archivo de texto debe contener la configuración del modelo, indicando los parámetros de cada una de las capas de la red como dimensiones y cantidad de neuronas, así como otros hiperparámetros de entrenamiento como opciones de aumentado, o propios del algoritmo, como definición de *anchor boxes* (“.cfg”).
- Un archivo de texto debe listar las ruta de los archivos mencionados, necesarios para un proceso de entrenamiento o evaluación (“.data”).

Dentro de las tareas de preparación de datos se implementaron funciones para simplificar este tipo de conversiones y de reorganización de archivos.

### Ajuste de hiperparámetros

Existen algunas recomendaciones para mejorar el desempeño de un detector, algunas de ellas a partir de los tipos de errores que se observen en la evaluación. Una característica que mejora la inferencia de objetos pequeños, por ejemplo, es el tamaño de los *anchor boxes*.

En este caso sólo se modificó el tamaño de celda de entrada a 416x416 píxeles de los 608x608 por defecto y se ajustó la cantidad de filtros de las capas convolucionales para la cantidad de clases.

Se eliminaron además algunas opciones de aumentado que no estaban disponibles en el ambiente, por depender de una versión de OpenCV [64] que no pudo ser instalada.

De las ocho categorías disponibles en los datos originales, se decidió utilizar únicamente las siete clases identificadas, descartando el grupo “otros”.

### Entrenamiento en ambiente Darknet

El entrenamiento se realizó con la aplicación de Darknet en un servidor dedicado y el seguimiento de la evolución del algoritmo se hizo filtrando la salida de los archivos de log. La integración con aplicaciones de monitoreo como Tensorboard no fue posible y no se utilizó una herramienta para generar las curvas a partir de los logs.

### Evaluación

La evaluación del algoritmo fue realizada con Darknet, utilizando la configuración de *mAP* con umbral de IoU de 50 %. Además se obtuvo para cada clase la cantidad de verdaderos positivos y falsos positivos. Los resultados se detallan en el capítulo 4.

### Exportación de modelos a Tensorflow

Para la exportación de modelos a Tensorflow, se utilizó una herramienta externa [65]. La salida es un modelo que puede ser importado para inferencia con Tensorflow.

La figura 3.7 muestra un ejemplo de utilización del detector.



FIGURA 3.7. Ejemplos como el de la figura muestran que la detección es una estrategia válida para piezas grandes que guardan alguna separación entre sí en la medida en que tengan algún rasgo exterior significativo que permita distinguirlas.

#### 3.3.2. Entrenamiento para detección de redes, operadores y descarte en pesca de langostinos

El langostino es un tipo de captura de la zona de Río Negro y Chubut, para la cuál la detección de piezas individuales resulta inviable por el tamaño y volumen de las capturas, como se puede ver en la figura 3.8.

Entrenar un detector de objetos, no obstante, sí puede ser útil para registrar movimientos de operadores, redes y *bycatch*, es decir, pescados u otros animales que quedaron atrapados en las redes y no son el objetivo de la captura, pudiendo incluso tratarse de especies protegidas.

El video de entrada utilizado en este caso tiene diez minutos y siete segundos de duración. En él se exhibe una maniobra realizada para la pesca de langostino en el buque tangonero “Mirta R” en la zona de Playa Unión, en la provincia de Chubut [66]. Se consideró que era un caso representativo de los videos que se utilizarían para la pesca de langostinos en la provincia de Río Negro.

<sup>2</sup>Foto obtenida de: <https://pescachubut.ar/bridi-sobre-el-adelanto-de-la-pesca-del-langostino-en-rio-negro-hay-muy-buen-resultado/>. Accedido 09/09/2021.

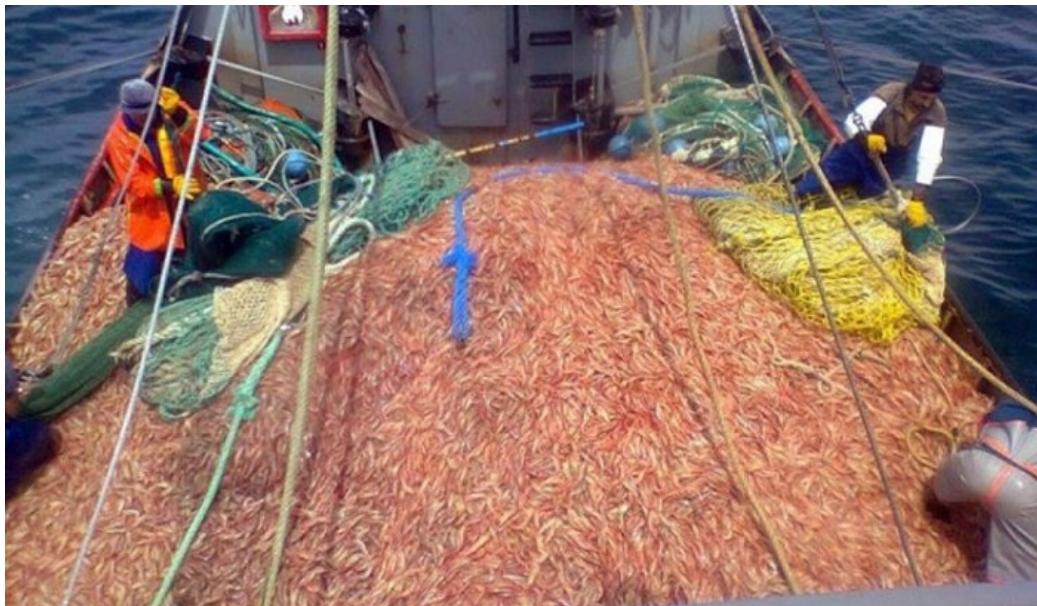


FIGURA 3.8. Puede concluirse de la imagen que realizar un conteo individual de cada pieza es computacionalmente costoso, además de poco preciso. Para este tipo de escenas resulta conveniente recurrir a otros métodos de estimación de volumen que utilicen cámaras estéreo o segmenten la imagen y estimen el volumen a partir del área. Foto de [www.pescachubut.ar](http://www.pescachubut.ar)<sup>2</sup>.

### 3.3.3. Preparación de datos y aumentado

Por tratarse de un video, se realizó un script para extraer cuadros elegidos de manera aleatoria y posteriormente se particionaron en un conjunto de entrenamiento y evaluación. Finalmente se organizaron para Darknet.

A diferencia de lo que ocurrió con los datos de Kaggle, esta vez sí fue necesario etiquetar los datos, para lo que se utilizó la herramienta LabelImg [67]. Dado que este proceso demanda un esfuerzo importante, se etiquetaron del orden de cincuenta ejemplos y se utilizó la técnica de aumentado [68], que consistió en aplicar transformaciones afines, distorsiones de color, y agregado de ruido gaussiano utilizando la librería imgaug [69].

La figura 3.9 muestra algunos ejemplos luego del aumentado. Es importante mencionar que YOLOv4 ya realiza operaciones de aumentado, pero existieron dos razones por las cuales se optó por realizarlo aparte:

- Hacer independiente el proceso de preparación de datos de Darknet en caso de que se quieran utilizar otros ambientes de entrenamiento.
- El aumentado de YOLOv4 requería dependencias que no estaban disponibles en la distribución de Darknet del sistema de paquetes conda en el momento de realizar este trabajo.

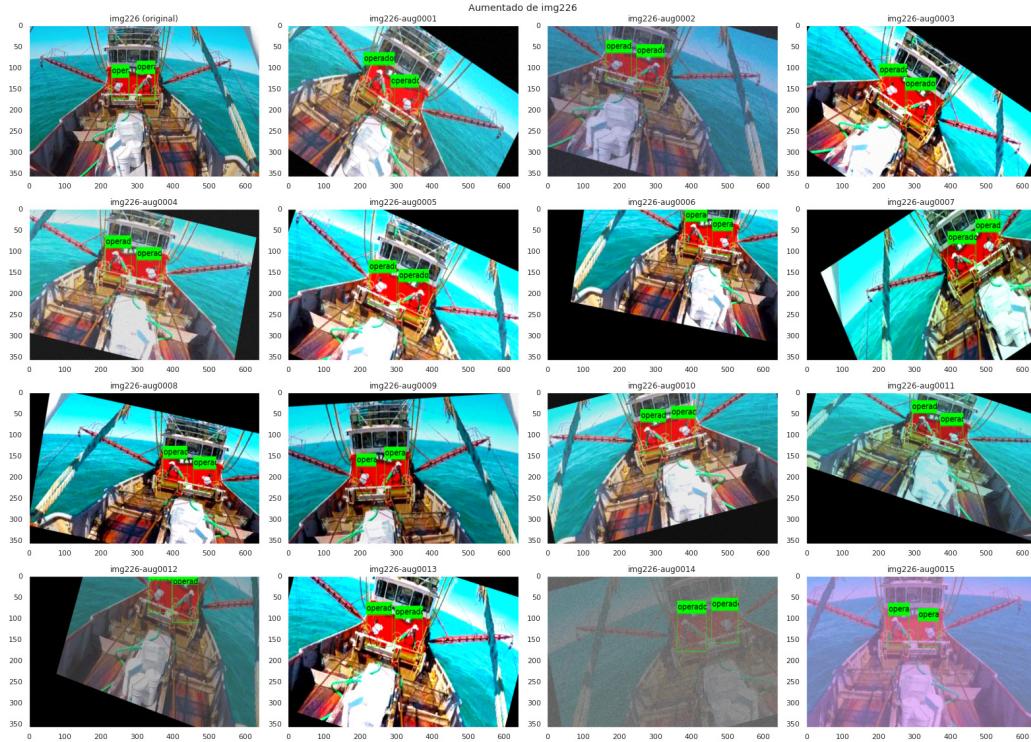


FIGURA 3.9. Resultado de aplicar transformaciones afines, ruido gausiano, multiplicación de color, contraste, espejados y recortes para aumentar la varianza de los datos de entrada.

### 3.4. Desarrollo de un modelo de clasificación de actividades

En el capítulo 1 se expuso que uno de los requerimientos de este trabajo es poder identificar de manera automática las actividades que ocurren en un video a bordo de un buque.

Para ello se desarrolló un procedimiento que fue ensayado con un video de pesca de langostinos asumiendo que es extensible a otros casos. Se utilizó un video de diez minutos de duración. El procedimiento consistió en preparar los datos y entrenar distintos modelos de aprendizaje supervisado. Puede dividirse en las siguientes etapas:

1. Preparación de los datos:
  - a) Etiquetado manual: se definió un conjunto de actividades que interesa identificar y se indicaron los intervalos en que ocurren.
  - b) Particionado: se agruparon los datos por cuadros y se filtraron únicamente las detecciones de operadores (cada cuadro puede tener múltiples detecciones incluyendo operadores, redes y capturas). Luego se dividieron los datos en entrenamiento y evaluación. Esta última partición se reservó para utilizar en la selección de modelos.
  - c) Ingeniería de características: se generaron nuevas variables estadísticas de las posiciones de los operadores.

- d) Análisis exploratorio inicial: se estudiaron las distribuciones, valores extremos, y características de las variables de entrada para obtener una idea preliminar de qué modelos ensayar y qué transformaciones considerar para las variables numéricas y tratamiento de valores extremos.
- 2. Desarrollo de modelos: se definió un procedimiento para entrenar y evaluar distintos modelos de clasificación. Cada modelo incluye una forma de preprocesamiento.
- 3. Evaluación de resultados y conclusiones: se estableció una métrica del desempeño de los modelos y fueron comparados. Se tuvo en cuenta también el tiempo de inferencia.

Si bien estos pasos fueron descriptos como una secuencia, este proceso es iterativo.

### 3.4.1. Etiquetado manual y preparación de datos

Se utilizó el video referido en el apartado 3.3.2, pero esta vez identificando actividades en lugar de entidades para un detector. Se espera repetir este mismo procedimiento contando con más videos, y sometiendo la asignación de momentos del video a revisión de expertos. Excepto por alguna optimización o mejora de los hiperparámetros de los modelos o el preprocesamiento previo, se asume que los pasos a realizar deberían ser independientes de los videos de entrada.

El video fue procesado con el modelo YOLOv4 entrenado para detectar operadores que se describe en el apartado 3.3.2. Sólo se registraron las detecciones de personas (en este caso operadores en cubierta, filtrando las de las clases restantes). La salida de este paso es un archivo CSV donde cada fila contiene una detección encontrada en cada cuadro (por lo tanto casi siempre hay más de una fila por cuadro).

#### Identificación de actividades

Sin contar aún con conocimientos de la técnica de pesca de langostino en barcos tangoneros y la terminología utilizada para describir los pasos involucrados, se consultó bibliografía sobre el tema [70, 71, 72] y se realizó una definición preliminar de los momentos de interés:

1. Los operadores hacen correcciones en la orientación y ubicación de las redes. Se anuncia el levado de redes con una campana y se acercan más operadores a cubierta.
2. Algunos de los operadores levan las redes, mientras que el resto espera.
3. Se levan completamente las redes y se descargan en la cubierta, con asistencia de algunos operadores.
4. El barco aumenta la marcha, y quedan algunos operadores en cubierta separando la captura en recipientes.

La tabla 3.1 muestra los momentos identificados en el video del caso de prueba. A todos los cuadros restantes se los agrupó en la categoría “otros”.

TABLA 3.1. Momentos identificados en un video que muestra una maniobra realizada para la pesca de langostino en el buque tangonero “Mirta R” en la zona de Playa Unión, en la provincia de Chubut.

Evento	Descripción	Inicio	Fin
Levado de redes 1	De uno a tres operadores realizan maniobras de recogimiento de redes utilizando el malacate ubicado en el centro de la imagen (figura 3.10).	00:00	00:48
Levado de redes 2	Dos operadores tiran de los cabos para recoger las redes.	01:05	01:40
Levado de redes 3	Se recogen las redes y los operadores se reubican en cubierta.	02:45	05:30
Redes recogidas	Las redes están visibles en el video. Puede ser de interés para estimar el volumen de captura (figura 3.11).	05:33	05:47
Descarga de redes.	Se vacía el contenido de las redes en la cubierta (figura 3.12).	05:47	07:20
Redes descargadas / clasificación	El barco se desplaza a mayor velocidad y algunos operadores clasifican la captura, arrojando algunos especímenes y preparando recipientes para otros (figura 3.13).	07:20	10:03

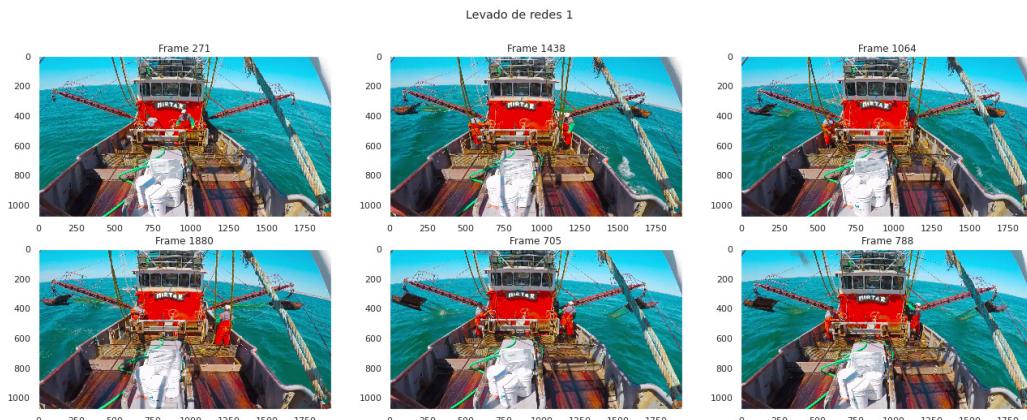


FIGURA 3.10. Un grupo reducido de operadores coordinan maniobras de recogimiento de redes utilizando el malacate.

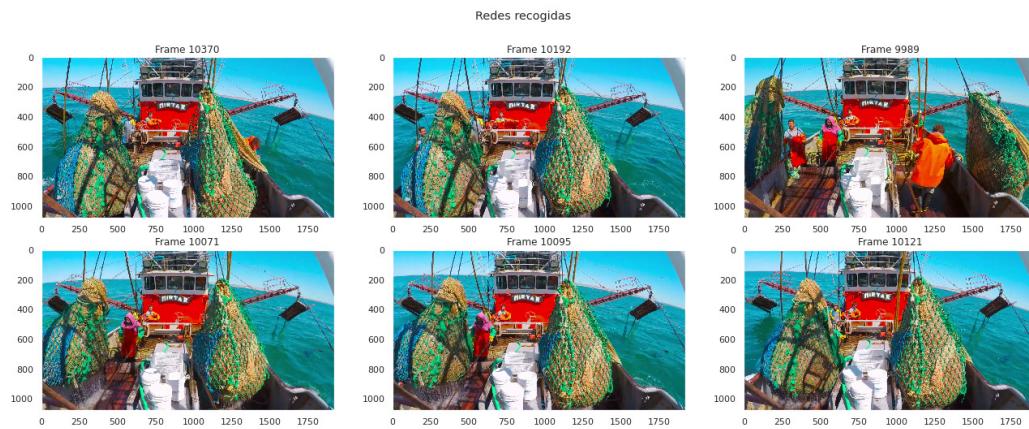


FIGURA 3.11. Las redes están visibles en el video. Esta sección del video es de interés para estimar el volumen de captura.

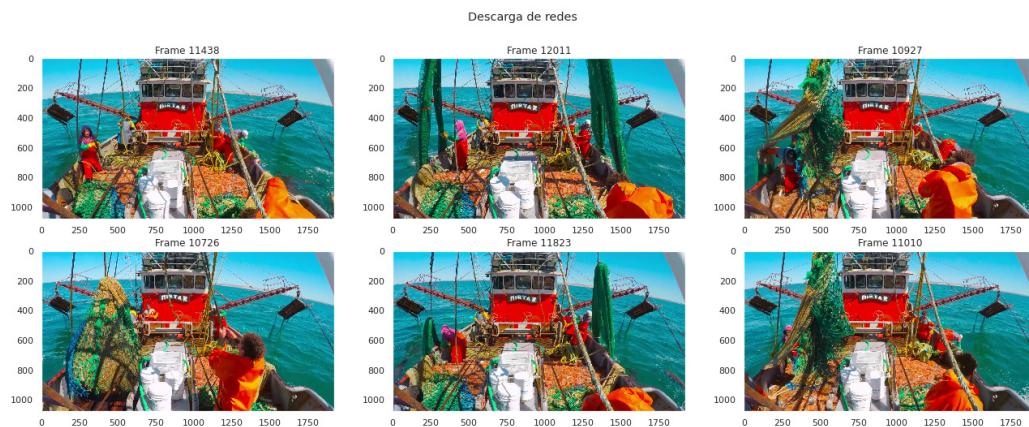


FIGURA 3.12. Se vacía el contenido de las redes en la cubierta.

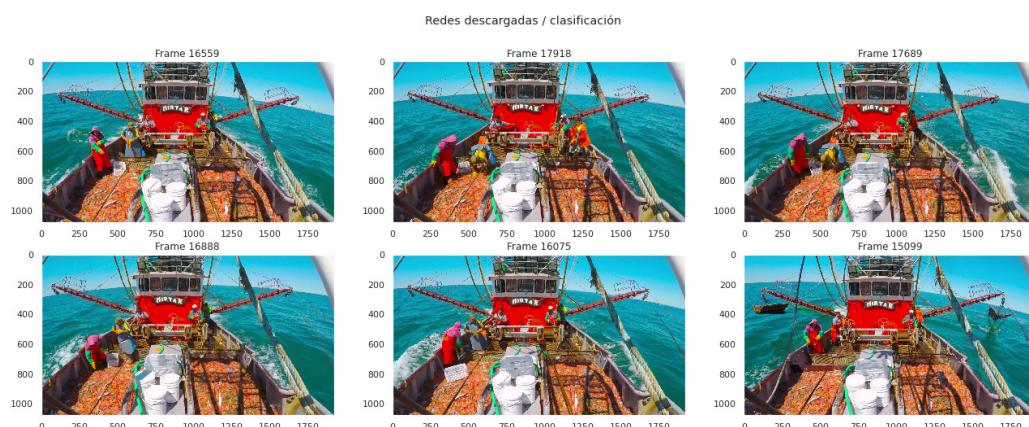


FIGURA 3.13. Algunos operadores clasifican la captura, arrojando algunos especímenes y preparando recipientes para otros.

### Análisis exploratorio e ingeniería de características

Luego de observar el video y de intuir que existía una relación entre la forma en que se desplazaban los operadores en la cubierta del barco para realizar distintas actividades, se consideró conveniente reemplazar las detecciones individuales por variables estadísticas, de modo que para cada cuadro del video exista una única observación que contenga momentos estadísticos de interés, quedando los datos de entrada formados por las siguientes columnas:

- $n$ : cantidad de detecciones.
- $mean_x, mean_y, mean_{area}$ : media muestral de las posiciones y área de todas las detecciones.
- $s_x, s_y, s_{area}$ : desvío estándar muestral de las posiciones y área de todas las detecciones.
- $var_x, var_y, var_{area}$ : varianza muestral de las posiciones y área de todas las detecciones.
- $skew_x, skew_y, skew_{area}$ : oblicuidad de las posiciones y área de todas las detecciones.
- $kurt_x, kurt_y, kurt_{area}$ : kurtosis de las posiciones y área de todas las detecciones.

Las figuras 3.14 y 3.15 muestran la evolución temporal y distribución de tres de las quince variables agregadas. A nivel intuitivo parece existir una correlación entre los valores que adquieren una determinada combinación de estas variables y el tipo de actividad que se está realizando.

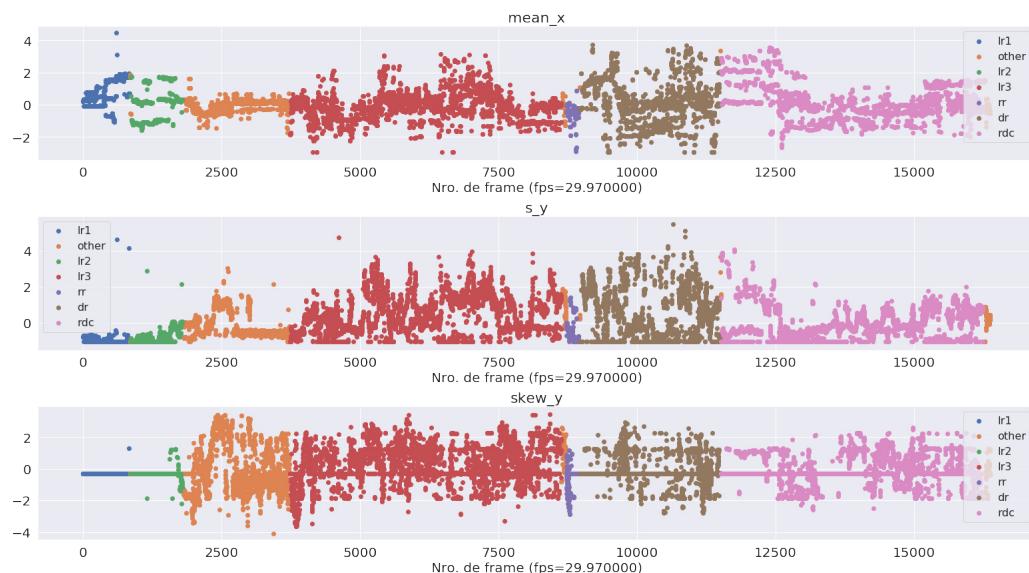


FIGURA 3.14. Evolución en el tiempo de media de posición horizontal y desvío estándar y oblicuidad de posición vertical de operadores. Se indican los tres momentos de levado de redes (lr), recogimiento de redes (rr), descarga de redes (dr) y recolección y clasificación (rdc).

Ejemplos de estas conclusiones preliminares son:

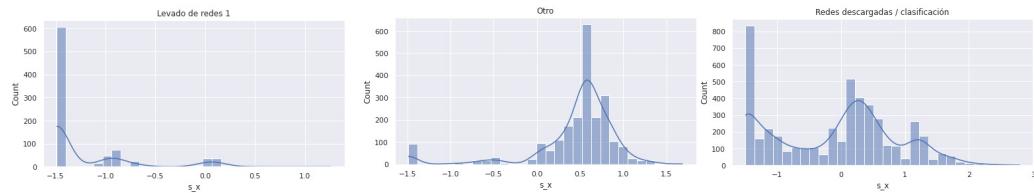


FIGURA 3.15. Histogramas para desvío estándar de posición horizontal de operadores en distintos momentos del video.

- La media de la posición vertical de los operadores parece ser diferente para cada actividad.
- La oblicuidad negativa caracteriza casi completamente al levado de redes 1.
- La kurtosis aumenta significativamente para los intervalos no etiquetados (otros).

Es importante tener en cuenta tanto cuando se hace este tipo de análisis como para el desarrollo de modelos la representatividad que tiene cada clase en los ejemplos disponibles. La figura 3.16 muestra la representación que tiene cada una de las clases identificadas en el conjunto de datos de entrada.

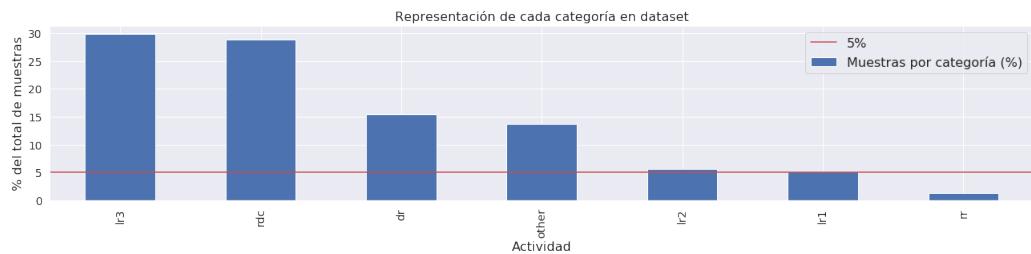


FIGURA 3.16. La línea roja representa el 5 % del dataset.

Para este caso no se realizaron acciones para obtener datos de entrada mejor balanceados como obtener muestras adicionales de las clases faltantes, eliminar de las sobrantes o aplicar métodos de síntesis como SMOTE [73] o ADASyn [74]. Se optó por utilizar la métrica *AUC*, de menor sensibilidad al desbalance de los datos.

### Selección de características

Si bien un método de selección de variables manual puede ser aplicado exitosamente para un número reducido de variables, como ocurre en este problema, uno de los objetivos del trabajo era sistematizar la solución, y esto incluye la selección de variables de entrada.

Una forma de seleccionar características es mediante tests de correlación entre la variable de entrada y de salida. El objetivo de estos tests es permitir identificar las variables que mayor impacto tienen sobre la salida y eliminar las restantes.

En este caso, el problema contiene todas las variables de entrada de tipo numérico y la variable objetivo es categórica, por lo tanto son aplicables los métodos de ANOVA, correlación de Kendall e información mutua [63].

### 3.4.2. Sistematización del método

Un objetivo importante de este trabajo es desarrollar la capacidad de sistematizar los métodos para que sean reproducibles con nuevos datos con el mayor grado de automatización posible.

La figura 3.17 muestra el proceso resultante de las tareas que se describieron desde que se seleccionan videos representativos de problemas de clasificación hasta que se obtiene un clasificador que puede utilizarse en producción encapsulado como un servicio.

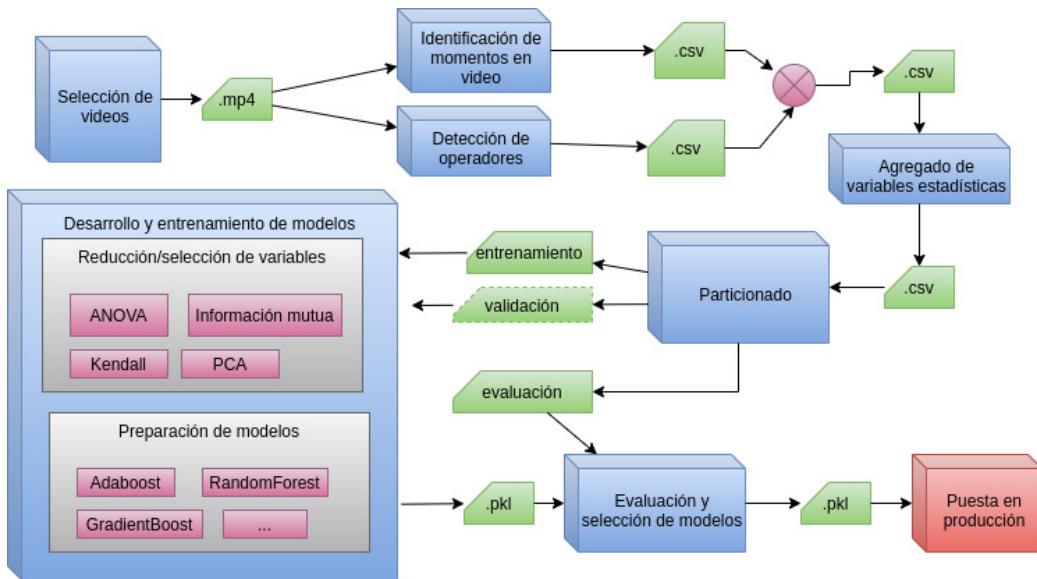


FIGURA 3.17. Proceso completo para generar un clasificador de actividades.

Estas tareas fueron implementadas en su mayoría como código en Python en cuadernos interactivos (Jupyter) que toman como parámetro un archivo de entrada e indican como asociar las actividades identificadas a los intervalos en los que ocurren. Este es el único paso que requiere la participación de un experto con conocimiento de las actividades que ocurren en el buque.

## 3.5. Desarrollo de modelos de extracción de características

Un extractor de características es un modelo que tiene como objetivo codificar una entrada multidimensional, como por ejemplo una imagen de dimensiones  $(W, H, 3)$  en un vector de características con una cantidad de elementos reducida (valores típicos suelen rondar entre los 64 y 256 elementos). Se suele utilizar el término *embedding* para hacer referencia a este vector.

Existen distintos motivos por los que se quisiera hacer esto, por ejemplo:

- Capturar las características más significativas para implementar un clasificador (esto es lo que realizan las capas más próximas a la función sigmoide o softmax de salida de un clasificador que utiliza redes convolucionales).

- Capturar las características más significativas que permiten agrupar los ejemplos similares y separar los que difieren. Esto permite identificar duplicados o realizar identificación de rostros, entre otras posibles aplicaciones.
- Obtener una representación de una señal que permita regenerarla, por ejemplo en un *autoencoder*. Una aplicación típica es la detección de anomalías, en la que un sistema “aprende” una representación de los datos que le permite regenerarlos con un error mínimo. Al recibir datos nuevos o “anómalos”, la diferencia es mayor, y estableciendo un umbral de error esto permite reconocerlos cuando una señal reconstruida supera ese umbral.

La misma arquitectura de un extractor de características puede servir para cualquiera de estos tres propósitos, sin embargo, existen técnicas de entrenamiento que resultan más apropiadas para cada tipo de tarea.

En este trabajo el extractor de características tiene dos aplicaciones:

- Algoritmo DeepSORT: para seguimiento de piezas, es un complemento al modelo de estimación cinemática en el problema de asignación. Es de interés contar con un extractor especialmente entrenado para las salidas del detector (en este caso pescados y operadores).
- Búsqueda en video: dada una imagen con una superficie o recipiente en distintos estados (conteniendo pescados y vacía) o una imagen de una red, identificar todos los segmentos en el video en que se presentan escenas conteniendo regiones similares.

### 3.5.1. Redes siamesas

Una red siamesa es una arquitectura de red neuronal que contiene dos o más subredes idénticas utilizadas para generar un vector de características para cada entrada y comparar los mismos.

El objetivo de entrenamiento es computar la distancia entre las dos entradas, minimizándola cuando se trata de imágenes similares y maximizándola cuando no.

Algunas aplicaciones típicas de este tipo de redes son:

- Identificación de duplicados.
- Identificación de rostros (que no debe confundirse con detección de rostro).

#### Entrenamiento con *triplet loss*

Existen distintas formas de entrenar redes siamesas. En este trabajo se utilizó un modelo con tres subredes idénticas. La entrada del modelo son tres imágenes, dos de ellas similares llamadas ancla (*anchor*) y ejemplo positivo y una tercera que no guarda relación, llamada ejemplo negativo.

El objetivo del modelo es aprender a estimar la similitud entre imágenes. La función utilizada en este caso es triplet loss, definida como [32]:

$$L(A, P, N) = \max(|f(A) - f(P)|^2 - |f(A) - f(N)|^2 + margin, 0) \quad (3.1)$$

El margen es un valor arbitrario que debe superarse para que dos ejemplos se consideren suficientemente distintos.

La figura 3.18 presenta un diagrama de la arquitectura de red siamesa que se ha utilizado, en el que se han omitido los extractores.

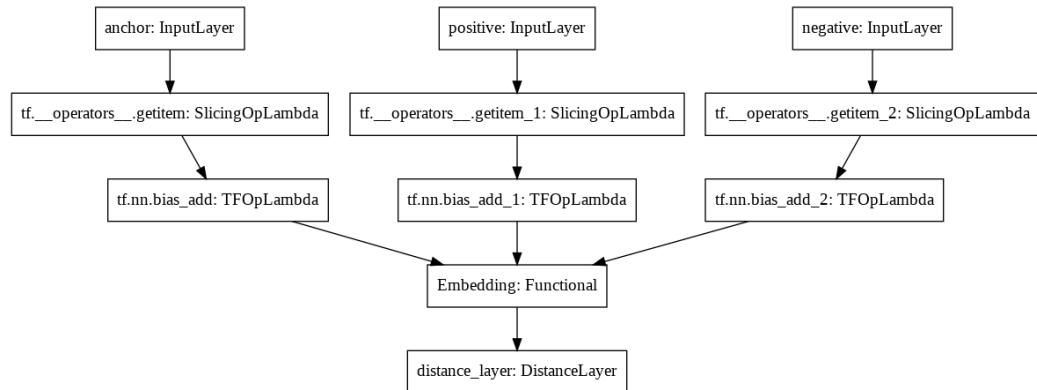


FIGURA 3.18. En el diagrama se muestran las salidas de las tres cabezas de los extractores de características que convergen en el vector de características (*embedding*)

En Tensorflow/Keras no se dispone de una capa que permita computar  $|f(A) - f(P)|^2$  y  $|f(A) - f(N)|^2$ , por lo que debe implementarse con una capa personalizada, como se muestra en el fragmento de código 3.1.

```

1 class DistanceLayer(layers.Layer):
2     """ Computar distancia entre anchor y ejemplo positivo y entre
3         anchor y
4         ejemplo negativo.
5     """
6     def __init__(self, **kwargs):
7         super().__init__(**kwargs)
8
9     def call(self, anchor, positive, negative):
10        ap_distance = tf.reduce_sum(tf.square(anchor - positive), -1)
11        an_distance = tf.reduce_sum(tf.square(anchor - negative), -1)
12        return (ap_distance, an_distance)
  
```

CÓDIGO 3.1. Capa que implementa la función de distancia en Keras.

Además, a diferencia de los detectores de objetos de Darknet o los modelos de Scikit-Learn desarrollados para el clasificador de actividades, tampoco existe en Keras/Tensorflow un modelo listo para usar que permita entrenar una red siamesa.

El código 3.2 muestra como se implementa un modelo personalizado de red siamesa que define su propio lazo de entrenamiento y evaluación con la función de costo de la ecuación 3.1.

Para cada iteración (*step*), el método de entrenamiento (*train\_step*) realiza las siguientes acciones, tomando como entrada un fragmento de los datos de entrenamiento (el tamaño de este fragmento se define en el momento de ejecutar el entrenamiento):

1. Calcular el error (*loss*) y registrarlo en un contexto GradientTape, que es un mecanismo de Tensorflow que permite almacenar los gradientes para que luego puedan ser aplicados por el paso de optimización, además de servir para estudiar el aprendizaje de los modelos.
2. Obtener los gradientes con respecto a los coeficientes o pesos de la red que están habilitados para entrenamiento. En otro tipo de red podría asumirse que son todos, pero en una red siamesa, los extractores generalmente ya fueron previamente entrenados, y por lo tanto están etiquetados como fijos (*frozen* o *non trainable*).
3. Invocar al paso de optimización con los gradientes recién obtenidos.
4. Actualizar el último valor de error.

El método de evaluación (*test\_step*) únicamente realiza una inferencia sobre los datos de evaluación y calcula el error.

Otras funciones, como *metrics*, no son obligatorias pero permiten cumplir con la interfaz común a todos los modelos, que tiene entre sus métodos expuestos el de obtener referencias a sus componentes internos, como por ejemplo sus métricas.

```

1 class SiameseModel(Model):
2     def __init__(self, siamese_network, margin=0.5):
3         super(SiameseModel, self).__init__()
4         self.siamese_network = siamese_network
5         self.margin = margin
6         self.loss_tracker = metrics.Mean(name="loss")
7
8     def call(self, inputs):
9         return self.siamese_network(inputs)
10
11    def train_step(self, data):
12        with tf.GradientTape() as tape:
13            loss = self._compute_loss(data)
14            gradients = tape.gradient(loss, self.siamese_network.
15            trainable_weights)
16            self.optimizer.apply_gradients(
17                zip(gradients, self.siamese_network.trainable_weights))
18            self.loss_tracker.update_state(loss)
19            return {"loss": self.loss_tracker.result()}
20
21    def test_step(self, data):
22        loss = self._compute_loss(data)
23        self.loss_tracker.update_state(loss)
24        return {"loss": self.loss_tracker.result()}
25
26    def _compute_loss(self, data):

```

```

27     ap_distance , an_distance = self.siamese_network(data)
28     loss = ap_distance - an_distance
29     loss = tf.maximum(loss + self.margin, 0.0)
30     return loss
31
32 @property
33 def metrics(self):
34     return [self.loss_tracker]

```

CÓDIGO 3.2. Definición de modelo personalizado de red siamesa en Keras con lazo de entrenamiento y evaluación.

### 3.5.2. Arquitecturas de extractores

El objetivo del extractor de características es generar un *embedding* para cada una de las imágenes de la tupla de entrada (*anchor*, *positive*, *negative*). El objetivo de la red siamesa es hallar los parámetros óptimos para minimizar la distancia entre *anchor* y el ejemplo positivo, y maximizarla entre *anchor* y el ejemplo negativo.

Cualquier red convolucional de clasificación puede utilizarse como un extractor de características si se le remueven algunas de las capas finales, que son aquellas que se ocupan de realizar la clasificación.

Para este trabajo, además de utilizar el modelo preentrenado que se distribuye con DeepSORT, se entrenaron dos extractores:

- Extractor basado en una red Resnet50 [75] previamente entrenada, para realizar aprendizaje de transferencia [76].
- Extractor sencillo, de arquitectura similar a la del extractor presentado en DeepSORT, pero entrenado desde cero.

#### Extractor basado en Resnet50

En este caso se utilizó una red Resnet50 [75], incluida en la colección de modelos de Keras preentrenada. El fragmento de código 3.3 construye un modelo de extracción a partir de los siguientes pasos:

1. Instanciar un modelo preentrenado de Resnet50 con pesos para clasificar imágenes generales (*Imagenet*) [77]. Para este modelo se define un tamaño de entrada y se excluyen las capas próximas a la salida de la red (por convención se llaman *bottom* a las cercanas a la entrada y *top* a las de la salida).
2. Crear la salida de la red con capas adicionales, que serán entrenadas. Para ello es necesario reducir a una única dimensión la última capa del modelo Resnet50 (esto se hace con la capa *Flatten*). Las capas que se agregan contienen 512, 256 y 256 neuronas respectivamente, separadas por una operación de *Batch Normalization*, que mejora el aprendizaje de redes profundas anulando aleatoriamente algunas neuronas durante el entrenamiento [78].
3. Combinar Resnet con las capas nuevas, definiendo Resnet como entrada y las recién creadas como salida, por lo que la salida será un vector de *embedding* de 256 elementos.

4. Establecer como entrenables únicamente los pesos de la red posteriores a la capa “conv5\_block1\_out”.

```
1 def create_feature_extractor_model():
2     base_cnn = resnet.ResNet50(
3         weights="imagenet", input_shape=INPUT_IMG_SHAPE + (3,),
4         include_top=False
5     )
6
7     flatten = layers.Flatten()(base_cnn.output)
8     dense1 = layers.Dense(512, activation="relu")(flatten)
9     dense1 = layers.BatchNormalization()(dense1)
10    dense2 = layers.Dense(256, activation="relu")(dense1)
11    dense2 = layers.BatchNormalization()(dense2)
12    output = layers.Dense(256)(dense2)
13
14    feature_extractor_model = Model(base_cnn.input, output, name="Embedding")
15
16    trainable = False
17    for layer in base_cnn.layers:
18        if layer.name == "conv5_block1_out":
19            trainable = True
20        layer.trainable = trainable
21
22    return feature_extractor_model
```

CÓDIGO 3.3. Extractor basado en Resnet50 con agregado de capas densas.

## 3.6. Desarrollo de un framework modular

Experiencias anteriores han demostrado la conveniencia de contar con una arquitectura modular que se adecúe a cambios de alcance y de tecnologías, habituales en proyectos de carácter experimental.

También es importante que la obtención de esta flexibilidad no implique un costo adicional excesivo redundando en interfaces y código auxiliar para permitirla, como sucede a veces con proyectos de software que definen una infraestructura compleja para otorgar control completo sobre la mayoría de sus aspectos.

En el espíritu de lograr la máxima flexibilidad posible con el mínimo código de infraestructura, para este proyecto se decidió utilizar el lenguaje Python y su ecosistema de bibliotecas para ciencias de datos: OpenCV, Numpy, Pandas, Scipy, ScikitLearn, entre otras. También se optó por un patrón de diseño de software que combina algunos aspectos del patrón de diseño *pipeline* o “tubería” y el patrón de diseño *blackboard* [79], ambos orientados a lograr la interoperatividad de componentes que realizan tareas específicas para la resolución de una tarea compleja.

### 3.6.1. El paradigma de tuberías

Se implementó un módulo que permita combinar bloques implementando el paradigma de tuberías (en inglés *pipeline*).

Esta implementación define dos conceptos:

- Fuentes: componentes que adquieren datos de un dispositivo como una cámara de video o un archivo.
- Sumideros: componentes que procesan datos generados previamente. Su salida puede agregar nueva información para que sea consumida por otro componente de la cadena de procesamiento o una acción, como publicar datos en una base de datos o a través de un sistema de mensajería como ZeroMQ.

Las tuberías son componentes encadenados que realizan pasos sucesivos para obtener un producto final. Cada tubería se conforma a partir de una única fuente de información y uno o más sumideros que realizan operaciones de procesamiento.

Las tuberías no están restringidas a cuadros de video pero, por simplicidad, la implementación realizada se acotó a video RGB, dejando la posibilidad para más adelante de extenderse a cámaras estéreo o con más bandas, o incluir señales de otro audio u otro tipo.

La figura 3.19 muestra una configuración de tubería que realiza seguimiento de objetos y clasificación de actividades a partir de sus trayectorias.

Se puede observar que la tubería se define como un grafo acíclico. Si bien podría paralelizarse, el modelo de ejecución elegido, por simplicidad, es ejecutar por cada iteración cada componente en orden secuencial. La secuencia para garantizar que se cumplen las dependencias se obtiene utilizando el algoritmo de ordenamiento topológico.

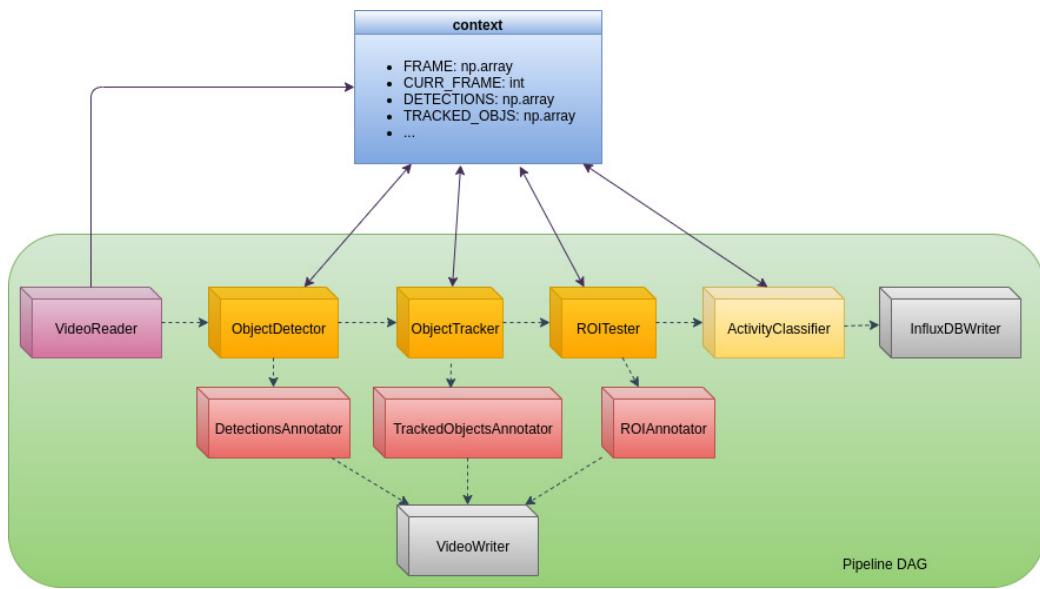


FIGURA 3.19. Se muestra la composición de una cadena para clasificación basada en trayectorias utilizando los componentes que se describen en las siguientes secciones.

Otro aspecto a considerar es que no existe comunicación directa entre los componentes. En cambio, los datos son compartidos en un contexto global que se pasa por referencia a cada componente en el momento de inicializarlo.

Esta decisión está inspirada en el patrón de diseño *blackboard*, principalmente porque reduce la complejidad de definir interfaces específicas entre componentes y facilita el prototipado de estrategias conjuntas. Los datos se intercambian en estructuras de datos estándar de Python y Scipy del tipo de diccionarios y arreglos de Numpy. Cada componente declara en su interfaz de programación de aplicaciones (API, por sus siglas en inglés *Application Programming Interface*) qué variables obtiene del contexto y cuáles crea o actualiza. Es importante tener presente que esta aproximación, mientras es útil para prototipado, puede no ser la solución óptima para puesta en producción. Afortunadamente, las ediciones que deben realizarse en el código para sustituir el uso de diccionarios y otras estructuras computacionalmente costosas son acotadas a las interfaces de los componentes seleccionados para la cadena final.

Uno de los aspectos importantes del prototipado de sistemas de múltiples componentes es poder ensayar configuraciones para cada problema particular. Esto puede incluir medir el desempeño de cada componente y diagnosticar fallas en el algoritmo o detectar cuellos de botella.

Junto con el desempeño de los modelos (en el caso de los componentes de inferencia), el tiempo de procesamiento y consumo de recursos son aspectos que deben contemplarse para seleccionar el algoritmo óptimo.

La infraestructura implementada contempla este aspecto y colecta métricas en tiempo de ejecución, que incluyen el tiempo inicial y final de procesamiento global y el promedio de cada componente. Estas métricas se han utilizado en los ensayos que se describen en el capítulo 4.

La biblioteca está organizada en una primera división de fuentes y sumideros. Para cada uno de estos grupos luego se hacen sucesivas subdivisiones para tareas o dominios específicos. Para este trabajo se implementaron componentes para las siguientes funciones:

- Substracción de fondo y estimación de movimiento.
- Detección de objetos.
- Seguimiento de objetos.
- Visualización y anotación.
- Definición de regiones de interés.
- Registro en archivos y publicación en bases de datos.

Como se ilustra en la figura 3.20, cada componente es una especialización de una clase abstracta de tipo Fuente o Sumidero que forma parte de la infraestructura de bloques para composición de cadenas.

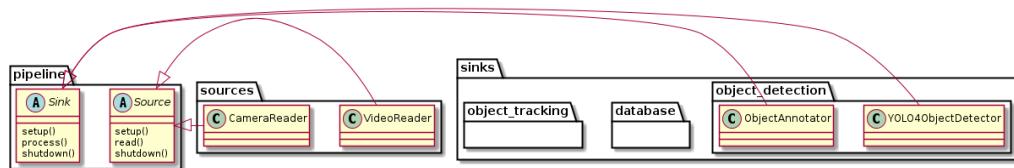


FIGURA 3.20. Cada nuevo componente especializa las clases de tipo Fuente o Sumidero, permitiendo un flujo de trabajo similar al que se dispone en frameworks de aprendizaje como Scikit-Learn.

### 3.6.2. Fuentes

Las fuentes abstraen los detalles de implementación de distintas formas de adquirir una señal. La señal obtenida es particionada en una secuencia de unidades de información que tengan sentido para la aplicación, como por ejemplo, cuadros de un video.

Las fuentes deben proporcionar un mecanismo para garantizar que la tasa de salida no ocasiona un cuello de botella en el resto de los componentes de la cadena de procesamiento. La forma más sencilla de implementación de este mecanismo, que es la adoptada actualmente, es mediante un parámetro de configuración que omite algunos cuadros (la cantidad es configurable).

En la versión que se implementó existen tres fuentes:

- Lector de archivo MP4.
- Lector de cámara.
- Lector de lote de archivos de imágenes.

Las dos primeras están destinadas al uso nominal con video, mientras que la segunda se implementó para los ensayos en los que no se disponía de videos y debían probarse componentes con imágenes, como ocurre con algunas aplicaciones de los detectores.

### 3.6.3. Sumideros

Los sumideros abstraen el concepto de recibir datos para producir nuevos datos o realizar una acción específica, como la publicación en bases de datos.

El lineamiento que se estableció para el diseño de los componentes de este tipo, tanto actuales como futuros es que realicen una tarea específica y las acciones complejas se puedan lograr a partir de la interacción entre componentes en lugar de un enfoque monolítico.

#### Detección de objetos

En la sección 3.3 se describió el proceso de generación de modelos de detección. Para utilizar estos modelos se requiere de un motor de inferencia, que puede ser el mismo que se usó para generar el modelo u otro que presente alguna ventaja en algún aspecto, como escalabilidad, compatibilidad con otros componentes, o estar optimizado para una plataforma específica.

Se consideraron para este trabajo distintos motores de inferencia, que se resumen en la tabla 3.2. No obstante, sólo se implementó en la versión actual de Video-analytics la detección de objetos en una clase que carga un modelo de Tensorflow y lee resultados precalculados de un archivo CSV.

Cada motor de inferencia tiene características propias que incluyen ventajas y restricciones. Es una funcionalidad que puede abstraerse con facilidad y permitir, en tiempo de ejecución, elegir tanto el modelo de detección como el ambiente de inferencia utilizado dependiendo de los recursos de cómputo e infraestructura a disposición.

La principal ventaja de realizar la inferencia con Tensorflow es que se utiliza el mismo ambiente de desarrollo una vez que se realizó la exportación de un modelo de Darknet. OpenCV soporta el formato de Darknet de forma nativa, por lo tanto ninguno de los dos requiere la conversión de formato y ambos proveen ambientes de inferencia tanto para GPU como para CPU. La decisión de cuál es más conveniente depende del desempeño en cada plataforma, que debe medirse y puede variar dependiendo de las versiones de *drivers*, bibliotecas y *hardware* utilizado. Para un despliegue *edge*, la opción más recomendable es TensorRT, pero requiere un proceso de preparación de modelos sujeto a la plataforma en que se utilice. Para un despliegue en servidores, Tensorflow Serving permite implementar la inferencia como un servicio con los protocolos gRPC o REST, resolviendo problemas como el balance de carga de manera transparente para el usuario. Esto es de particular interés si desea escalar la solución y utilizar múltiples CPUs y GPUs. Para este tipo de problema la interfaz gRPC parece la más adecuada.

También resulta de interés contar con funciones complementarias a la detección, como la anotación de los resultados en video, la capacidad de registrarlos en un archivo CSV y poder utilizar detecciones precalculadas. Un ejemplo de la utilidad de esto último es permitir evaluar componentes de una cadena de procesamiento que dependan de la salida del detector, como por ejemplo algoritmos de seguimiento. Si no se dispone de suficientes recursos de cómputo para realizar la inferencia, pueden calcularse las detecciones una única vez y luego utilizar los valores precalculados.

TABLA 3.2. Entornos de inferencia (sólo se consideran para modelos de detección de objetos).

Entorno	Descripción
Tensorflow/Keras	Inferencia utilizando el mismo ambiente de desarrollo de modelos. Los modelos de Darknet deben ser convertidos a Tensorflow.
OpenCV DNN	Motor de inferencia de OpenCV. Se pueden utilizar directamente los modelos de Darknet sin necesidad de conversión.
Tensorflow Serving	Inferencia utilizando un servicio separado que se consume con gRPC. Escalable, separa el despliegue de los servicios de inferencia del resto de la aplicación. Los modelos de Darknet deben ser convertidos a Tensorflow.
TensorRT	Óptimo para despliegues <i>edge</i> . Cada modelo debe ser convertido y optimizado para la plataforma de destino.
Darknet	No requiere pasos adicionales para los modelos de Darknet.

La figura 3.21 muestra la definición de una interface para un detector de objetos e implementaciones utilizando distintos motores de inferencia, y las clases complementarias para carga desde CSV y anotación.

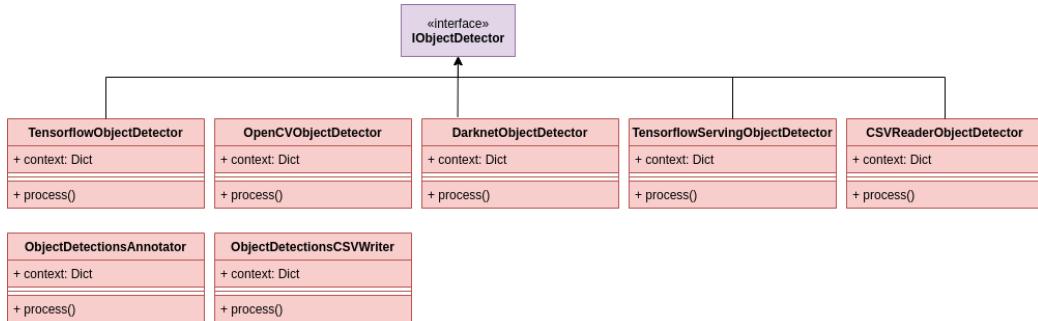


FIGURA 3.21. Diagrama de clases de componentes de detección en Videoanalytics.

## Seguimiento de objetos

Para el seguimiento de objetos se utilizó el código original provisto por los autores de cada trabajo [33, 30]. Se encapsuló cada implementación en una especialización de la clase Sumidero y se unificó el formato de los datos, además de agregar clases para registro de CSV y anotación, al igual que con los detectores de objetos.

### Definición de regiones de interés

Las regiones de interés (abreviadas ROIs por *Regions of Interest*) son regiones poligonales que pueden ser sometidas evaluaciones geométricas de contención e intersección, como por ejemplo si incluyen un punto o se intersectan con otra forma geométrica. En video analítico se utilizan para evaluar la presencia de objetos, actividad en un área u otros eventos de interés, como por ejemplo:

- Si un objeto ingresa o egresa de una ROI.
- El tiempo que un objeto permanece en una ROI.
- La cantidad de objetos de un determinado tipo en una ROI.

Combinadas con la salida de un detector de objetos, un algoritmo de seguimiento o un estimador de movimiento pueden proporcionar información sobre lo que está ocurriendo en una escena.

La figura 3.22 muestra un ejemplo de ROIs para contar presencia de objetos en zonas de la cubierta de un buque, mientras que la figura 3.23 muestra una aproximación distinta en la que se estima el movimiento en cada zona mediante un substractor de movimiento.

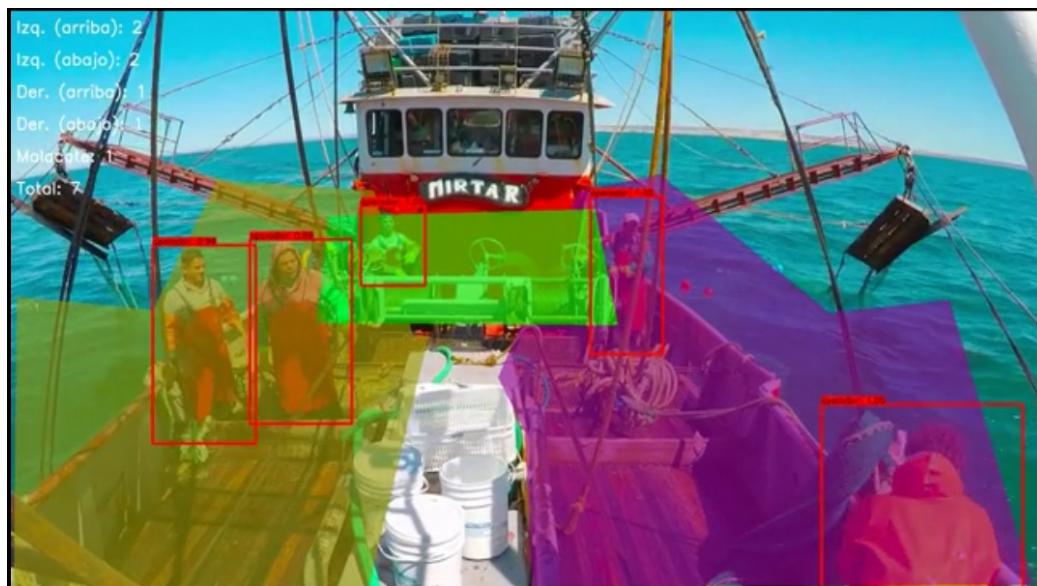


FIGURA 3.22. Conteo de objetos por ROI utilizando detector de objetos.

Para este trabajo no se implementó una interfaz gráfica para ingresar las ROIs, por lo que se deben indicar mediante un archivo JSON como el que se describe en el ejemplo 3.4.

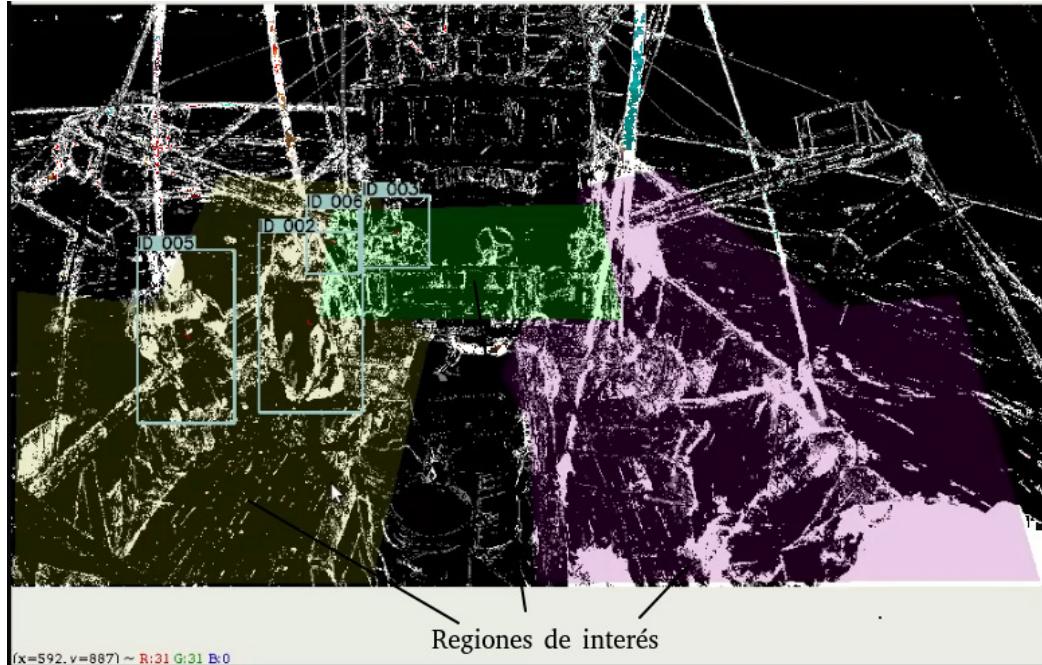


FIGURA 3.23. Estimación de movimiento por ROI utilizando sustractor de fondo. Los píxeles que quedan luego de la sustracción son los que tienen algún movimiento, que puede ser también por el movimiento natural del barco, cámara, y cambios de iluminación. Estableciendo un umbral, puede utilizarse como un indicador de movimiento o actividad en un área determinada.

```

1 {
2   "regions": [
3     {
4       "name": "tangon_babor_sup",
5       "polygon": [[385, 315], [676, 334], [754, 625], [668, 810], [165,
757]],
6       "color": [155,155,0]
7     },
8     {
9       "name": "tangon_babor_inf",
10      "polygon": [[11, 532], [777, 606], [666, 1073], [5, 1074]],
11      "color": [155,155,0]
12    },
13    {
14      "name": "tangon_estribor_sup",
15      "polygon": [[1057, 336], [908, 682], [954, 773], [1522, 754],
16      [1565, 595], [1210, 305]],
17      "color": [155,0,155]
18    },
19    {
20      "name": "tangon_estribor_inf",
21      "polygon": [[973, 1072], [949, 625], [1749, 538], [1904, 1068]],
22      "color": [155,0,155]
23    },
24    {
25      "name": "puente",
26      "polygon": [[1088, 369], [592, 382], [570, 580], [1131, 584]],
27    }
28  ]
29}
```

```

30     "color": [0,255,0]
31   }
32 ]
33 }
```

CÓDIGO 3.4. Definición de regiones de interés en archivo JSON.

Este archivo contiene una lista de regiones, donde cada región queda definida a partir de:

- Un nombre, que luego puede ser utilizado para generar variables asociadas a esa región, como por ejemplo la cantidad de instancias de un objeto contenido en ella. Si bien no es una restricción, es conveniente que el nombre no tenga espacios.
- Un polígono, que se conforma por una lista de vértices, cada uno de ellos indicado por la posición expresada en píxeles  $(x, y)$ , tomando como origen la esquina superior izquierda de la imagen.
- Un color especificado por la intensidad de sus componentes rojo, azul y verde. Este color es utilizado en la anotación para colorear el interior de la región.

### 3.7. Despliegue con microservicios e integración con bases de datos

Los microservicios son una forma de construir software en la que los bloques funcionales de una aplicación son partes independientes que se comunican por medio de un sistema de mensajería o protocolo como, por ejemplo, REST.

En una arquitectura de microservicios cada aplicación queda aislada y define la información que requiere y provee al resto de los componentes. Una de las principales ventajas de esta arquitectura es que facilita la integración de diferentes tecnologías y lenguajes, permitiendo además reemplazar o sumar componentes a un sistema en producción sin necesidad de modificar los que están operativos [80].

Una forma de implementar microservicios es con contenedores Docker y las herramientas asociadas para orquestarlos, como Docker Compose [81], Docker Swarm [82] o Kubernetes [83], siendo los últimos dos aptos para puesta en producción. En este trabajo se utilizó Docker Compose, orientado principalmente a desarrollo y prototipado, pero que utiliza un formato muy similar a Docker Swarm.

Hay una amplia gama de servicios que se distribuyen como contenedores Docker, entre los que se incluyen bases de datos, gestión de alertas y herramientas de monitoreo y consulta. Para este trabajo se eligieron los servicios para bases de datos InfluxDB y ElasticSearch para series temporales y eventos respectivamente, y Grafana y Kibana como interfaces de monitoreo y consulta. Se adoptó este enfoque para cubrir dos requerimientos:

- Permitir a usuarios con perfil de analista de datos consultar todos los eventos registrados durante un intervalo de tiempo para que puedan ser tratados con alguna herramienta específica, como por ejemplo Python, R o Matlab. Estos resultados deben discriminar, como mínimo, fecha y hora y

tipo de evento, para luego poder confeccionar estadísticas de actividad por momentos del día, total de capturas por tipo de pieza, total de descartes, cumplimiento de normas (permanencia máxima de una pieza en una cinta transportadora o recipiente), y otros.

- Presentar los resultados del procesamiento en tiempo real o de un video a un usuario o cliente de manera visual, manteniendo la posibilidad de ampliar la solución para otros usos. A modo de ejemplo, se deben poder mostrar contadores de piezas capturadas o descartadas e indicar la presencia de operadores en un sector, entre otros.

La escritura en las bases de datos se hace a través de componentes específicos (Sumideros) para escritura en InfluxDB y ElasticSearch. Cada implementación utiliza los clientes en lenguaje Python de los sitios oficiales de cada base de datos. Las variables y documentos que se publican se toman del contexto compartido.

### 3.7.1. InfluxDB y TICK

InfluxDB es una base de datos de series temporales y, por lo tanto, está orientada a datos donde la estampa de tiempo es importante. Permite la inserción y consulta de datos rápida, manteniendo en memoria la información más reciente. Al incluir como mínimo la estampa de tiempo como índice, todas las consultas que requieran la búsqueda de información u operaciones de agregación en intervalos de tiempo son rápidas en comparación con motores de bases de datos diseñados para otros fines, como bases de datos relacionales, por ejemplo [84].

Además, InfluxDB forma parte del *stack TICK*, que se muestra en la figura 3.24 llamado así por sus componentes Telegraph, InfluxDB, Cronograph y Kapacitor. Estos componentes cumplen las siguientes funciones:

- Telegraph: colecta de datos.
- InfluxDB: almacenamiento de datos.
- Cronograph: panel de administración y monitoreo (web).
- Kapacitor: motor de reglas y generación de alertas.

Para este trabajo se ha utilizado únicamente InfluxDB, y se ha sustituido Cronograph por Grafana, que es una aplicación muy similar para la que existen *plugins* que permiten su integración con las principales bases de datos. También hay *plugins* para generar paneles, gráficos, visualización de video, y otras funciones.

El alcance propuesto para este trabajo cubre hasta el registro de eventos en una base de datos, pero en caso de incorporar alarmas y otros comportamientos, Kapacitor sería una alternativa de implementación que no requeriría modificar ninguno de los componentes existentes.

### 3.7.2. ElasticSearch y ELK

Si bien podrían representarse la mayoría de los datos generados por las aplicaciones como una serie temporal, no es la forma más conveniente en todos los casos.

---

<sup>3</sup>Imagen de <https://www.influxdata.com/>. Accedido 09/08/2021.

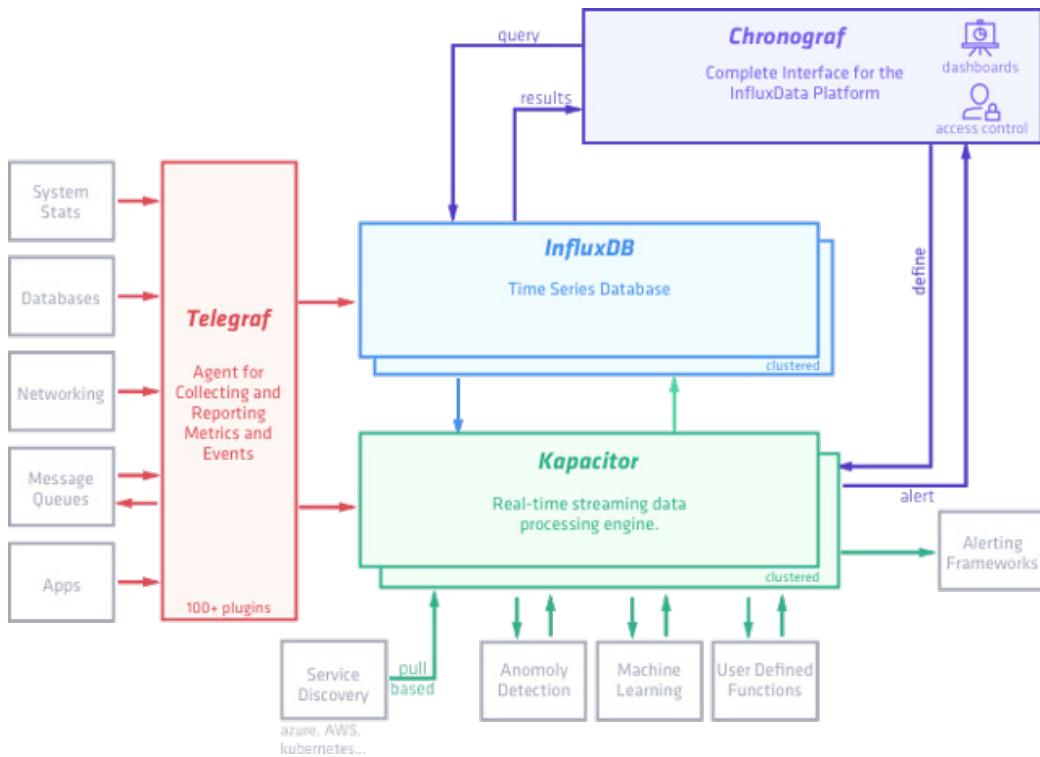


FIGURA 3.24. Componentes de TICK<sup>3</sup>.

Para los datos que tienen estructuras más complejas, que incluyen listas, identificadores, y otra información adicional resulta más apropiado tratarlos como eventos o documentos, pero teniendo también en cuenta el aspecto de la inserción y consulta rápida (no todos los motores de bases de datos disponen la información más reciente en memoria).

Una de las decisiones de diseño de Videoanalytics fue utilizar diccionarios de Python y arreglos de Numpy para representar los datos de entrada y salida de los componentes. Esto tiene la ventaja de que con esta representación es relativamente sencillo publicar esos datos en una base de datos orientada a documentos, como ElasticSearch.

ELK es un sistema orientado al registro de documentos y eventos para consulta rápida. El nombre es por estar originalmente integrado por ElasticSearch, Logstash y Kibana, a los que luego se sumaron componentes nuevos [85].

Al igual que en TICK, cada componente cumple una función específica:

- ElasticSearch: almacenamiento de datos.
- Logstash: colecta de datos, originalmente diseñado para interpretación de archivos de log de servidores y sistemas de IT.
- Kibana: panel de administración, consulta y monitoreo (web).

Para este trabajo se utilizaron únicamente ElasticSearch y Kibana.

### 3.7.3. Configuración con Docker Compose

En la figura 3.25 se muestra un diagrama simplificado con la configuración de los microservicios InfluxDB, Elasticsearch, Grafana y Kibana interconectados por una red de interna de Docker, de acuerdo a la configuración de Docker Compose que se detalla en el código 3.5.

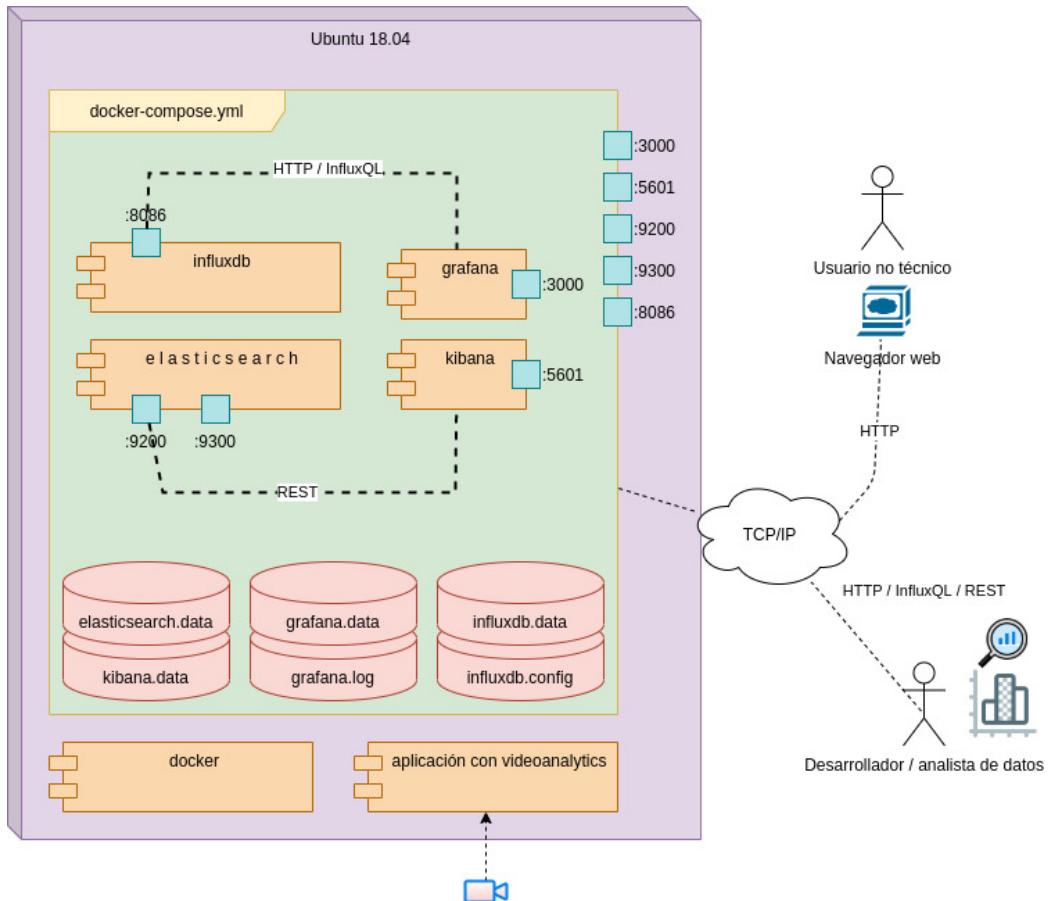


FIGURA 3.25. Configuración de microservicios con Docker Compose (diagrama simplificado). Una aplicación con Videoanalytics escribe datos en las bases de datos en los puertos 8086 (InfluxDB) y 9200 (ElasticSearch), que también son utilizados por los servicios Grafana (3000) y Kibana (5601) para realizar consultas. Un usuario final puede visualizar los datos y realizar consultas en un lenguaje similar a SQL desde un navegador, mientras que un usuario experto además puede acceder directamente a las bases de datos mediante los protocolos REST o InfluxQL sobre HTTP.

Todos los ensayos y aplicaciones con acceso a bases de datos realizadas para este trabajo fueron probados con esa configuración. Las instancias de aplicaciones pueden agregarse como servicios adicionales o ejecutarse por separado.

En la versión actual del sistema los datos se almacenan en volúmenes de Docker. La implementación de políticas de retención, decisiones de particionamiento y redundancia de los datos y otras quedan para una etapa más avanzada del proyecto.

```
1 version: '3.7'
2 services:
3   elasticsearch:
4     image: docker.elastic.co/elasticsearch/elasticsearch:7.9.1
5     ports:
6       - target: 9200
7         published: 9200
8         protocol: tcp
9       - target: 9300
10        published: 9300
11        protocol: tcp
12   environment:
13     discovery.type: single-node
14   ulimits:
15     memlock:
16       soft: -1
17       hard: -1
18   volumes:
19     - type: volume
20       source: elasticsearch.data
21       target: /usr/share/elasticsearch/data
22   networks:
23     - mynetwork
24
25 kibana:
26   depends_on:
27     - elasticsearch
28   image: docker.elastic.co/kibana/kibana:7.9.1
29   ports:
30     - target: 5601
31       published: 5601
32       protocol: tcp
33   volumes:
34     - type: volume
35       source: kibana.data
36       target: /usr/share/kibana/data
37   environment:
38     SERVER_NAME: kibana
39     ELASTICSEARCH_HOSTS: http://elasticsearch:9200
40   networks:
41     - mynetwork
42
43 influxdb:
44   image: influxdb:1.7-alpine
45   volumes:
46     - type: volume
47       source: influxdb.data
48       target: /var/lib/influxdb
49   environment:
50     - INFLUXDB_REPORTING_DISABLED=false
51     - INFLUXDB_LOGGING_LEVEL=info
52     - INFLUXDB_DB=my_application
53   ports:
54     - "8086:8086/tcp"
55   networks:
56     - mynetwork
57
58 grafana:
59   build:
60     context: ./grafana
61   volumes:
62     - type: bind
63       source: ./grafana/dashboards/
```

```

64      target: /dashboards
65      - type: volume
66          source: grafana.data
67          target: /var/lib/grafana
68      - type: volume
69          source: grafana.log
70          target: /var/log/grafana
71  ports:
72      - target: 3000
73          published: 3000
74          protocol: tcp
75  networks:
76      - mynetwork
77
78 networks:
79     mynetwork:
80         driver: bridge
81
82 volumes:
83     elasticsearch.data:
84     grafana.data:
85     grafana.log:
86     kibana.data:
87     influxdb.data:
88     influxdb.config:

```

CÓDIGO 3.5. Archivo de configuración de microservicios con Docker Compose.

La configuración presentada está organizada en tres secciones:

- Servicios: cada servicio queda definido por un identificador de la imagen de Docker (campo “image”, con el nombre indicado en el sitio oficial de cada producto). Además, se indican sus parámetros de configuración, que pueden ser generales, como puertos de red publicados (“port”), en cuyo caso se especifican con directivas de Docker Compose, o específicos de cada aplicación, en cuyo caso se indican con variables de entorno (nombres en mayúsculas) o archivos de configuración separados. También se detallan los directorios que se enlazan con volúmenes.
- Volumenes: son un mecanismo de Docker para persistir los datos generados por los servicios entre ejecuciones, dado que por defecto todos los datos se borran al detenerlos.
- Redes virtuales: al igual que una red real, una red virtual permite a los servicios comunicarse entre sí utilizando nombres de dominio.

Para cada servicio fueron adoptados los parámetros de configuración sugeridos por el fabricante para un escenario de desarrollo y prototipado, que se detallan en el sitio oficial de cada producto. Se describen algunas de ellos:

- Elasticsearch: mientras que en producción es conveniente disponer de múltiples nodos para lograr escalabilidad y robustez, para fines de prototipado es suficiente con un único nodo, reduciendo así el consumo de recursos. También para mejorar la velocidad de respuesta del sistema, puede especificarse que no se realice *swap* de memoria. La sección “memlock” establece que no hay restricciones en la cantidad de memoria que puede ser tomada por el servicio y que no estará disponible para *swap*.

- Kibana: Docker Compose no permite especificar el orden en que se ejecutan los servicios, pero sí indicar dependencias entre ellos. En este caso, se indica que Kibana depende de ElasticSearch.
- Grafana: a diferencia de los otros servicios, Grafana se construye en un archivo separado (“context”). Esto es, entre otras cosas, para poder instalar *plugins* de visualización y eliminar restricciones a los componentes HTML que en algunos casos requieren la ejecución de código *javascript* para comunicarse con otros sitios y mostrar video.

## Capítulo 4

# Ensayos y resultados

En este capítulo se describen los ensayos realizados con los modelos, componentes y procedimientos presentados en el capítulo anterior en aplicaciones representativas de casos de uso reales. Estas aplicaciones responden a los requerimientos del sistema e incluyen detección de objetos (aplicada a capturas y descartes), clasificación de actividades en el caso de pesca de langostinos, registro de eventos (conteo) y consulta en las bases de datos.

### 4.1. Aplicaciones

Para los ensayos que se describen a continuación se utilizó el entorno descripto en el capítulo anterior con los servicios de bases de datos InfluxDB y ElasticSearch y un ambiente de ejecución Conda que contiene el *framework* Videoanalytics instalado junto a todas sus dependencias.

Dado que el sistema tiene como usuarios finales tanto a quienes solo consumen la información totalmente procesada (parte de pesca electrónico) como a usuarios de perfiles más técnicos como desarrolladores y analistas de datos, se realizaron aplicaciones con salidas de distintos niveles de procesamiento que incluyen videos anotados, registros en bases de datos y archivos CSV.

En los ensayos realizados se utilizaron videos, pero sustituyendo el primer componente de cada cadena por otro componente para adquirir video de una cámara se mantiene la misma funcionalidad. También pueden paralelizarse múltiples cadenas de procesamiento que escriban de forma concurrente en las bases de datos.

Todas las aplicaciones se han implementado con una estructura similar en la que se realizan los siguientes pasos:

1. Importar componentes de la librería Videoanalytics.
2. Definir una función para instanciar una cadena de procesamiento parametrizable. Estos parámetros son la entrada, el modelo de detección, nombres de archivos de salida o índices de bases de datos, entre otros.
3. Definir una lista de entradas (lote de archivos a procesar).
4. Procesar en serie todas las entradas, generando las salidas correspondientes en un directorio separado.

Se desarrollaron aplicaciones para los siguientes propósitos:

1. Detección de objetos: se utilizó para generar videos de detecciones anotados y probar los modelos entrenados. El detector de objetos es necesario para el resto de las aplicaciones.
2. Seguimiento de objetos: se utilizó para comparar el desempeño de los algoritmos SORT y DeepSORT con distintos parámetros. El seguimiento de objetos es utilizado para conteo y registro de eventos.
3. Registro y monitoreo de eventos: esta aplicación es la que cumple el requerimiento inicial del sistema. Se apoya en haber verificado el funcionamiento del detector y del seguimiento de objetos. Utilizando los componentes para definición de eventos y registro en bases de datos permite a un usuario final visualizar los resultados y realizar consultas con interfaz web en Grafana y Kibana, o a un usuario con perfil de desarrollador o analista de datos acceder directamente a las bases de datos.
4. Aplicación de identificación de actividades.

Los fragmentos de código que se incluyen en la descripción de la aplicación de detección ejemplifican el uso de la librería para conformar distintas cadenas. Pueden escribirse en cualquier entorno Python en el que se tenga instalado Videoanalytics, siendo recomendado el uso de GPU si se utiliza el modelo de detección de objetos. Para los ensayos el entorno usado fue JupyterLab.

Por no disponer de GPU en el ambiente de pruebas de microservicios, se generaron las detecciones y registraron en CSV una única vez con la aplicación de detección y para el resto de las cadenas de procesamiento se utilizaron detecciones precalculadas.

## 4.2. Aplicación de detección de objetos

El primer paso para cada aplicación es importar los componentes necesarios de la librería Videoanalytics. Existen componentes generales, compartidos por casi todas las cadenas de procesamiento y otros específicos para cada tarea.

Todas las aplicaciones como mínimo instancian el componente Pipeline, una entrada de datos que puede ser un lector de video, cámara o lote de archivos, y generan algún tipo de salida, que pueden ser datos o registros en bases de datos, visualización en vivo, y eventualmente un video que puede utilizarse para demostraciones al cliente. El fragmento de código 4.1 muestra como se importan las clases comunes a todas las aplicaciones.

```
1 from videoanalytics.pipeline import Pipeline
2 from videoanalytics.pipeline.sources import VideoReader
3 from videoanalytics.pipeline.sinks import VideoWriter
```

CÓDIGO 4.1. Importación de clases en Videoanalytics.

Una cadena de detección de objetos además incluirá los componentes específicos de detección. En el fragmento de código 4.2 se muestra como se incluyen el motor

de inferencia YOLOv4 para Tensorflow y otros componentes para anotación y registro. Uno de los criterios de diseño de la biblioteca Videoanalytics fue separar los componentes en módulos tanto por su propósito como por las tecnologías que se usan para implementarlo.

```

1 from videoanalytics.pipeline.sinks.object_detection import
   DetectionsAnnotator, DetectionsCSVWriter
2 from videoanalytics.pipeline.sinks.object_detection.yolo4 import
   YOLOv4DetectorTF

```

CÓDIGO 4.2. Importación de clases específicas para detección de objetos.

Una vez importados los componentes, se procede a instanciar un contexto de intercambio de datos, una cadena de procesamiento y agregar cada uno de los componentes, como se muestra en el fragmento de código 4.3.

```

1 # Crear el contexto global
2 context = {}
3
4 # Crear una cadena
5 pipeline = Pipeline()
6
7 # 3. Agregar componentes
8
9 # 3.1 Fuente de video
10 pipeline.add_component( VideoReader( "input" ,context ,
11                                     video_path=INPUT_VIDEO,
12                                     start_frame=START_FRAME,
13                                     max_frames=MAX_FRAMES ) )
14
15 # 3.2 Detector de objetos
16 pipeline.add_component( YOLOv4DetectorTF("detector" ,
17                                         context ,weights_filename=DETECTOR_WEIGHTS_FILENAME) )
18
19 # 3.3 Registrar detecciones en CSV
20 pipeline.add_component( DetectionsCSVWriter(
21     "det_csv_writer" ,context ,filename=DETECTIONS_FILENAME) )
22
23 # 3.4 Anotar las detecciones en video de salida
24 pipeline.add_component( DetectionsAnnotator(
25     "annotator" ,context ,
26     class_names_filename=DETECTOR_CLASSES_FILENAME,
27     show_label=True) )
28
29 # 3.5 Escritura a video
30 pipeline.add_component(VideoWriter("writer" ,context ,filename=
   OUTPUT_VIDEO))

```

CÓDIGO 4.3. Instanciación de una cadena de procesamiento para detección de objetos.

Una vez agregados todos los componentes a la cadena, se procede a conectarlos indicando el orden de dependencias, como se muestra en el fragmento de código 4.4.

```

1 # 4. Definir conexiones entre componentes
2 pipeline.set_connections([
3     ("input", "detector"),
4     ("detector", "det_csv_writer"),
5     ("detector", "annotator"),
6     ("annotator", "writer")
7 ])

```

CÓDIGO 4.4. Conexión de componentes.

Los componentes conectados conforman un grafo dirigido en el que cada componente se ejecuta una vez por cada iteración respetando el orden de dependencias. El fragmento de código 4.5 genera una visualización del grafo dirigido de la cadena como el que se muestra en la figura 4.1.

```

1 import matplotlib.pyplot as plt
2 fig, axes = plt.subplots(1,1, figsize=(22,8))
3 pipeline.plot(ax=axes)

```

CÓDIGO 4.5. Visualización del grafo de cómputo.

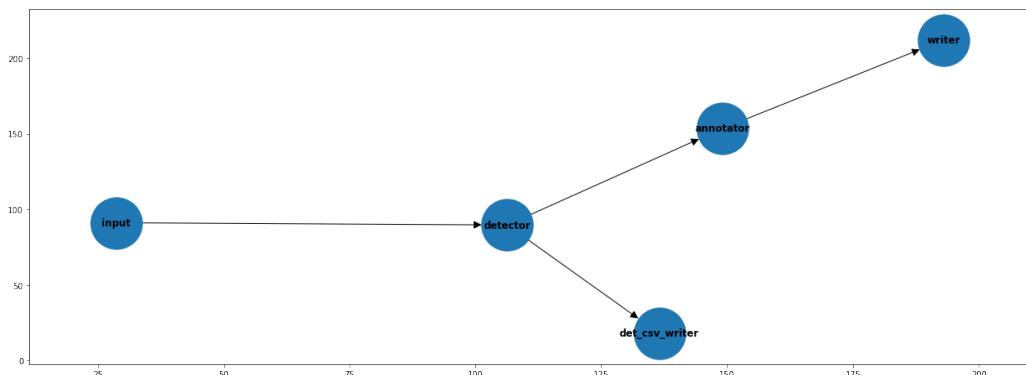


FIGURA 4.1. Visualización del grafo para detección de objetos, que consiste en un lector de video, un detector de objetos con YOLOv4, un componente que registra las detecciones en un archivo CSV, un anotador de las detecciones en video, y un componente que registra el video anotado en un archivo MP4.

Si el componente de una entrada es una cámara, el proceso durará infinitamente hasta que sea interrumpido y si es un archivo MP4, finalizará cuando se haya procesado el último cuadro. Finalizada la ejecución pueden obtenerse el tiempo total de procesamiento como se muestra en la figura 4.6 y el promedio de tiempo de ejecución de cada componente, útil para detectar cuellos de botella en las cadenas y comparar configuraciones.

```

1 # 5. Ejecutar
2 pipeline.execute()
3 print("Tiempo total [s]:", pipeline.get_total_execution_time())

```

CÓDIGO 4.6. Ejecución de la cadena.

La cadena que se describió fue ensayada para dos modelos de detectores de objetos con YOLOv4.

#### 4.2.1. Resultados para modelo de capturas

Antes de ensayar un modelo para una aplicación, se debe verificar que tenga un desempeño mínimo en la partición de evaluación. La tabla 4.1 presenta la matriz de confusión para la evaluación del modelo detector entrenado con el conjunto de datos de Kaggle.

TABLA 4.1. Precisión promedio media (maP) para detector entrenado para detección de presas grandes (datos de Kaggle).

Clase	Nombre	aP	TP	FP
4	shark	94.63	129	11
3	lag	92.08	73	3
6	other	90.95	233	46
5	yft	90.59	495	91
0	alb	90.13	1590	198
2	dol	76.72	68	10
1	bet	66.96	150	35

Como el resultado es satisfactorio, se procedió a probarlo sobre distintos videos. La figura 4.2 contiene algunos ejemplos de escenas donde fue aplicado el detector. En algunos casos la detección es un fin en si mismo, como por ejemplo para identificar cuadros de video en los que aparece una especie protegida, mientras que en otros la detección es un paso previo para poder realizar conteo o registrar otros eventos.

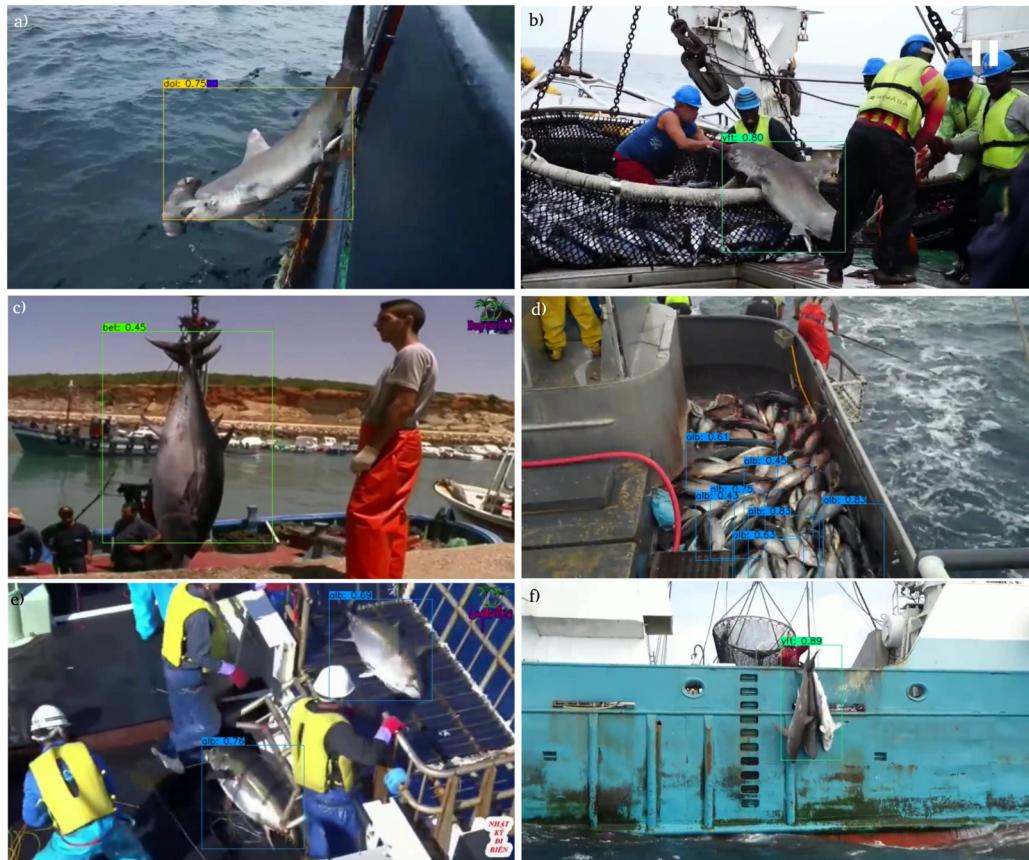


FIGURA 4.2. Resultados de la aplicación de detección sobre videos de dominio público. Aún cuando hay algunos errores que pueden atribuirse a la falta de muestras en los datos de entrenamiento, la detección de piezas parece ser una aproximación válida para identificar especies protegidas. Ejemplos son el tiburón cabeza de martillo (en a, b y f, etiquetado erróneamente pero detectado) y el conteo de atunes grandes (en c y e). Para el conteo de piezas apiladas sobre un área reducida, la detección no parece suficiente, y debería ser combinada con otros métodos (d).

### 4.2.2. Resultados para modelo para aplicación de langostinos

La detección de piezas individuales parece ser una aproximación válida cuando las piezas son grandes y pueden ser distinguidas a partir de cualidades externas como la forma de las aletas o el color.

Para piezas pequeñas o escenas en las que resulte difícil delimitar los objetos, la detección puede ser un paso previo requerido para otro algoritmo, o incluso para una inspección manual, pero acotada a un segmento de interés del video.

La pesca de langostinos es un ejemplo. Resulta imposible detectar de forma individual cada langostino pero, en cambio, sí se pueden detectar los operadores a bordo, las redes cargadas y los pescados grandes -que sin importar su tipo, siempre serán considerados *bycatch*. Detectar los operadores permite estudiar su movimiento y establecer patrones de comportamiento nominal, y esto a su vez luego permite automatizar la detección de patrones anómalos. Detectar las redes es un paso previo para estimar su volumen o para aislar los cuadros de video en que es visible.

Al igual que con el detector de pescados, antes de ensayar el modelo en la aplicación de detección se verificó su desempeño en la partición de evaluación. La tabla 4.2 presenta la matriz de confusión para la evaluación del modelo detector entrenado para el caso de aplicación de pesca de langostinos.

TABLA 4.2. Precisión promedio media (maP) para detector entrenado para problema de langostinos con datos aumentados.

Clase	Nombre	aP	TP	FP
0	operador	93.48	73	10
1	red	78.85	5	0
2	pescado	63.03	22	7

Los resultados en los videos ensayados mostraron algunos errores en que el detector confundió operadores con la red, que se atribuyen también a la baja representatividad del conjunto de datos que se utilizó para entrenamiento y evaluación. Una acción que se puede tomar es agregar ejemplos de operadores con redes de fondo y redes con operadores en el frente. La figura 4.3 muestra algunos de estos ejemplos.

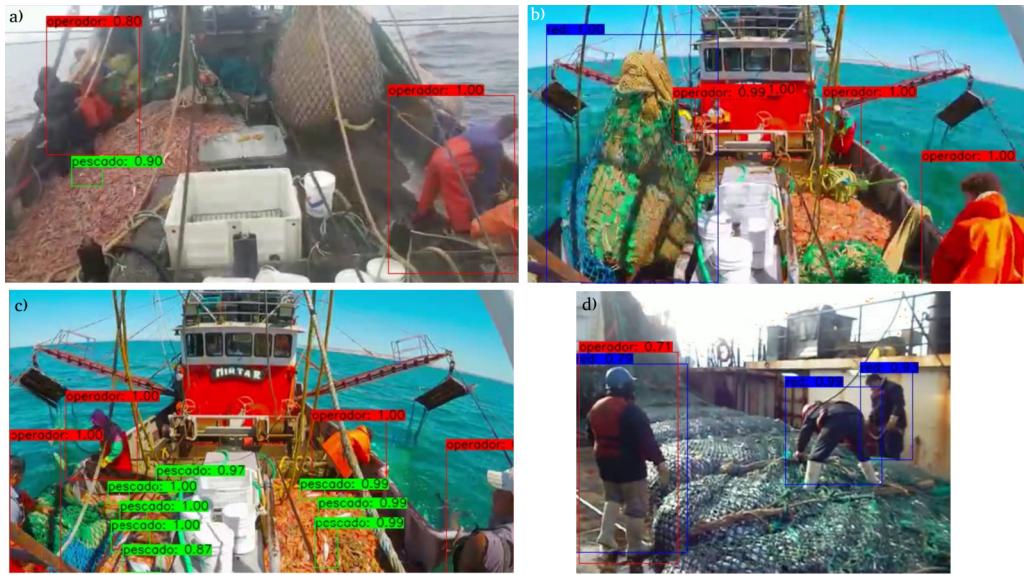


FIGURA 4.3. Resultados de la aplicación de detección de operadores, redes y pescados sobre videos de dominio público. El detector reconoce correctamente los operadores en a, b y c, pero no detecta la red en a y confunde los operadores con redes en d.

### 4.3. Aplicación de seguimiento de objetos

La aplicación de seguimiento utiliza la salida de un detector para construir la trayectoria de los objetos con los componentes SORT o DeepSORT. Agregando la definición de ROIs y el registro de eventos asociados a evaluaciones de presencia de objetos en ellas, puede utilizarse para conteo y otras funciones. El código 4.7 muestra cómo se instancia esta cadena, permitiendo elegir si utilizar el algoritmo SORT o DeepSORT y los parámetros de configuración para cada uno.

La secuencia de configuración de esta cadena es similar a la de detección, con las siguientes diferencias:

1. El detector de objetos YOLO fue sustituido por ObjectDetectorCSV, que obtiene las detecciones previamente generadas en un archivo CSV. Este concepto de mantener una misma interfaz y distintas implementaciones es también aplicable para otros componentes.
2. Es posible parametrizar el tipo de clase que se instanciará para realizar el seguimiento. En este caso se realiza con las funciones `make_*_tracker()`.
3. Los componentes de seguimiento y ROIs mantienen una filosofía general a la de los detectores, separando aspectos de visualización (anotación), de procesamiento o registro de datos.

```

1
2 # Se puede usar un detector o detecciones precalculadas
3 from videoanalytics.pipeline.sinks.object_detection.yolo4 import
   ObjectDetectorCSV
4
5 from videoanalytics.pipeline.sinks.object_tracking import
   TrackedObjectsAnnotator, TrackedObjectsCSVWriter
6
7 # SORT y DeepSORT

```

```
8 from videoanalytics.pipeline.sinks.object_tracking.sort import SORT
9 from videoanalytics.pipeline.sinks.object_tracking.deepsort import
    DeepSORT
10
11
12 def make_tracking_pipeline( input_video_filename ,
13                             output_track_csv_filename ,
14                             output_video_filename ,
15                             detector_precalc_filename ,
16                             detector_model_classes_filename ,
17                             roi_definition_filename ,
18                             tracker_type="SORT" ,
19                             start_frame=0 ,
20                             max_frames=None , tracker_args={} ) :
21     context = {
22         "TRACKED_OBJS" : []
23     }
24     pipeline = Pipeline()
25
26     pipeline.add_component( VideoReader( "input" , context ,
27                                         video_path=input_video_filename ,
28                                         start_frame=start_frame ,
29                                         max_frames=max_frames) )
30
31
32     pipeline.add_component( ObjectDetectorCSV( "detector_precalc" ,
33                                              context ,
34                                              filename =
35                                              detector_precalc_filename ) )
36
37
38     def make_sort_tracker( tracker_args ):
39         print("Making SORT")
40         return SORT( "tracker" , context , **tracker_args )
41
42     def make_deep_sort_tracker( tracker_args ):
43         print("Making DeepSORT")
44         return DeepSORT( "tracker" , context , **tracker_args )
45
46     tracker_factory = {
47         "SORT": make_sort_tracker ,
48         "DEEPSORT": make_deep_sort_tracker
49     }
50
51     pipeline.add_component( tracker_factory[ tracker_type ]( tracker_args ) )
52
53     pipeline.add_component( TrackedObjectsAnnotator( "annotator" , context ) )
54
55     pipeline.add_component( TrackedObjectsCSVWriter( "tracker_csv_writer" ,
56                                                    context ,
57                                                    filename =
58                                                    output_track_csv_filename ) )
59
60     pipeline.add_component( VideoWriter( "writer" , context , filename=
61                                         output_video_filename ) )
62
63     pipeline.set_connections([
64         ("input" , "detector") ,
65         ("detector" , "tracker") ,
66         ("tracker" , "tracker_csv_writer") ,
67         ("tracker" , "annotator") ,
68         ("annotator" , "writer") ])
```

```

64
65
66
    return context, pipeline
])

```

CÓDIGO 4.7. Configuración de cadena de procesamiento para seguimiento de objetos

El grafo mantiene la filosofía de la aplicación de detección. Las detecciones, que se registran en el contexto compartido -y son independientes del modelo de detector que se utilizó- son consultadas por el algoritmo de seguimiento, que también dejará las trayectorias en el contexto para que puedan ser consultadas por otros componentes.

La figura 4.4 muestra el grafo de cómputo para seguimiento de objetos generado a partir del código anterior.

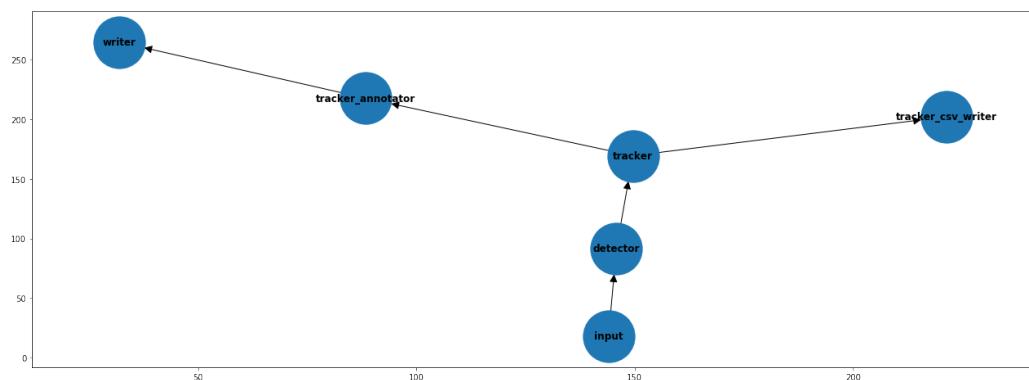


FIGURA 4.4. Visualización del grafo para seguimiento de objetos, que consiste en un lector de video, un detector de objetos, el algoritmo de seguimiento que puede ser SORT o DeepSORT, un componente que registra las trayectorias en un archivo CSV, un anotador de los objetos identificados en video, y un componente que registra el video anotado en un archivo MP4.

### 4.3.1. Caso de uso: conteo con ROIs

Este tipo de configuración, combinado con la definición de ROIs, permite contar capturas cuando se trata de piezas grandes, como se muestra en los dos casos de pesca de atún de la figura 4.5.

### 4.3.2. Configuración de SORT y DeepSORT

En la sección 2.5 se mencionó que los algoritmos de seguimiento, además de ser necesarios para la generación de trayectorias, pueden ser también de utilidad para complementar a los detectores de objetos.

Puede ocurrir que un objeto en determinados ángulos, o por la iluminación, ocultación u otros factores, no sea detectado. En estos casos puede utilizarse un algoritmo de seguimiento para continuar la ruta estimada que tendría ese objeto hasta que haya una detección disponible. También es posible que el detector falle y active falsos positivos.



FIGURA 4.5. Aplicación con seguimiento y ROIs para conteo de atunes. En la imagen de la izquierda se definen dos regiones, mar (verde) y cubierta (roja). El contador puede incrementarse cuando un objeto sale de la región verde e ingresa a la región roja (para evitar que los objetos que aparecen inmediatamente en la región roja incrementen el contador). En la imagen de la derecha se define solo una región.

Es una capacidad útil poder configurar los algoritmos de seguimiento para tener control sobre estos aspectos, habiendo establecido un mínimo de detecciones antes de decidir el registro y seguimiento de un nuevo objeto y un máximo de cuadros que se puede mantener un objeto sin haber recibido una nueva detección. También, de acuerdo a la velocidad de los objetos en seguimiento y la cantidad de cuadros por segundo del video es de interés estimar la velocidad en pixels por segundo que tendrán los objetos para así definir valores óptimos de la distancia máxima que puede tener un objeto entre dos cuadros consecutivos.

Estos parámetros son *min\_hits*, *max\_age* e *iou\_threshold* o *max\_iou\_distance* respectivamente, y son compartidos por las implementaciones de SORT y DeepSORT. En DeepSORT, además, es posible especificar el modelo de extractor que se utilizará para la apariencia (*model\_filename*) y la distancia a partir de la cual dos objetos se consideran distintos por su apariencia (*max\_cosine\_distance*). El fragmento de código 4.8 muestra los parámetros que se eligieron para los ensayos de conteo de atunes.

```

1 s_tracker_type="SORT"
2 s_tracker_args= {
3     "max_age":10,
4     "min_hits": 1,
5     "iou_threshold":0.05
6 }
7
8 ds_tracker_type="DEEPSORT"
9 ds_tracker_args= {
10     "model_filename": "mars-small128.pb",
11     "max_cosine_distance": 0.4,
12     "max_iou_distance": 0.7,
13     "max_age": 60,
14     "min_hits": 3
15 }
```

CÓDIGO 4.8. Importación de clases generales.

En DeepSORT se utilizó el extractor por defecto que se provee con el algoritmo, orientado a personas. Debería haberse utilizado un modelo de extractor especialmente entrenado para atunes y otros pescados, pero preparar los datos de entrenamiento escapaba al alcance del proyecto en esta etapa.

#### 4.4. Monitoreo e integración con bases de datos

Las aplicaciones que se presentaron permiten extraer información de interés de los videos pero, con los componentes que se instanciaron hasta ahora, esa información sólo está disponible en archivos CSV.

Existen componentes en Videoanalytics para procesar las detecciones y trayectorias disponibles en el contexto para generar nuevas variables, registrar eventos, y publicar esta información en las bases de datos InfluxDB y ElasticSearch.

Las herramientas Grafana y Kibana permiten realizar consultas y construir paneles con gráficos y estadísticas para mostrar esta información en vivo, además de permitir realizar consultas complejas, que se escriben en un lenguaje similar a SQL.

La figura 4.6 muestra un ejemplo de un panel de monitoreo de algunas de las variables publicadas en InfluxDB, en este caso describiendo la cantidad de operadores en cada sector de un buque como por ejemplo el malacate y los tangones.

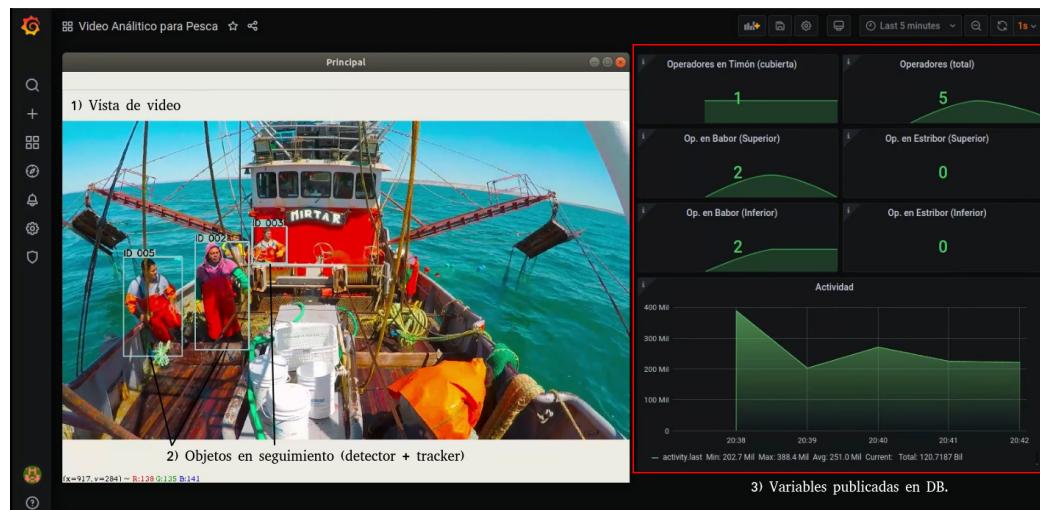


FIGURA 4.6. Integración con InfluxDB y Grafana en aplicación de conteo. Las variables de contadores se publican en InfluxDB y se visualizan en vivo en Grafana.

En el caso de Elasticsearch, se pueden realizar consultas para obtener, por ejemplo, todos los momentos del video donde aparece un determinado objeto, o buscar todos los momentos en los que se superpone el área de un objeto con otro, por ejemplo un operador con un pescado, indicando algún tipo de manipulación sospechosa que deba estudiarse. La figura 4.7 muestra una captura de pantalla del panel de consultas.

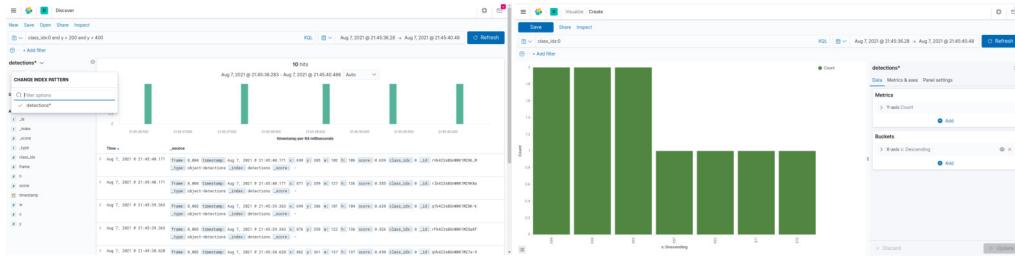


FIGURA 4.7. Integración con ElasticSearch y Kibana en aplicación de registro de detecciones. Cada detección queda registrada en la base de datos con su estampa de tiempo, región de la imagen, y otros datos. La captura de pantalla corresponde al panel de consultas, que utiliza un lenguaje similar a SQL.

#### 4.4.1. Requisitos de cómputo

Uno de los justificativos de diseño por los cuales se adoptó el paradigma de construcción de cadenas de procesamiento a partir de componentes, es poder evaluar distintas configuraciones, detectar cuellos de botella y determinar cuál es la forma óptima de resolver un problema. La figura 4.8 muestra el tiempo de cómputo de cada uno de los componentes que integran una aplicación con un detector, algoritmo de seguimiento y escritura en bases de datos, ejecutado en un ambiente de prestaciones reducidas (sin GPU). Se puede observar que el cuello de botella es el detector. De acuerdo a los autores de YOLOv4, se pueden superar 30 FPS para una configuración como la que se está utilizando en la que la entrada es de 416x416 píxeles si se dispone del hardware apropiado. Se concluye, por lo tanto, que la solución propuesta cumple el requerimiento de desempeño.

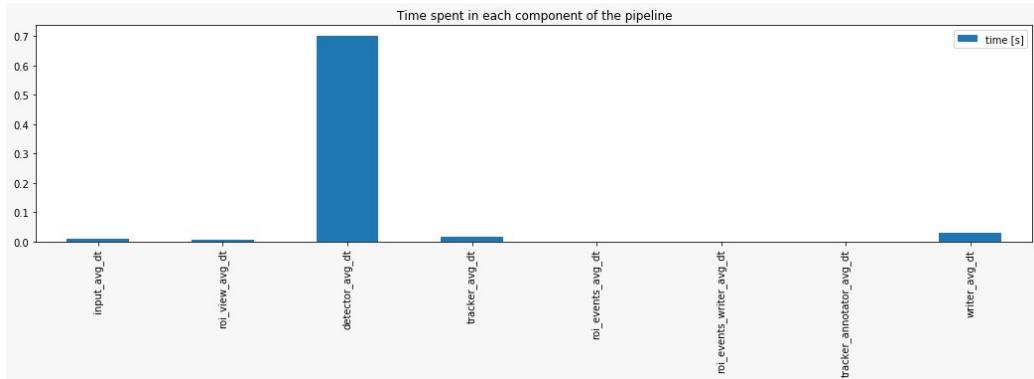


FIGURA 4.8. Tiempo de cómputo por cada componente para una aplicación con detección, seguimiento y publicación en bases de datos. La medición corresponde a un ambiente sin GPU. Se puede observar que el cuello de botella es el detector.

## 4.5. Aplicación de clasificación de actividades

En la sección 3.4 se describió un procedimiento para generar un modelo que permita clasificar las actividades a bordo de un buque pesquero, partiendo del supuesto de que esas actividades están asociadas a los movimientos de los operadores.

Los pasos de preparación de datos, incluyendo la generación de variables estadísticas, fueron descriptos como parte del procedimiento.

En cada caso, deberá estudiarse qué variables son requeridas. Una forma de hacerlo es mediante los tests de correlación, ANOVA e importancia mutua, cuyos resultados para los datos ensayados se muestran en la figura 4.9.

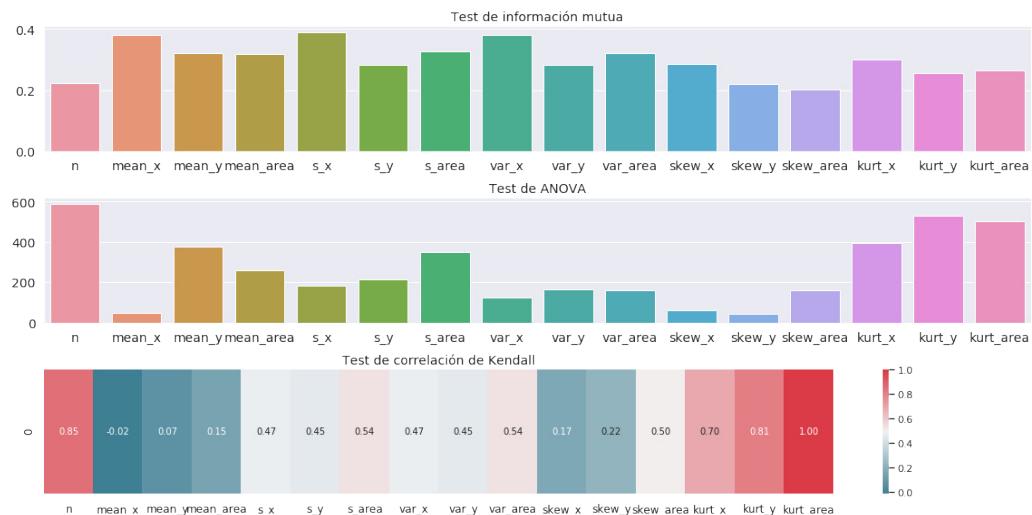


FIGURA 4.9. Test de información mutua, test de ANOVA y test de correlación de Kendall. Puede atribuirse la importancia de la cantidad de operadores a que existe mayor presencia en los momentos de descarga de redes. La kurtosis también puede explicarse como influyente porque en los momentos de descanso los operadores tienden a estar fijos en una posición, mientras que en la clasificación se desplazan e intercambian posiciones.

Siguiendo con el procedimiento, luego de haber realizado un estudio previo de las variables, se procedió al entrenamiento de algunos modelos. En todos los casos se implementó un paso de estandarización de las variables de entrada. En algunos casos se agregaron otras transformaciones como la transformación de cuantil para evaluar si tenían un impacto positivo en el desempeño.

Los modelos entrenados fueron:

- Regresión logística con y sin transformación de cuantil [86, 87].
- AdaBoost [88].
- RandomForest [89].
- GradientBoost [90].
- KNN [91].

Para cada modelo se registró el puntaje sobre la partición de entrenamiento y de evaluación con la métrica *AUC* y el tiempo de inferencia promedio por muestra en segundos (obtenido sobre la partición de evaluación).

Como se puede observar en la figura 4.10, en general todos los modelos tienen un muy buen desempeño, superando el 70 %.

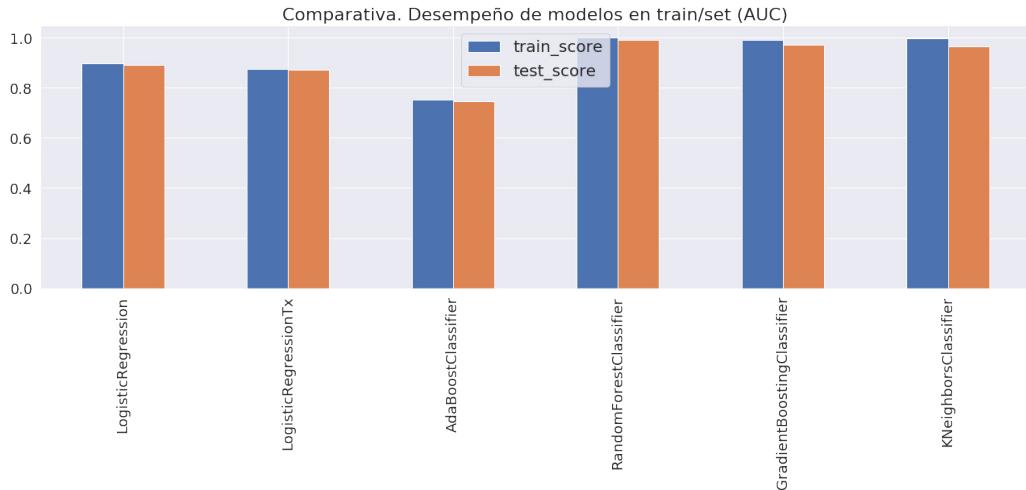


FIGURA 4.10. Desempeño de los modelos de clasificación (AUC).

Al comparar el tiempo de inferencia, es evidente que el algoritmo KNN tiene un costo computacional significativamente mayor al del resto de los modelos, como se puede observar en la figura 4.11.

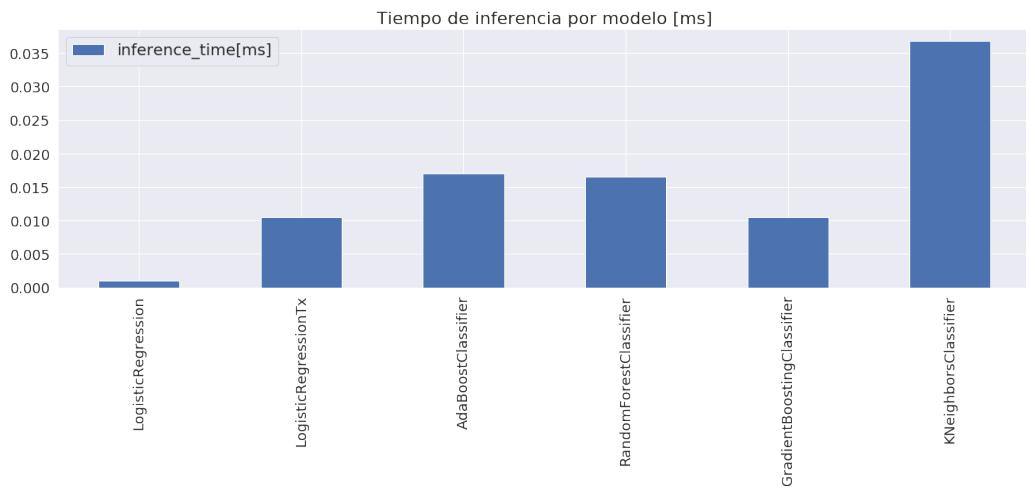


FIGURA 4.11. Tiempo de inferencia de cada modelo.

La figura 4.12 muestra los tiempos de inferencia excluyendo a KNN. Un resultado de interés es el bajo costo computacional del modelo de regresión logística y su excelente relación costo/desempeño. Esto lo posiciona como una alternativa de implementación apropiada para ambientes con capacidades de cómputo limitadas, permitiendo destinar recursos escasos como GPUs para otras tareas más críticas.

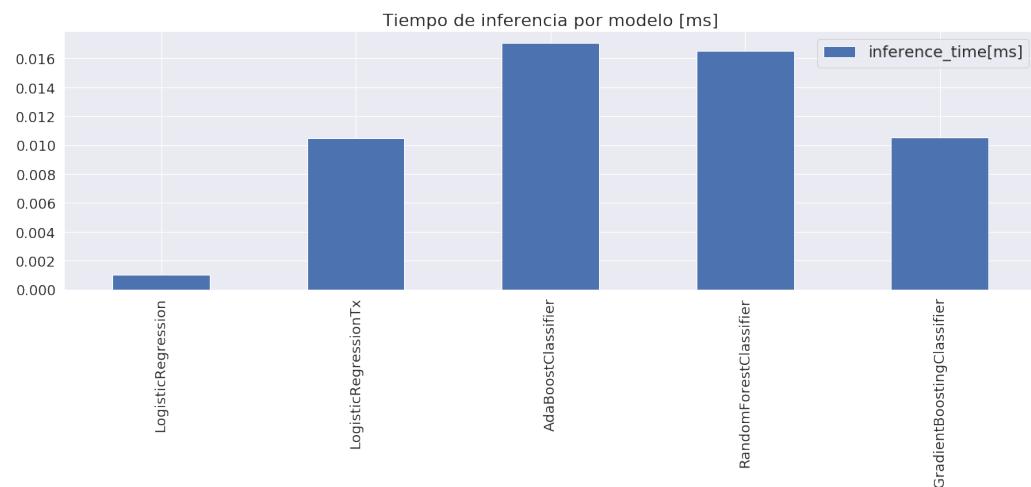


FIGURA 4.12. Tiempo de inferencia de cada modelo, excluyendo KNN.

## Capítulo 5

# Conclusiones

En este capítulo se analizan los resultados obtenidos y el grado de cumplimiento de los objetivos. Además, se describen los pasos a seguir para una siguiente etapa de desarrollo.

### 5.1. Resultados obtenidos

El objetivo de este trabajo fue implementar un prototipo de sistema de video analítico que permitiera fiscalizar la actividad a bordo de buques pesqueros. Por tratarse de un área desconocida tanto en lo que respecta al dominio del problema como de las técnicas de inteligencia artificial aplicadas, era una restricción de diseño que pudiera extenderse y adecuarse a redefiniciones de alcance y nuevos escenarios. También debía resolverse de un modo que permitiera la participación de desarrolladores y analistas de datos que se incorporaran al proyecto en el futuro.

En la propuesta inicial, la automatización de la fiscalización consistía únicamente en detectar las presas y registrar eventos mediante un algoritmo de seguimiento y regiones poligonales especificadas por el usuario. Los experimentos que se realizaron a medida que avanzaba el proyecto y un análisis más profundo del problema pusieron en evidencia que si bien esta aproximación era adecuada en algunos tipos de escenas, también con los componentes implementados, aplicados de distinta forma, se podían obtener otros resultados útiles.

Ya iniciado el proyecto se difundió en los medios un creciente interés por la explotación del langostino [92], a lo que se sumó la confirmación de que la prueba piloto para la que se obtendrían videos sería en un buque operativo en Río Negro, probablemente para este tipo de captura. Esta noticia, sumada a la ausencia de material para entrenar y evaluar los modelos para otras especies motivó a destinar esfuerzos a la aplicación de clasificación de actividades y contar con un mecanismo para detectar las redes, paso requerido para luego poder estimar su volumen. En ambos casos se utilizó el detector de objetos de la propuesta inicial reentrenado con otro conjunto de datos.

Se considera que el mayor logro de este trabajo fue la capacidad de haberse adaptado a cambios de requerimientos y ambientes, además de haber generado un *framework* propio. Este *framework* puede continuar desarrollándose tanto en el ámbito académico y de la comunidad de código abierto como adentro del equipo de desarrollo de aplicaciones de inteligencia artificial y ciencia de datos de INVAP S.E.

Más allá de las dificultades y los cambios de alcance, fue posible cumplir con la planificación original. En este aspecto, fue clave la predisposición del cliente para redefinir los objetivos de acuerdo a los datos y recursos con los que se disponía en cada momento. Además, no fue necesario implementar el componente de calibración de la cámara porque no se disponía de una forma de evaluarlo con el equipo final (durante el desarrollo fueron utilizados videos obtenidos de cámaras sin especificar). El esfuerzo destinado esa tarea fue reasignado a los componentes requeridos para los cambios de alcance mencionados.

## 5.2. Próximos pasos

La automatización de tareas realizadas por operadores mediante inteligencia artificial es un proceso gradual. Este trabajo puede continuar extendiéndose para la detección y conteo de otros tipos de pescados e incorporar técnicas de estimación de volumen, pero a medida que se incorporen capacidades automáticas, deben también considerarse mecanismos para garantizar la robustez del sistema y detección de fallas o posibles comportamientos anómalos.

Si bien se automatizan algunas tareas, la intervención humana es imprescindible. Por lo tanto, siempre será necesaria la supervisión y colaboración con expertos tanto en el dominio del problema como de inteligencia artificial para lograr productos más complejos y críticos. Esto requiere avances tanto en los aspectos técnicos como en los procesos que coordinan la gestión de los recursos y la interacción entre especialistas.

En esta dirección, algunas líneas de trabajo que pueden considerarse para el futuro son:

- Incorporación de herramientas para la gestión de los información y el ciclo completo de desarrollo de modelos que faciliten trabajar con grandes volúmenes de datos, que contemplen aspectos de seguridad, confidencialidad y uso concurrente, entre otros. Algunas herramientas a considerar pueden ser CVAT [93] y Apache Airflow [94].
- Mejorar la explicabilidad e interpretabilidad de los modelos. Para modelos de alta complejidad como YOLOv4 no existen herramientas, algoritmos o metodologías de uso inmediato. Sin embargo, existen publicaciones que describen avances en esta área [95].
- Incorporar modelos de detectores, extractores y otros algoritmos para cubrir más escenarios o mejorar el desempeño en los actuales.
- Extender el uso de regiones poligonales a imágenes donde la cámara pueda tener movimientos de zoom o desplazamiento.
- Establecer metodologías para que este proyecto pueda evolucionar combinando componentes de código abierto mantenido en la comunidad y el ámbito académico con componentes propietarios, permitiendo exponer aspectos de su funcionamiento sin comprometer información sensible.

# Bibliografía

- [1] Andrea Pagani y Patricia Gualdoni. *2do Informe de Monitoreo Ciudadano - Sector Pesquero*. Visitado el 2021-09-05. 2018. URL: <http://nulan.mdp.edu.ar/3004/1/pagani-gualdoni-2018.pdf>.
- [2] Consejo Federal Pesquero. Visitado el 2021-09-05. URL: <https://cfp.gob.ar/>.
- [3] Erik Bergh y Sandy Davies. «A Fishery Manager's Guidebook - Management Measures and Their Application». En: Visitado el 2021-09-05. FAO (Food y Agriculture Organization of the United Nations), 2002. URL: <http://www.fao.org/3/y3427e/y3427e0a.htm#bm10.1>.
- [4] Consejo Federal Pesquero - Resolución 12 / 2016 - Pesquería de Centolla - Medidas de Ordenamiento y Administración. Visitado el 2021-09-05. URL: <https://www.argentina.gob.ar/normativa/nacional/resoluci%C3%B3n-12-2016-266952>.
- [5] Seguimiento electrónico: una herramienta clave para las pesquerías a nivel global. Cómo los Gobiernos y las organizaciones regionales de ordenación pesquera pueden controlar mejor las flotas en altamar. Visitado el 2021-09-05. 2019. URL: [https://www.pewtrusts.org/-/media/assets/2019/09/esla\\_electronicmonitoringkeytool\\_v3.pdf](https://www.pewtrusts.org/-/media/assets/2019/09/esla_electronicmonitoringkeytool_v3.pdf).
- [6] Aloysius T. M. van Helmond and Chun Chen and Jan Jaap Poos. How effective is electronic monitoring in mixed bottom-trawl fisheries? Visitado el 2016-06-25. 2014. URL: doi:10.1093/icesjms/fsu200.
- [7] Boaz Zion. «The use of computer vision technologies in aquaculture – A review». En: *Computers and Electronics in Agriculture* 88 (2012), págs. 125-132. ISSN: 0168-1699. DOI: <https://doi.org/10.1016/j.compag.2012.07.010>. URL: <https://www.sciencedirect.com/science/article/pii/S0168169912001950>.
- [8] Geoffrey French y col. «Deep neural networks for analysis of fisheries surveillance video and automated monitoring of fish discards». English. En: *ICES Journal of Marine Science* 77.4 (jul. de 2020), 1340–1353. ISSN: 1054-3139. DOI: [10.1093/icesjms/fsz149](https://doi.org/10.1093/icesjms/fsz149).
- [9] Simone Marini y col. «Tracking Fish Abundance by Underwater Image Recognition». En: *Scientific Reports* 8 (1018). ISSN: 2045-2322. DOI: [10.1038/s41598-018-32089-8](https://doi.org/10.1038/s41598-018-32089-8). URL: <https://doi.org/10.1038/s41598-018-32089-8>.
- [10] Geoffrey French y col. «Convolutional Neural Networks for Counting Fish in Fisheries Surveillance Video». En: *Proceedings of the Machine Vision of Animals and their Behaviour (MVAB)*. Ed. por T. Amaral y col. BMVA Press, 2015, págs. 7.1-7.10. ISBN: 1-901725-57-X. DOI: [10.5244/C.29.MVAB.7](https://dx.doi.org/10.5244/C.29.MVAB.7). URL: <https://dx.doi.org/10.5244/C.29.MVAB.7>.
- [11] Mark R. Shortis y col. «Progress in the Automated Identification, Measurement, and Counting of Fish in Underwater Image Sequences». En: *Marine Technology Society Journal* 50.1 (2019). URL: <https://doi.org/10.1016/j.marpol.2019.103554>.

- [12] Petri Suuronen y Eric Gilman. «Monitoring and managing fisheries discards: New technologies and approaches». En: *Marine Policy* 116 (2020), pág. 103554. ISSN: 0308-597X. DOI: <https://doi.org/10.1016/j.marpol.2019.103554>. URL: <https://www.sciencedirect.com/science/article/pii/S0308597X18308716>.
- [13] Kaggle. *Kaggle - The Nature Conservancy Fisheries Monitoring Can you detect and classify species of fish?* Visitado el 2021-09-05. URL: <https://www.kaggle.com/c/the-nature-conservancy-fisheries-monitoring>.
- [14] INVAP S.E. Visitado el 2021-09-05. URL: <https://www.invap.com.ar>.
- [15] INIDEP. Visitado el 2021-09-05. URL: <https://www.argentina.gob.ar/inidep>.
- [16] *A history of CCTV technology. How video surveillance technology has evolved.* Visitado el 2021-09-05. URL: <https://www.surveillance-video.com/blog/a-history-of-cctv-technology-how-video-surveillance-technology-has-evolved.html/>.
- [17] Samsung SDS. Visitado el 2021-09-05. URL: <https://www.samsungsds.com/us/>.
- [18] Bosch Security Video Analytics. Visitado el 2021-09-05. URL: <https://www.boschsecurity.com/xc/en/solutions/video-systems/video-analytics/>.
- [19] OpenDataCam. Visitado el 2021-09-05. URL: <https://github.com/opendatacam/opendatacam>.
- [20] NVIDIA DeepStream SDK. Visitado el 2021-09-05. URL: <https://developer.nvidia.com/deepstream-sdk>.
- [21] VideoFlow. *VideoFlow*. Visitado el 2021-09-05. URL: <https://pypi.org/project/videoflow/>.
- [22] VidGear. *VidGear*. Visitado el 2021-09-05. URL: <https://pypi.org/project/vidgear/>.
- [23] Scikit-learn. *Scikit-learn*. Visitado el 2021-09-05. URL: <https://scikit-learn.org/>.
- [24] Transferencia de Estado Representacional (REST). Visitado el 2021-09-25. URL: [https://es.wikipedia.org/wiki/Transferencia\\_de\\_Estado\\_Representacional](https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional).
- [25] Extensible Markup Language (XML). Visitado el 2021-09-25. URL: [https://es.wikipedia.org/wiki/Extensible\\_Markup\\_Language](https://es.wikipedia.org/wiki/Extensible_Markup_Language).
- [26] JSON. Visitado el 2021-09-25. URL: <https://es.wikipedia.org/wiki/JSON>.
- [27] Edge computing. Visitado el 2021-09-25. URL: [https://en.wikipedia.org/wiki/Edge\\_computing](https://en.wikipedia.org/wiki/Edge_computing).
- [28] Jetson Xavier NX. Visitado el 2021-09-25. URL: <https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-xavier-nx/>.
- [29] Joseph Redmon y col. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: [1506.02640 \[cs.CV\]](https://arxiv.org/abs/1506.02640).
- [30] Nicolai Wojke, Alex Bewley y Dietrich Paulus. «Simple Online and Realtime Tracking with a Deep Association Metric». En: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, págs. 3645-3649. DOI: [10.1109/ICIP.2017.8296962](https://doi.org/10.1109/ICIP.2017.8296962).
- [31] Laura Leal-Taixé, Cristian Canton Ferrer y Konrad Schindler. *Learning by tracking: Siamese CNN for robust target association*. 2016. arXiv: [1604.07866 \[cs.LG\]](https://arxiv.org/abs/1604.07866).

- [32] Nicolai Wojke y Alex Bewley. «Deep Cosine Metric Learning for Person Re-identification». En: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, págs. 748-756. DOI: [10.1109/WACV.2018.00087](https://doi.org/10.1109/WACV.2018.00087).
- [33] Alex Bewley y col. «Simple online and realtime tracking». En: *2016 IEEE International Conference on Image Processing (ICIP)* (2016). DOI: [10.1109/icip.2016.7533003](https://doi.org/10.1109/icip.2016.7533003). URL: <http://dx.doi.org/10.1109/ICIP.2016.7533003>.
- [34] Shengyong Chen y col. «Deep Learning for Multiple Object Tracking: A Survey». En: *IET Computer Vision* 13 (ene. de 2019). DOI: [10.1049/iet-cvi.2018.5598](https://doi.org/10.1049/iet-cvi.2018.5598).
- [35] FarhodovXurshedjon y col. «LSTM Network with Tracking Association for Multi-Object Tracking». En: *Journal of Korea Multimedia Society* 23.10 (oct. de 2020), págs. 1236-1249.
- [36] Sebastian Raschka. *Python Machine Learning*. Packt Publishing, 2015. ISBN: 9781783555130; 1783555130.
- [37] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [38] Ryan Gillard Valliappa Lakshmanan Martin Görner. *Practical Machine Learning for Computer Vision: End-to-End Machine Learning for Images*. O'Reilly Media, Inc, USA, 2021. ISBN: 9781098102364; 1098102363.
- [39] Andrew Ng y Kian Katanforooosh. *CS230 Section 8. Advanced Evaluation Metrics. Object Detection: IoU, AP, and mAP*. Visitado el 2021-09-16. URL: <https://cs230.stanford.edu/section/8>.
- [40] *Visual Object Classes Challenge 2012 (VOC2012)*. URL: <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html>.
- [41] COCO - Common Objects in Context. URL: <https://cocodataset.org/#home>.
- [42] Syed Sahil Abbas Zaidi y col. *A Survey of Modern Deep Learning based Object Detection Models*. 2021. arXiv: [2104.11892 \[cs.CV\]](https://arxiv.org/abs/2104.11892).
- [43] Ross Girshick y col. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: [1311.2524 \[cs.CV\]](https://arxiv.org/abs/1311.2524).
- [44] Ross Girshick. *Fast R-CNN*. 2015. arXiv: [1504.08083 \[cs.CV\]](https://arxiv.org/abs/1504.08083).
- [45] Shaoqing Ren y col. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: [1506.01497 \[cs.CV\]](https://arxiv.org/abs/1506.01497).
- [46] Wei Liu y col. «SSD: Single Shot MultiBox Detector». En: *Lecture Notes in Computer Science* (2016), 21–37. ISSN: 1611-3349. DOI: [10.1007/978-3-319-46448-0\\_2](https://doi.org/10.1007/978-3-319-46448-0_2). URL: [http://dx.doi.org/10.1007/978-3-319-46448-0\\_2](http://dx.doi.org/10.1007/978-3-319-46448-0_2).
- [47] Umberto Muchelucci. *Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection*. Apress901 Grayson Street Suite 204 Berkely, CAUnited States, 2019.
- [48] Joseph Redmon y Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: [1612.08242 \[cs.CV\]](https://arxiv.org/abs/1612.08242).
- [49] Joseph Redmon y Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: [1804.02767 \[cs.CV\]](https://arxiv.org/abs/1804.02767).
- [50] Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: [2004.10934 \[cs.CV\]](https://arxiv.org/abs/2004.10934).
- [51] Anthony D. Rhodes. *A Overview of Computer Vision Tasks, including Multiple-Object Detection (MOT)*. Visitado el 2021-09-16. Mayo de 2018. URL: [http://web.pdx.edu/~arhodes/CV\\_MOT.pdf](http://web.pdx.edu/~arhodes/CV_MOT.pdf).

- [52] *Algoritmo húngaro*. Visitado el 2021-09-05. URL: [https://es.wikipedia.org/wiki/Algoritmo\\_h%C3%BAngaro](https://es.wikipedia.org/wiki/Algoritmo_h%C3%BAngaro).
- [53] Docker. *Docker*. Visitado el 2021-09-16. URL: <https://www.docker.com/>.
- [54] Nicolás Eduardo Horro. *videoanalytics*. Visitado el 2021-09-05. URL: <https://github.com/nhorro/videoanalytics>.
- [55] Nicolás Eduardo Horro. *videoanalytics*. Visitado el 2021-09-05. URL: <https://pypi.org/project/videoanalytics/>.
- [56] Nicolás Eduardo Horro. *videoanalytics*. Visitado el 2021-09-05. URL: <https://readthedocs.org/projects/videoanalytics/>.
- [57] Conda. *Sistema de paquetes Conda*. Visitado el 2021-09-16. URL: <https://docs.conda.io/en/latest/>.
- [58] Jupyter. Visitado el 2021-09-25. URL: <https://jupyter.org/>.
- [59] Darknet. *Darknet*. Visitado el 2021-09-05. URL: <https://pjreddie.com/darknet/>.
- [60] Tensorflow. *Tensorflow*. Visitado el 2021-09-05. URL: <https://www.tensorflow.org/>.
- [61] Siddharth Sharma y col. *Speed up TensorFlow Inference on GPUs with TensorRT*. Visitado el 2021-09-05. URL: <https://blog.tensorflow.org/2018/04/speed-up-tensorflow-inference-on-gpus-tensorRT.html>.
- [62] Open Neural Network Exchange. *Open Neural Network Exchange*. Visitado el 2021-09-05. URL: <https://onnx.ai/>.
- [63] Pablo Briff and Magdalena Bouza and Nicolás Horro. *Repositorio del curso de Análisis de Datos de la Especialización en Inteligencia Artificial de la UBA. Presentaciones, apuntes, y prácticas*. Visitado el 2021-09-16. 2020. URL: [https://github.com/nhorro/ceia\\_add2021](https://github.com/nhorro/ceia_add2021).
- [64] OpenCV. *OpenCV*. Visitado el 2021-09-05. URL: <https://opencv.org/>.
- [65] Viet Hung. *tensorflow-yolov4-tflite*. Visitado el 2021-09-05. URL: <https://github.com/hunglc007/tensorflow-yolov4-tflite>.
- [66] *Maniobra realizada para la pesca de langostino en el buque tan-gonero "Mirta R" en la zona de Playa Unión, en la provincia de Chubut*. Visitado el 2021-09-05. URL: [https://www.youtube.com/watch?v=zGqXKR\\_wvFU](https://www.youtube.com/watch?v=zGqXKR_wvFU).
- [67] TzuTa Lin. *LabelImg*. Visitado el 2021-09-05. URL: <https://github.com/tzutalin/labelImg>.
- [68] Luis Perez y Jason Wang. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. arXiv: [1712.04621 \[cs.CV\]](https://arxiv.org/abs/1712.04621).
- [69] Alexander Jung. *imgaug*. Visitado el 2021-09-05. URL: <https://github.com/aleju/imgaug>.
- [70] Consejo Federal Pesquero - Resolucion 7 (17-05-18) *Medidas de administración de langostino*. Visitado el 2021-09-05. URL: [https://cfp.gob.ar/resoluciones/Resolucion%207%20\(17-05-18\)%20Medidas%20de%20administracion%20langostino.pdf](https://cfp.gob.ar/resoluciones/Resolucion%207%20(17-05-18)%20Medidas%20de%20administracion%20langostino.pdf).
- [71] Daniel Bertuche y col. *Langostino (Pleoticus muelleri)*. Visitado el 2021-09-05. URL: <https://www.inidep.edu.ar/wp-content/uploads/Langostino.pdf>.
- [72] INIDEP. *Langostino - Infografía de divulgación*. Visitado el 2021-09-05. URL: [https://www.argentina.gob.ar/sites/default/files/inidep\\_infografia\\_langostino.pdf](https://www.argentina.gob.ar/sites/default/files/inidep_infografia_langostino.pdf).
- [73] N. V. Chawla y col. «SMOTE: Synthetic Minority Over-sampling Technique». En: *Journal of Artificial Intelligence Research* 16 (2002), 321–357. ISSN: 1076-9757. DOI: [10.1613/jair.953](https://doi.org/10.1613/jair.953). URL: <http://dx.doi.org/10.1613/jair.953>.

- [74] Haibo He y col. «ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning». En: jul. de 2008, págs. 1322 -1328. DOI: [10.1109/IJCNN.2008.4633969](https://doi.org/10.1109/IJCNN.2008.4633969).
- [75] Kaiming He y col. *Deep Residual Learning for Image Recognition*. 2015. arXiv: [1512.03385 \[cs.CV\]](https://arxiv.org/abs/1512.03385).
- [76] Fuzhen Zhuang y col. *A Comprehensive Survey on Transfer Learning*. 2020. arXiv: [1911.02685 \[cs.LG\]](https://arxiv.org/abs/1911.02685).
- [77] *Imagenet*. Visitado el 2021-09-27. URL: <https://www.image-net.org/>.
- [78] Sergey Ioffe y Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: [1502.03167 \[cs.LG\]](https://arxiv.org/abs/1502.03167).
- [79] Pete Laker. *Blackboard Design Pattern: A Practical C# Example - Radar Defence System*. Visitado el 2021-09-16. URL: <https://social.technet.microsoft.com/wiki/contents/articles/13461.blackboard-design-pattern-a-practical-c-example-radar-defence-system.aspx>.
- [80] Omar Al-Debagy y Peter Martinek. *A Comparative Review of Microservices and Monolithic Architectures*. 2019. arXiv: [1905.07997 \[cs.SE\]](https://arxiv.org/abs/1905.07997).
- [81] *Docker Compose*. Visitado el 2021-09-05. URL: <https://docs.docker.com/compose/>.
- [82] *Docker Swarm*. Visitado el 2021-09-05. URL: <https://docs.docker.com/engine/swarm/>.
- [83] *Kubernetes*. Visitado el 2021-09-05. URL: <https://kubernetes.io/es/>.
- [84] InfluxData. *Introduction to InfluxData's InfluxDB and TICK stack*. Visitado el 2021-09-16. URL: <https://www.influxdata.com/blog/introduction-to-influxdatas-influxdb-and-tick-stack/>.
- [85] Elastic. ¿Qué es el ELK Stack? Visitado el 2021-09-16. URL: <https://www.elastic.co/es/what-is/elk-stack>.
- [86] *Scikit-learn, referencia de LogisticRegression*. Visitado el 2021-09-05. URL: [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html).
- [87] *Scikit-learn, referencia de QuantileTransformer*. Visitado el 2021-09-05. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html>.
- [88] *Scikit-learn, referencia de AdaBoostClassifier*. Visitado el 2021-09-05. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>.
- [89] *Scikit-learn, referencia de RandomForestClassifier*. Visitado el 2021-09-05. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.
- [90] *Scikit-learn, referencia de GradientBoostingClassifier*. Visitado el 2021-09-05. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>.
- [91] *Scikit-learn, referencia de KNeighborsClassifier*. Visitado el 2021-09-05. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>.
- [92] informativohoy.com.ar. *Una mirada sobre la pesquería del langostino y el mal manejo del recurso en los últimos años*. Visitado el 2021-09-14. URL: <https://informativohoy.com.ar/una-mirada-sobre-la-pesqueria-del-langostino-y-el-mal-manejo-del-recurso-en-los-ultimos-anos/>.
- [93] Intel. *Computer Vision Annotation Toolkit*. Visitado el 2021-09-05. URL: <https://cvat.org/>.

- 
- [94] Apache. *Apache Airflow*. Visitado el 2021-09-05. URL: <https://airflow.apache.org/>.
  - [95] Jonas Herskind Sejr, Peter Schneider-Kamp y Naeem Ayoub. «Surrogate Object Detection Explainer (SODEx) with YOLOv4 and LIME». En: *Machine Learning and Knowledge Extraction* 3.3 (2021), págs. 662-671. ISSN: 2504-4990. DOI: [10.3390/make3030033](https://doi.org/10.3390/make3030033). URL: <https://www.mdpi.com/2504-4990/3/3/33>.