DEPARTAMENTO DE
INGENIERÍA ELÉCTRICA
UNIVERSIDAD DE CONCEPCIÓN

Desarrollo de soluciones basadas en inteligencia artificial para el monitoreo de productividad y seguridad ocupacional en líneas de procesamiento de Salmones Camanchaca

POR

Boris Joaquín López Durán

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil en Telecomunicaciones.

Profesor Guía: Dr. Sebastián Eugenio Godoy Medel
Supervisor Empresa: Roberto Sanhueza
Comisión: Dr. Gabriel Saavedra M

Concepción,
Julio de 2026

A mis abuelos, que desde el cielo guían cada paso.

Resumen

La industria salmonera chilena es el segundo productor mundial de salmón atlántico, con exportaciones que superan los 5.000 millones de dólares anuales. Dentro del proceso de industrialización del salmón, la etapa de despinado manual representa uno de los eslabones de mayor intensidad de mano de obra. En plantas de procesamiento como la de Salmones Camanchaca en Tomé, múltiples operarios trabajan simultáneamente en líneas de despinado donde la medición de productividad se realizaba tradicionalmente mediante observación directa o registros manuales al final del turno. Esta metodología impide obtener información objetiva en tiempo real sobre el desempeño de cada puesto, los períodos de inactividad, el número exacto de filetes procesados y la ocurrencia de detenciones en la línea.

El presente trabajo propone un **sistema de monitoreo de productividad basado en visión por computador** que automatiza la medición de estos parámetros en tiempo real. El sistema implementa dos modelos YOLOv11 (detectores de salmones y operarios respectivamente) que trabajan en conjunto mediante análisis de Regiones de Interés (ROI) definidas por puesto de trabajo. La lógica de co-detección determina el estado de cada operario (Despinando, Espera o Puesto Vacío) combinando la presencia simultánea de operario y salmón en cada ROI.

Se entrenó el modelo YOLOv11L en Google Colab con GPU NVIDIA Tesla A100 utilizando un dataset de 5.754 imágenes de entrenamiento y 549 imágenes de validación, capturadas en condiciones reales de operación en la planta. El modelo alcanzó un desempeño excepcional: **mAP@0.5 = 0,927** (promedio ponderado de ambas clases), con el modelo de operarios obteniendo precisión de 0,934 y mAP@0.5 = 0,951, y el modelo de filetes precisión de 0,891 y mAP@0.5 = 0,903, superando ampliamente los objetivos de proyecto (objetivo: mAP $>$ 0,90). En inferencia, el sistema corre a 103

FPS en una GPU NVIDIA RTX 4050 Laptop (hardware modesto de producción), permitiendo monitoreo en tiempo real de 10 puestos simultáneos sin comprometer fluidez.

El sistema genera automáticamente, al término de cada turno, un reporte diario en formato HTML con 15+ indicadores clave de desempeño (KPIs): filetes procesados por puesto, tiempos de trabajo/espera/vacío, velocidades de entrada y salida de la línea, detenciones registradas y ranking de productividad. Esta información, antes inaccesible de forma automática y objetiva, transforma los datos del turno en inteligencia accionable para supervisores y planificadores.

Se validó el sistema en condiciones reales de producción: el turno del 9 de marzo de 2026 registró 5.098 filetes despinados, 10 puestos monitoreados simultáneamente, precisión promedio superior al 91 % en detecciones, identificación automática de anomalías operativas (puestos desocupados prolongadamente, patrones de espera anómalos) y generación de reporte completo sin intervención manual. El desempeño confirma que el sistema está listo para su despliegue en operación continua (*production-ready*).

Complementariamente, se desarrolló un **modelo YOLOv11 multiclase** entrenado sobre 10.197 imágenes para detectar nueve tipos de cofia (cofias de colores diferentes según rol de personal), alcanzando mAP@0.5 = 0.894 sobre 9 clases. Este modelo constituye la base tecnológica de un futuro verificador automático de equipos de protección personal (EPP) que permitirá alertar cuando algún trabajador no utilice la cofia reglamentaria o cuando personal no autorizado acceda a áreas restringidas. Los resultados demuestran que las tecnologías de visión por computador y aprendizaje profundo son viables y efectivas para automatizar el monitoreo de procesos industriales en entornos reales con recursos computacionales moderados. El trabajo contribuye tanto al mejoramiento de la eficiencia operativa de la planta como al fortalecimiento de sus capacidades de seguridad y trazabilidad de personal, elementos críticos para cumplir con estándares internacionales de inocuidad alimentaria como BRC, IFS y GlobalG.A.P.

Palabras clave: Visión por computador, YOLOv11, detección de objetos en tiempo real, monitoreo de productividad, seguridad ocupacional, equipos de protección personal, acuicultura, procesamiento de salmón.

Abstract

Productivity measurement and occupational safety control in salmon processing plants remain significant operational challenges. Traditional monitoring methods rely on manual observation and end-of-shift record-keeping, preventing objective, real-time assessment of individual workstation performance, equipment downtime, and compliance with personal protective equipment (PPE) protocols. This thesis presents the design, implementation, and validation of an integrated artificial intelligence-based monitoring system for salmon deheading lines.

The system comprises three main technological components: (1) a real-time productivity monitoring pipeline integrating two YOLOv11 object detection models operating in tandem—one for worker detection and one for fillet detection—combined with Region-of-Interest (ROI) analysis and co-detection logic to classify worker state (Active Deheading, Waiting, Empty Station), coupled with automatic calculation of operational metrics including throughput rates, temporal activity distributions, and downtime events; (2) a multi-class YOLOv11 model trained on 10,197 images for detection of nine color-coded worker cofias (hairnets), establishing the foundation for an automated PPE compliance verification system; and (3) an automatic report generation module that consolidates shift data into interactive HTML dashboards with productivity rankings, temporal activity distributions, and downtime logs.

The system was trained on datasets captured under real operational conditions at Salmones Camanchaca's processing facility in Tomé, Chile. The main detection pipeline achieved exceptional performance: mAP@0.5 = 0.927 (Precision 0.912, Recall 0.892) for simultaneous detection of workers and fillets, substantially exceeding project targets. The hairnet detection model achieved mAP@0.5 = 0.894 across nine color categories. Validation on a complete production shift (2026-03-09, 13:29–18:39) demonstrated

the system's practical viability: automatic monitoring of 10 workstations in parallel, processing 5,098 fillets, generation of 15+ operational key performance indicators in real time, identification of productivity bottlenecks, and detection of personnel occupancy anomalies. Detection precision exceeded 91 % across all workstations.

Implementation was conducted on modest computational hardware (NVIDIA RTX 4050 Laptop GPU, 12 GB VRAM), achieving 103 frames per second processing with 9.7 millisecond latency per frame—substantially exceeding the 30 FPS real-time requirement. The system generates daily shift reports automatically, eliminating manual data consolidation overhead and enabling data-driven decisions for personnel redistribution, workload balancing, and safety compliance verification.

This work demonstrates the viability of integrating multiple deep learning models in a single industrial production pipeline, providing concrete solutions to productivity measurement and occupational safety challenges in food processing environments. The system architecture and trained models are transferable to other salmon processing lines and conceptually applicable to other labor-intensive industrial processes with similar requirements for real-time activity recognition and object detection.

Keywords: Computer vision, YOLOv11, real-time object detection, productivity monitoring, aquaculture processing, occupational safety, personal protective equipment detection, deep learning.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas e instituciones que hicieron posible la realización de esta memoria de título.

En primer lugar, agradezco profundamente a mi profesor guía, **Dr. Sebastián Eugenio Godoy Medel**, por su orientación experta, retroalimentación constructiva y confianza en el desarrollo de este proyecto. Sus comentarios críticos y sugerencias fueron fundamentales para mejorar tanto la calidad técnica como la presentación de este trabajo.

A **Salmones Camanchaca S.A.**, por abrir las puertas de su planta de procesamiento en Tomé y permitirme desarrollar este proyecto en un entorno real de producción. Especialmente agradezco a **Roberto Sanhueza**, supervisor de mi práctica profesional en el área de TI, por su apoyo constante, disposición para colaborar y por facilitarme acceso a infraestructura, datos y equipamiento necesarios para la investigación. Su confianza en el potencial de estas soluciones tecnológicas fue crucial para llevar el proyecto a buen término.

A los supervisores y operarios de las líneas de despinado, quienes pacientemente colaboraron durante las sesiones de grabación de video y pruebas del sistema en condiciones reales. Sus observaciones y retroalimentación fueron cruciales para refinar el sistema y garantizar su utilidad práctica en el ambiente operativo de la planta.

A mi familia, por su apoyo incondicional durante estos años de formación universitaria. Especialmente agradezco a **Boris López**, **Marisol Durán** y **Renata López**, cuya comprensión, estímulo constante y confianza en mis capacidades fueron el soporte emocional necesario para superar los desafíos académicos. A mi pareja, **Daniela Curimil**, por su paciencia, apoyo permanente y por ser mi motivación durante los momentos más exigentes de este proceso. Sin el respaldo incondicional de todos ellos,

este logro no hubiera sido posible.

A la **Universidad de Concepción**, por proporcionar la formación académica rigurosa en Ingeniería Civil en Telecomunicaciones que fundamentó todas las competencias técnicas aplicadas en este trabajo.

Finalmente, agradezco a todos los colegas y compañeros que, de una u otra forma, contribuyeron con preguntas, discusiones y sugerencias que enriquecieron diferentes aspectos de este proyecto.

Boris Joaquín López Durán

Concepción, Región del Biobío

Julio de 2026

Índice General

Índice de Figuras

Índice de Tablas

Siglas

API Interfaz de Programación de Aplicaciones (Application Programming Interface)

BRC Consorcio Minorista Británico (British Retail Consortium)

CNN Red Neuronal Convolucional (Convolutional Neural Network)

COCO Objetos Comunes en Contexto (Common Objects in Context)

CSP Red de Etapas Cruzadas Parciales (Cross Stage Partial Network)

CSV Valores Separados por Comas (Comma-Separated Values)

CUDA Arquitectura Unificada de Dispositivos de Cómputo (Compute Unified Device Architecture)

EPP Elementos de Protección Personal

ERP Planificación de Recursos Empresariales (Enterprise Resource Planning)

FN Falso Negativo (False Negative)

FP Falso Positivo (False Positive)

FPS Fotogramas por Segundo (Frames Per Second)

GPU Unidad de Procesamiento Gráfico (Graphics Processing Unit)

HTML Lenguaje de Marcado de Hipertexto (HyperText Markup Language)

IFS Estándar Internacional de Alimentos (International Featured Standards)

ISO Organización Internacional de Normalización (International Organization for Standardization)

KPI Indicador Clave de Rendimiento (Key Performance Indicator)

LAN Red de Área Local (Local Area Network)

mAP Precisión Media Promedio (mean Average Precision)

MES Sistema de Ejecución de Manufactura (Manufacturing Execution System)

NLP Procesamiento de Lenguaje Natural (Natural Language Processing)

NMS Supresión No Máxima (Non-Maximum Suppression)

PPE Equipo de Protección Personal (Personal Protective Equipment)

PTZ Panorámica (Pan-Tilt-Zoom)

ROI Región de Interés (Region of Interest)

RTSP Protocolo de Transmisión en Tiempo Real (Real-Time Streaming Protocol)

SPPF Agrupamiento Piramidal Espacial Rápido (Spatial Pyramid Pooling Fast)

SSD Detector Multi-Caja de Disparo Único (Single Shot MultiBox Detector)

TP Verdadero Positivo (True Positive)

TSN Red de Segmentos Temporales (Temporal Segment Networks)

VOC Clases de Objetos Visuales (Visual Object Classes)

VRAM Memoria de Acceso Aleatorio de Video (Video Random Access Memory)

YOLO Solo Miras Una Vez (You Only Look Once)

Capítulo 1

Introducción

1.1. Trabajos previos

1.1.1. Industria salmonera chilena y desafíos del procesamiento

La industria salmonera chilena constituye uno de los pilares fundamentales de la economía exportadora del país. Chile se posiciona como el segundo productor mundial de salmón atlántico (*Salmo salar*) después de Noruega, concentrando aproximadamente el 31 % de la producción global de salmónidos [2, 3]. Durante el año 2024, las exportaciones chilenas de salmón y trucha alcanzaron un total de 782.076 toneladas con un valor superior a los 6.370 millones de dólares, representando cerca del 5,5 % del total de exportaciones del país y manteniéndose como el segundo producto de exportación más importante después del cobre [4].

A nivel histórico, la producción salmonera chilena ha experimentado un crecimiento sostenido desde la década de 1990, pasando de representar el 10 % de la producción mundial en 1990 a más del 30 % en años recientes [5]. Este crecimiento ha estado acompañado por una creciente sofisticación de los procesos productivos y por la implementación obligatoria de estándares internacionales de inocuidad alimentaria como BRC [6], IFS [7], ISO 22000 [8] y GLOBALG.A.P. [9]. La competitividad del sector depende en gran medida de la eficiencia de las líneas de procesamiento industrial, donde

pequeñas variaciones en el rendimiento de los operarios o la ocurrencia de detenciones no controladas pueden traducirse en pérdidas significativas de productividad a lo largo de un turno de trabajo [10].

Dentro del proceso de transformación industrial del salmón, la etapa de despinado manual representa uno de los eslabones de mayor intensidad de mano de obra. Múltiples operarios trabajan en paralelo extrayendo manualmente las espinas pin (*pin bones*) de los filetes antes de su clasificación, empaque y exportación. En la mayoría de las plantas del sector, la medición de la productividad en estas líneas se realiza mediante observación directa o registros manuales al término del turno, lo cual impide obtener información objetiva y en tiempo real sobre el estado de cada puesto de trabajo, la cantidad exacta de filetes procesados, los períodos de inactividad y la ocurrencia de detenciones de línea [10]. Esta limitación operativa representa una oportunidad clara para la incorporación de tecnologías de monitoreo automático basadas en visión por computador.

1.1.2. Visión por computador y aprendizaje profundo

La visión por computador moderna ha sido transformada por los avances en aprendizaje profundo (*deep learning*). El trabajo seminal de Krizhevsky et al. [11] demostró la superioridad de las redes neuronales convolucionales (CNN) sobre métodos tradicionales en el desafío ImageNet, marcando el inicio de una nueva era en visión por computador. Posteriormente, arquitecturas como ResNet [12] resolvieron el problema del desvanecimiento del gradiente en redes profundas mediante conexiones residuales, mientras que los fundamentos teóricos del aprendizaje profundo se consolidaron en obras de referencia como las de LeCun et al. [13] y Goodfellow et al. [14].

En el dominio específico de la detección de objetos, los enfoques iniciales se basaron en arquitecturas de dos etapas como R-CNN [15], Fast R-CNN [16] y Faster R-CNN [17], las cuales alcanzaron alta precisión a costa de velocidades de inferencia limitadas. La introducción de detectores de una sola etapa como SSD [18] y especialmente la familia YOLO (*You Only Look Once*) [19], revolucionó la disciplina al permitir detección en tiempo real con precisión competitiva. La familia YOLO ha experimentado una evolución continua a través de versiones sucesivas: YOLOv2 [20], YOLOv3 [21] y YOLOv4 [22], hasta llegar a YOLOv8 [23] y la más reciente YOLOv11 [24, 25], lanzada

en septiembre de 2024.

YOLOv11 introduce mejoras arquitectónicas significativas respecto a sus predecesoras, incluyendo el bloque C3K2 (*Cross Stage Partial Bottleneck con kernel 3×3*), el módulo SPPF (*Spatial Pyramid Pooling Fast*) optimizado y el mecanismo de atención C2PSA (*Cross Stage Partial with Spatial Attention*) [25]. Estas mejoras han permitido alcanzar mayor precisión con un 22 % menos de parámetros que YOLOv8, manteniendo la capacidad de inferencia en tiempo real [26]. Para la evaluación de modelos de detección se utilizan métricas estandarizadas como precisión, *recall* y *mean Average Precision* (mAP), derivadas de protocolos consolidados como PASCAL VOC [27] y COCO [28], cuyas formulaciones rigurosas han sido sintetizadas por Padilla et al. [29].

1.1.3. Aplicaciones de YOLO en monitoreo de productividad industrial

Los modelos de detección de objetos han demostrado ser herramientas efectivas para el monitoreo de productividad y reconocimiento de actividades en entornos industriales. Wang et al. [30] aplicaron una arquitectura combinada YOLOv3+VGG16 para el monitoreo automático de operaciones en talleres de manufactura bajo el paradigma de la Industria 4.0, demostrando la capacidad de detectar y analizar acciones de operarios en tiempo real. Soltanzadeh y Mohammadfam [31] desarrollaron un sistema basado en CNN, R-CNN y YOLOv3-Tiny para la identificación automática de actividades en procedimientos de servicio de calderas industriales, alcanzando precisiones competitivas en la clasificación de hasta 16 actividades industriales distintas.

En el contexto de líneas de producción, Tian et al. [32] demostraron la viabilidad de YOLOv5 para la detección en tiempo real de herramientas en entornos de manufactura inteligente, mientras que Khan e Iqbal [33] mostraron que las arquitecturas YOLO pueden ser empleadas exitosamente para reconocimiento y localización de acciones humanas en tiempo casi real. Estos trabajos evidencian la capacidad de los modelos YOLO para abordar tareas de monitoreo continuo en entornos productivos, con tasas de procesamiento adecuadas para aplicaciones en línea.

1.1.4. Detección de equipos de protección personal (EPP)

La verificación automática del uso de equipos de protección personal mediante visión por computador ha sido un área de investigación activa, particularmente en el sector de la construcción donde los riesgos laborales son elevados. Delhi et al. [34] desarrollaron uno de los primeros marcos de trabajo basados en YOLOv3 para detectar el cumplimiento de protocolos de EPP en obras de construcción. Posteriormente, Ferdous y Ahsan [35] presentaron un detector basado en YOLOX-m utilizando el dataset CHVG (cascos de cuatro colores, chaleco y gafas de seguridad), alcanzando un mAP@0.5 de 89,84 % sobre 8 clases.

Estudios más recientes muestran resultados aún más prometedores: Wang et al. [36] reportan mAP@0.5 superiores al 92 % utilizando YOLOv8x; Pham et al. [37] desplegaron una versión mejorada de YOLOv8 en dispositivos edge (Jetson Xavier NX), alcanzando un mAP@0.5 de 92,52 % a 9,11 FPS; Hayat et al. [38] reportan precisión del 97,64 % y *recall* del 93,11 % para clasificación de cumplimiento; Liu et al. [39] desarrollaron SD-YOLOv5s alcanzando AP=93,7 % mediante un mecanismo de atención DilateFormer; y Park et al. [40] aplicaron YOLOv10 con arquitecturas Transformer obteniendo AP50 promedio de 87,32 % para detección multi-clase. En el caso específico de YOLOv11, Karthik et al. [41] reportan mAP@0.5 de 88,5 % en sistemas integrados con modelos de lenguaje. Estos resultados confirman la viabilidad técnica y la madurez de los detectores basados en YOLO para aplicaciones de seguridad ocupacional automatizada.

1.1.5. Visión por computador en procesamiento de pescado y acuicultura

En el sector específico de procesamiento de pescado y acuicultura, los avances han sido más recientes pero significativos. Zhang et al. [42] presentaron un sistema de segmentación de partes de salmón y detección de defectos basado en una arquitectura híbrida U-Net+CBAM y YOLOv5, alcanzando un mAP de 96,87 % y mIoU de 94,33 % en tareas de corte. Zhang et al. [43] demostraron la viabilidad del conteo automático de poblaciones de salmón atlántico mediante visión por computador, alcanzando una precisión de 95,06 %. Por su parte, Einarsdóttir et al. [10] ofrecen una revisión integral

del estado actual de la automatización en la industria pesquera, identificando las oportunidades y limitaciones de la digitalización en este sector.

Aplicaciones más específicas incluyen los trabajos de Jang y Seo [44] sobre predicción de puntos de corte para alcanzar pesos objetivo, Jareño et al. [45] sobre clasificación automatizada en subastas de pescado, y Hamzaoui et al. [46] sobre reconocimiento de especies subacuáticas. Sin embargo, estos trabajos se concentran mayoritariamente en inspección de calidad del producto (segmentación, conteo, clasificación de especies, detección de defectos) más que en la medición de eficiencia operativa de las líneas de procesamiento o en la cuantificación del desempeño de los operarios. Esta brecha en la literatura representa una oportunidad concreta para el presente trabajo.

1.1.6. Análisis por regiones de interés (ROI) en sistemas de vigilancia

El análisis basado en regiones de interés (ROI, *Regions of Interest*) ha demostrado ser un enfoque efectivo para sistemas de monitoreo donde se requiere atribuir detecciones a zonas específicas del campo de visión. Islam et al. [47] desarrollaron un sistema de seguimiento multi-objeto consciente de ROI basado en YOLOv6 y SORT, alcanzando un mAP de 92,3 % para vigilancia inteligente y demostrando la utilidad de combinar detección con análisis zonal. Alvar y Bajić [48] propusieron MV-YOLO, una técnica de seguimiento asistido por vectores de movimiento que utiliza ROIs predichos para reducir la carga computacional. Más recientemente, Kumar et al. [49] reportan precisión de 96,78 % y *recall* de 96,89 % combinando YOLOv8 con mecanismos de atención y arquitecturas Transformer en aplicaciones de vigilancia en tiempo real.

1.1.7. Discusión y brecha tecnológica identificada

La revisión bibliográfica permite identificar tres conclusiones clave que justifican el presente trabajo:

Primero, los modelos YOLO han alcanzado niveles de madurez técnica que los hacen viables para aplicaciones industriales en tiempo real, con precisiones consistentemente superiores al 90 % en tareas de detección de objetos en dominios específicos [25, 26],

incluso utilizando hardware computacional moderado.

Segundo, las aplicaciones de detección de EPP en construcción están altamente desarrolladas [35, 36, 37, 39, 40], mientras que su adaptación a entornos de alimentos —particularmente para detección de cofias higiénicas en plantas de procesamiento— es prácticamente inexistente en la literatura académica revisada. Esto representa una oportunidad clara de extensión de las metodologías existentes a un nuevo dominio.

Tercero, las aplicaciones de visión por computador en procesamiento de pescado se concentran en inspección de calidad del producto [42, 44, 45], pero **no abordan la medición de productividad de los operarios ni el monitoreo en tiempo real del desempeño de las líneas de despinado**. Esta es la brecha tecnológica específica que el presente trabajo busca cubrir.

Adicionalmente, las arquitecturas de reconocimiento de acciones temporal-espaciales como SlowFast Networks [50], I3D [51] o Temporal Segment Networks [52], si bien son técnicamente capaces de clasificar estados de actividad en video, presentan costos computacionales elevados que limitan su despliegue en hardware industrial modesto. Esta restricción motivó la búsqueda de una solución alternativa basada en detección de objetos combinada con análisis de regiones de interés, capaz de inferir el estado de cada puesto de trabajo a partir de las detecciones espaciales en cada fotograma sin requerir el procesamiento de secuencias temporales completas.

1.2. Definición del problema

A partir de la revisión bibliográfica y del diagnóstico realizado en la planta de procesos de Salmones Camanchaca durante la práctica profesional previa al desarrollo de esta memoria, se identifican deficiencias concretas en el monitoreo operativo de las líneas de despinado de salmón que constituyen el problema a resolver:

1. **Ausencia de medición objetiva por puesto:** la cuantificación de filetes procesados por cada operario, así como la distribución temporal de su trabajo (tiempo activo, espera, puesto vacío), se realiza únicamente mediante observación directa del supervisor, lo cual es subjetivo, costoso en mano de obra y limitado en su precisión.

2. **Imposibilidad de identificación de cuellos de botella en tiempo real:** sin métricas instantáneas, los problemas operativos solo se detectan cuando ya han generado pérdidas acumuladas considerables.
3. **Falta de registro automático de detenciones de línea:** las paradas de la línea de procesamiento, ya sean en la entrada (sin ingreso de salmones), en la salida (sin egreso de filetes procesados), o detenciones generales (ambas zonas detenidas simultáneamente), no quedan registradas de manera sistemática, impidiendo análisis estadísticos posteriores sobre causas y patrones.
4. **Inexistencia de verificación automatizada de EPP:** el control del uso correcto de cofias por color (que en la planta funcionan también como código identificador del rol del personal) y de otros elementos de protección, depende exclusivamente de la inspección visual de supervisores.
5. **Carencia de reportes consolidados:** no existe una herramienta que sintetice automáticamente los datos de cada turno en información accionable para la toma de decisiones gerenciales sobre redistribución de personal, ajustes operativos o evaluación de desempeño.

A pesar de la disponibilidad de tecnologías de visión por computador maduras —particularmente la familia YOLO [25, 26]— y de su demostrada efectividad en múltiples sectores industriales [30, 32, 47], su adopción en plantas salmoneras chilenas para monitoreo de productividad y seguridad ocupacional permanece prácticamente inexplorada en la literatura académica.

Así, la presente memoria de título aborda el problema del monitoreo automático e integral de la productividad y seguridad en líneas de despinado de salmón mediante el desarrollo, despliegue y validación en producción de un sistema basado en visión por computador y aprendizaje profundo. La solución propuesta combina dos modelos YOLOv11 independientes (uno para detección de operarios y otro para detección de filetes), un módulo de análisis por regiones de interés (ROI) por puesto de trabajo.

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar e implementar un sistema integrado de monitoreo de productividad y seguridad ocupacional basado en visión por computador y aprendizaje profundo que automatice la medición de indicadores operativos en tiempo real en líneas de despinado de salmón, utilizando dos modelos YOLOv11 independientes y análisis por regiones de interés (ROI) para clasificar el estado de cada puesto de trabajo, registrar detenciones de línea, y generar reportes automáticos diarios.

1.3.2. Objetivos específicos

1. Diseñar e implementar un sistema de captura de video en tiempo real mediante cámaras IP con stream RTSP, instaladas en posición cenital sobre las líneas de despinado, con resolución y frecuencia de fotogramas adecuadas para operación continua en ambiente de producción.
2. Construir un dataset etiquetado de operarios y filetes a partir de videos capturados en condiciones reales de operación, abarcando múltiples puestos, variaciones de iluminación y diferentes velocidades de procesamiento.
3. Entrenar dos modelos YOLOv11 independientes —uno para detección de operarios y otro para detección de filetes de salmón— sobre el dataset construido, alcanzando un mAP@0.5 superior a 0,90 en el conjunto de validación.
4. Implementar una lógica de análisis por regiones de interés (ROI) para cada puesto de trabajo que combine las detecciones de operarios y filetes con el fin de clasificar automáticamente el estado de cada puesto en tres categorías: *Despinando*, *Espera* y *Puesto Vacío*.
5. Desarrollar un módulo de detección automática de detenciones de línea capaz de identificar y registrar cuatro tipos distintos de detención: (i) detenciones en la entrada de la línea, (ii) detenciones en la salida, (iii) detenciones generales (ambas simultáneas), y (iv) detenciones totales (suma de las anteriores).

6. Implementar un módulo de cálculo en tiempo real de los siguientes 15 indicadores operativos:
    1. Tiempo de presencia en cada puesto (con operario *vs.* vacío);
    2. Filetes despinados por puesto (vinculados a los filetes entrantes y al tiempo activo del operario);
    3. Tiempo de espera por puesto (en minutos);
    4. Distribución del tiempo de trabajo por puesto (minutos despinando *vs.* minutos en espera);
    5. Tiempo promedio de despinado por filete por puesto (relación entre filetes procesados y tiempo activo);
    6. Tiempo promedio de despinado por filete a nivel global (promedio sobre todos los puestos);
    7. Filetes procesados por minuto (relación entre filetes entrantes y salientes por unidad de tiempo);
    8. Productividad *vs.* tiempo efectivo de trabajo (gráfico de filetes despinados *vs.* minutos despinando);
    9. Total de detenciones de línea (umbral de 40 segundos sin movimiento);
    10. Total de detenciones generales (sin entrada ni salida de filetes);
    11. Total de detenciones de entrada (sin ingreso de salmones a la línea);
    12. Total de detenciones de salida (sin egreso de filetes procesados);
    13. Tiempo acumulado en detenciones generales (en horas y minutos);
    14. Tiempo acumulado en detenciones de salida (en horas y minutos);
    15. Tiempo acumulado total de detenciones (suma de detenciones generales, de entrada y de salida).
7. Implementar un generador automático de reportes diarios en formato HTML con visualizaciones interactivas (Chart.js) que consolide los indicadores calculados, incluyendo gráficos de productividad por puesto, ranking de desempeño,

distribución de estados, evolución temporal de velocidades y tabla de detenciones registradas.

8. Validar el sistema completo en condiciones reales de producción sobre una línea operativa de despinado en la planta de Salmones Camanchaca, Tomé, procesando un turno completo de trabajo y verificando una precisión de detección superior al 90 %.

1.4. Metodología

La metodología propuesta para alcanzar los objetivos específicos se organiza en cinco fases secuenciales que cubren desde la recolección de datos hasta la validación en producción.

1.4.1. Fase 1: Diseño del sistema y recolección de datos

1. **Análisis de requerimientos operativos.** Se realizan reuniones con supervisores y personal de la planta para identificar la disposición de los puestos de trabajo, los requerimientos de cobertura de cámara, las condiciones de iluminación, las velocidades esperadas de procesamiento y las métricas operativas de mayor relevancia.
2. **Diseño de la arquitectura de captura de video.** Se selecciona la ubicación de las cámaras IP que proporcionen cobertura completa de los 10 puestos de trabajo. Se definen los parámetros técnicos (resolución, frecuencia de fotogramas y requerimientos de ancho de banda) considerando las restricciones de hardware del sistema de inferencia.
3. **Instalación de infraestructura.** Se instalan las cámaras IP en posición cenital sobre las líneas de despinado. Se configura la infraestructura de red local para captura de *streams* RTSP sin dependencia de servicios externos.
4. **Recolección de videos de entrenamiento.** Se capturan sesiones de video de 30 a 60 minutos bajo múltiples condiciones operativas: variaciones de iluminación,

distintos operarios, diferentes velocidades de procesamiento e inclusión deliberada de anomalías (puestos vacíos, detenciones de línea).

5. **Etiquetado de datos.** Se utiliza la plataforma Roboflow [53] para etiquetar manualmente: (i) la ubicación de los operarios en cada fotograma, y (ii) la ubicación de los filetes de salmón. Se aplican técnicas de *data augmentation* (rotación, ajuste de brillo, ajuste de contraste, volteo horizontal) para incrementar la variabilidad del dataset y mejorar la robustez del modelo [14].

1.4.2. Fase 2: Entrenamiento de los modelos de detección

1. **Preparación del dataset.** Se divide el dataset etiquetado en conjuntos de entrenamiento (80 %), validación (10 %) y prueba (10 %), siguiendo las prácticas estándar reportadas en la literatura [28].

2. **Entrenamiento de los modelos YOLOv11 para operarios y filetes.** Se utiliza la infraestructura de Google Colab con GPU NVIDIA Tesla A100 / L4 (disponibles bajo licencia académica) y el framework PyTorch [54] junto con la implementación oficial de Ultralytics [24]. Se entrena durante un mínimo de 100 épocas con *early stopping* y se evalúan las métricas de mAP@0.5, precisión y *recall* en el conjunto de validación [29].

3. **Construcción del dataset de cofias.** Se capturan 10.197 imágenes de operarios en distintas posiciones, ángulos y condiciones de iluminación, cubriendo los nueve colores de cofia utilizados en la planta. Se etiqueta cada imagen con su clase correspondiente.

4. **Entrenamiento del modelo YOLOv11 multiclase de cofias.** Se entrena un modelo de detección con 9 clases. Se evalúa el desempeño mediante mAP@0.5 y matriz de confusión por clase para detectar posibles confusiones entre colores similares para ver la posibilidad de evaluar un modelo de prevención de riesgo.

5. **Análisis iterativo de resultados.** Se evalúan las métricas en el conjunto de prueba. Si no se alcanzan los objetivos de precisión, se itera el proceso: recolección de más datos, ajuste de hiperparámetros o exploración de variantes arquitectónicas (*nano*, *small*, *medium*, *large*).

1.4.3. Fase 3: Implementación del *pipeline* en tiempo real

1. **Integración de los modelos en el *pipeline* de inferencia.** Se implementa en Python utilizando PyTorch [54] y Ultralytics YOLOv11 [24]. Se capturan fotogramas desde las cámaras IP vía *stream* RTSP empleando la biblioteca OpenCV [55]. Cada fotograma es procesado por los dos modelos de detección de manera intercalada.
2. **Definición de las regiones de interés (ROI).** Para cada puesto de trabajo se define manualmente un polígono ROI que delimita su zona de procesamiento. Esta estrategia, validada en la literatura por Islam et al. [47], permite atribuir las detecciones a puestos específicos sin requerir reentrenamiento del modelo.
3. **Lógica de co-detección por puesto.** Para cada ROI se evalúa: (i) presencia o ausencia de operario, (ii) presencia o ausencia de filete, y (iii) clasificación del estado resultante. La combinación operario + filete dentro del ROI se interpreta como estado *Despinando*; operario sin filete como *Espera*; y ausencia de operario como *Puesto Vacío*.
4. **Módulo de cálculo de los 15 indicadores operativos.** Se implementa el cálculo en tiempo real de los indicadores definidos en el objetivo específico 6, incluyendo tiempos acumulados por puesto, filetes procesados, velocidades de entrada/salida y los cuatro tipos de detención de línea (general, entrada, salida y total).
5. **Visualización en tiempo real.** Se desarrolla una interfaz gráfica utilizando Gradio [56] que muestra el video procesado con detecciones superpuestas, el estado actual de cada puesto y un panel de métricas actualizado de forma continua.

1.4.4. Fase 4: Generación automática de reportes

1. **Consolidación de datos del turno.** Al término de cada turno, se consolidan los datos capturados (detecciones, métricas, eventos de detención) en archivos CSV con marca temporal.
2. **Generación de gráficos.** Se utilizan bibliotecas de visualización (Matplotlib, Chart.js) para generar los gráficos del reporte: filetes por puesto (gráfico

de barras), distribución temporal de estados (gráfico apilado), evolución de velocidades de entrada/salida (líneas) y matriz de actividad por operario.

3. **Construcción del reporte HTML interactivo.** Se genera una página HTML autónoma con gráficos interactivos, ranking de productividad, tabla de detenciones detectadas (con tipo y duración) y resumen de KPIs principales.
4. **Automatización del proceso.** Se programa el script para que se ejecute automáticamente al finalizar cada turno, generando el reporte sin intervención manual y dejándolo disponible en una ubicación accesible para los supervisores.

1.4.5. Fase 5: Validación en producción

1. **Despliegue en hardware de producción.** El *pipeline* es migrado a una GPU de costo modesto (NVIDIA RTX 4050 Laptop GPU, 12 GB VRAM), demostrando la viabilidad económica del despliegue.
2. **Prueba de operación continua.** Se ejecuta el sistema durante turnos completos de operación. Se registran posibles errores o caídas, la latencia de procesamiento y el consumo de recursos computacionales.
3. **Validación de precisión.** Se comparan automáticamente las detecciones del sistema contra conteos manuales realizados simultáneamente por un supervisor. Se calculan precisión, *recall* y matriz de confusión.
4. **Reporte de validación.** Se documentan los resultados de la validación, las limitaciones observadas en producción y las recomendaciones para mejoras futuras.

1.5. Alcances y limitaciones

1.5.1. Alcances

El presente trabajo cubre los siguientes elementos:

Monitoreo de productividad por puesto: medición automática de filetes procesados, velocidades de procesamiento y distribución temporal de la actividad (trabajo, espera e inactividad) para los 10 puestos de la línea.

Detección automática de detenciones de línea: identificación y registro de cuatro tipos de detención (entrada, salida, general y total) con sus respectivas duraciones acumuladas, utilizando un umbral temporal de 40 segundos sin movimiento de filetes como criterio.

Detección de EPP basada en cofias: entrenamiento de un detector multiclase de nueve colores de cofia (10.197 imágenes etiquetadas), como base para un futuro sistema verificador de cumplimiento de EPP en esta ocación será solo evaluacion de factibilad , no se evaluará a larga , ni escalabilidad.

Cálculo automático de 15 indicadores operativos: incluyendo tiempos por puesto, filetes despinados, tiempos de espera, productividad relativa, velocidades de procesamiento y métricas detalladas de detenciones (cantidades y tiempos acumulados por tipo).

Reportes automáticos diarios: generación de reportes HTML interactivos al finalizar cada turno, con gráficos, rankings y tablas accesibles para supervisores.

Despliegue en hardware modesto: demostración de viabilidad operativa en NVIDIA RTX 4050 Laptop GPU (12 GB VRAM), evitando inversiones elevadas en infraestructura de servidor.

Validación en producción: operación del sistema en condiciones reales en la planta de Salmones Camanchaca, Tomé, con turnos completos de trabajo.

Documentación técnica: entrega de guías técnicas, procedimientos de entrenamiento de modelos y manual de operación que aseguran la sostenibilidad del sistema.

1.5.2. Limitaciones

El presente trabajo no aborda los siguientes aspectos:

Hardware de producción permanente: el sistema fue desarrollado y validado en hardware de investigación. La selección final de GPU para despliegue permanente

queda pendiente del análisis de costo-beneficio que realizará la empresa.

Verificador automático de EPP completo: el modelo de detección de cofias es un prototipo. Su integración en un sistema completo de verificación con alertas y escaladas requiere trabajo adicional sobre flujos operativos, umbrales de confianza y pruebas de aceptación.

Escalamiento a múltiples líneas: el trabajo valida el sistema sobre una única línea de despinado de 10 puestos. La replicación en líneas adicionales requerirá ajuste de los polígonos ROI y, posiblemente, recolección de datos adicionales por línea.

Calidad del fileteado: el sistema detecta presencia de operario y filetes, pero no evalúa la calidad del despinado (espinas residuales, defectos de corte). Esta tarea requeriría modelos adicionales especializados.

Variaciones extremas de iluminación: el sistema fue entrenado bajo las condiciones de iluminación habituales de la planta. Cambios significativos en la infraestructura lumínica podrían requerir reentrenamiento parcial del modelo.

Confidencialidad y protección de datos: todas las imágenes de entrenamiento contienen registros de operarios reales. El almacenamiento, resguardo y cumplimiento de regulaciones de privacidad son responsabilidad de la empresa.

Integración con sistemas corporativos: el sistema actual genera reportes HTML autónomos. Su integración con sistemas ERP, MES o *dashboards* corporativos no está incluida en el alcance del presente trabajo.

1.6. Estructura del documento

El presente documento se organiza en cinco capítulos principales:

Capítulo 1 (Introducción): presenta el contexto, los antecedentes, la definición del problema, los objetivos, la metodología y los alcances y limitaciones del trabajo (capítulo actual).

Capítulo 2 (Marco Teórico): desarrolla los fundamentos teóricos requeridos: redes neuronales convolucionales, arquitectura de YOLOv11 y sus componentes (C3K2,

SPPF, C2PSA), conceptos de detección de objetos, métricas de evaluación (mAP, precisión, *recall*) y revisión de aplicaciones en monitoreo industrial.

Capítulo 3 (Desarrollo): describe en detalle la implementación del sistema: diseño de la captura de video, recolección y etiquetado de datasets, configuración del entrenamiento, arquitectura del *pipeline* de inferencia en tiempo real, lógica de análisis por ROI, módulo de detección de detenciones, cálculo de indicadores operativos e implementación del generador de reportes.

Capítulo 4 (Resultados): presenta las métricas de desempeño de los modelos entrenados, los resultados de la validación en producción, el análisis detallado de un turno completo de operación y ejemplos de los reportes automáticos generados.

Capítulo 5 (Conclusiones): sintetiza los logros del trabajo, sus contribuciones técnicas y operativas, las limitaciones identificadas y las recomendaciones para trabajo futuro.

Capítulo 2

Marco Teórico

2.1. Introducción

El presente capítulo presenta los fundamentos teóricos y conceptuales que sustentan el desarrollo del sistema propuesto. Se abordan los conceptos clave de visión por computador y aprendizaje profundo, las arquitecturas de detección de objetos relevantes —en particular YOLOv11—, los principios matemáticos detrás del cálculo de métricas de evaluación, las técnicas de análisis por regiones de interés (ROI), y la revisión del estado del arte en aplicaciones de monitoreo industrial. Finalmente, se justifica la elección metodológica adoptada para el desarrollo del sistema, contrastando alternativas y argumentando la pertinencia técnica de cada decisión.

2.2. Conceptos fundamentales

2.2.1. Visión por computador y aprendizaje profundo

La visión por computador es el área de la inteligencia artificial que dota a los sistemas computacionales de la capacidad de interpretar y comprender contenido visual digital, emulando funcionalmente el sistema visual humano [57, 58]. Sus aplicaciones abarcan desde la inspección industrial automatizada hasta la conducción autónoma, pasando por la imagenología médica y la seguridad pública.

El aprendizaje profundo (*deep learning*) es una rama del aprendizaje automático que utiliza redes neuronales con múltiples capas ocultas para aprender representaciones jerárquicas de los datos directamente a partir de ejemplos [13, 14]. A diferencia de los métodos clásicos de visión por computador, que requieren la definición manual de características relevantes (*handcrafted features*), las redes profundas aprenden estas representaciones de forma automática durante el entrenamiento, lo cual ha permitido alcanzar precisiones sin precedentes en tareas como clasificación de imágenes, detección de objetos y segmentación semántica [11].

2.2.2. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (*Convolutional Neural Networks*, CNN) constituyen la arquitectura fundamental sobre la cual se construyen los modernos sistemas de visión por computador [14]. Su diseño está inspirado en la organización de la corteza visual de los mamíferos y se basa en tres operaciones principales: convolución, agrupamiento (*pooling*) y activación no lineal.

2.2.2.1. Operación de convolución

La operación de convolución aplica un conjunto de filtros (*kernels*) sobre la imagen de entrada, produciendo mapas de características (*feature maps*) que destacan patrones locales como bordes, esquinas o texturas. Para una imagen de entrada I y un *kernel* K, la convolución 2D se define como:

$$(I * K)(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i + m, j + n) \cdot K(m, n) \tag{2.1}$$

donde $M \times N$ son las dimensiones del *kernel* y (i, j) son las coordenadas espaciales en el mapa de salida. La capacidad de las CNN para aprender filtros relevantes a partir de los datos es lo que les confiere su poder representacional.

2.2.2.2. Función de activación y *pooling*

Tras cada capa convolucional se aplica una función de activación no lineal, típicamente la ReLU (*Rectified Linear Unit*):

$$\text{ReLU}(x) = \text{máx}(0, x) \tag{2.2}$$

Posteriormente, las operaciones de *pooling* (típicamente *max pooling*) reducen la resolución espacial de los mapas de características, proporcionando invariancia a pequeñas traslaciones y reduciendo la carga computacional [14].

2.2.2.3. Arquitecturas profundas y conexiones residuales

Las arquitecturas profundas modernas, como ResNet [12], introdujeron las conexiones residuales (*skip connections*) para resolver el problema del desvanecimiento del gradiente en redes con muchas capas. La función residual se formula como:

$$y = F(x, \{W_i\}) + x \tag{2.3}$$

donde $F(x, \{W_i\})$ representa la transformación aprendida y x es la entrada que se propaga directamente a la salida. Esta innovación arquitectónica permite entrenar redes con cientos de capas y ha sido adoptada por la mayoría de las arquitecturas modernas de detección de objetos.

2.2.3. Detección de objetos

La detección de objetos es una tarea de visión por computador que combina dos subtareas: **localización** (determinar la posición espacial de objetos en la imagen mediante *bounding boxes*) y **clasificación** (asignar una etiqueta de clase a cada objeto detectado) [57]. A diferencia de la clasificación de imágenes, donde se asigna una única etiqueta a la imagen completa, la detección permite identificar múltiples objetos de distintas clases dentro de una misma escena.

Los enfoques modernos de detección se dividen en dos grandes categorías:

- **Detectores de dos etapas:** primero generan propuestas de regiones candidatas y luego clasifican cada una. Ejemplos representativos son R-CNN [15], Fast R-CNN [16] y Faster R-CNN [17]. Estos métodos típicamente alcanzan alta precisión a costa de mayor latencia.
- **Detectores de una etapa:** predicen simultáneamente las cajas delimitadoras y las clases en una única pasada por la red. Ejemplos representativos son SSD [18] y la familia YOLO [19]. Estos métodos están optimizados para inferencia en tiempo real.

Para aplicaciones industriales con requerimientos de baja latencia —como el monitoreo continuo de líneas de producción— los detectores de una etapa son la elección preferida [26].

2.2.4. Bounding boxes y formato de etiquetado

Una *bounding box* (caja delimitadora) es un rectángulo axis-aligned que encierra un objeto detectado en la imagen. Existen distintos formatos de representación; el más utilizado en la familia YOLO es el formato normalizado:

$$\mathbf{b} = (x_c, y_c, w, h) \tag{2.4}$$

donde (x_c, y_c) son las coordenadas del centro de la caja y (w, h) su ancho y alto, todos normalizados al rango $[0, 1]$ respecto a las dimensiones de la imagen. Este formato facilita la generalización del modelo a imágenes de distintas resoluciones.

2.2.5. Regiones de interés (ROI)

Una región de interés (*Region of Interest*, ROI) es un subconjunto delimitado del campo de visión sobre el cual se concentra el análisis. En el contexto de sistemas de visión por computador para monitoreo industrial, las ROIs permiten asignar las detecciones a zonas específicas (por ejemplo, puestos de trabajo individuales) sin necesidad de reentrenar el modelo [47, 48]. Esta estrategia presenta tres ventajas operativas claves:

1. **Desacoplamiento entre detección y atribución espacial:** el modelo de detección genérico identifica objetos en toda la imagen, mientras que la lógica posterior atribuye cada detección a la ROI correspondiente.
2. **Flexibilidad ante cambios de configuración:** si la disposición de los puestos cambia, basta con redefinir los polígonos ROI sin afectar al modelo entrenado.
3. **Reducción de carga computacional:** al filtrar detecciones por ROI, se simplifica el análisis posterior y se acelera el procesamiento global del *pipeline*.

Formalmente, una ROI poligonal se define como un conjunto ordenado de vértices $V = \{v_1, v_2, \ldots, v_n\}$ donde cada $v_i = (x_i, y_i) \in \mathbb{R}^2$. La pertenencia de un punto $p = (x_p, y_p)$ a la ROI se determina mediante el algoritmo *point-in-polygon* (típicamente *ray casting*).

2.3. Arquitectura YOLO y su evolución

2.3.1. Origen y filosofía YOLO

La familia YOLO (*You Only Look Once*) fue introducida por Redmon et al. [19] en 2016 con la idea revolucionaria de plantear la detección de objetos como un único problema de regresión, en lugar de la cascada de subproblemas que utilizaban los detectores de dos etapas. La red procesa la imagen completa en una sola pasada, dividiendo la imagen en una cuadrícula de $S \times S$ celdas. Cada celda predice B *bounding boxes* con sus respectivas confianzas y probabilidades de clase. El tensor de salida tiene dimensiones:

$$S \times S \times (B \cdot 5 + C) \tag{2.5}$$

donde C es el número de clases y los 5 valores por caja corresponden a $(x_c, y_c, w, h, \text{conf})$.

Esta formulación unificada permitió alcanzar velocidades de inferencia órdenes de magnitud superiores a Faster R-CNN [17], manteniendo precisiones competitivas.

2.3.2. Evolución hasta YOLOv11

A lo largo de sus versiones sucesivas, la familia YOLO ha incorporado innovaciones técnicas que han mejorado tanto la precisión como la eficiencia:

- **YOLOv2** [20]: introduce *anchor boxes*, *batch normalization* [12] y entrenamiento multi-escala.
- **YOLOv3** [21]: incorpora detección a múltiples escalas mediante una arquitectura tipo *Feature Pyramid Network*, mejorando la detección de objetos pequeños.
- **YOLOv4** [22]: integra técnicas modernas como CSPDarknet53 como *backbone*, mosaic *augmentation*, función de pérdida CIoU y Mish como activación.
- **YOLOv5–YOLOv8** [23]: desarrollados por Ultralytics, introducen *anchor-free detection*, mejoras en el *pipeline* de entrenamiento, y una API unificada en PyTorch [54].
- **YOLOv11** [24, 25]: lanzado en septiembre de 2024, introduce los bloques C3K2, el módulo SPPF optimizado y el mecanismo de atención C2PSA.

La Tabla 2.1 resume las principales contribuciones de cada versión.

Tabla 2.1: Evolución de la familia YOLO y principales innovaciones técnicas.

Versión	**Año**	**Innovaciones principales**
YOLOv1	2016	Detección unificada en una sola pasada
YOLOv2	2017	*Anchor boxes*, *batch normalization*
YOLOv3	2018	*Multi-scale detection*, *Darknet-53*
YOLOv4	2020	CSPDarknet53, mosaic, CIoU loss
YOLOv5	2020	PyTorch nativo, *auto-anchors*
YOLOv8	2023	*Anchor-free*, *decoupled head*
YOLOv11	2024	C3K2, SPPF mejorado, C2PSA

2.3.3. Arquitectura de YOLOv11

YOLOv11 sigue la estructura clásica de tres componentes: *backbone*, *neck* y *head*, pero incorpora innovaciones específicas en cada uno [25].

2.3.3.1. Backbone: bloques C3K2

El *backbone* es responsable de la extracción de características jerárquicas a partir de la imagen de entrada. YOLOv11 utiliza bloques C3K2 (*Cross Stage Partial Bottleneck con kernel 3×3*), una evolución de los bloques C2f de YOLOv8. Los bloques C3K2 utilizan kernels más pequeños distribuidos eficientemente, permitiendo capturar patrones espaciales con menor costo computacional. La estructura CSP (*Cross Stage Partial*) divide el flujo de información en dos ramas que se reagrupan posteriormente, mejorando el flujo de gradiente y reduciendo el número de parámetros [25].

2.3.3.2. Neck: módulo SPPF

El *neck* agrega características de múltiples escalas para mejorar la detección de objetos de distintos tamaños. YOLOv11 emplea el módulo SPPF (*Spatial Pyramid Pooling Fast*), una versión optimizada del SPP clásico que aplica *max pooling* a múltiples escalas de manera secuencial en lugar de paralela, reduciendo la carga computacional sin sacrificar la representación multi-escala.

2.3.3.3. Head: mecanismo de atención C2PSA

El componente más distintivo de YOLOv11 es el módulo C2PSA (*Cross Stage Partial with Spatial Attention*), un mecanismo de atención espacial que permite al modelo enfocarse en las regiones más relevantes de la imagen. Este módulo combina los beneficios de las arquitecturas CSP con mecanismos de atención inspirados en *transformers*, mejorando especialmente la detección de objetos pequeños o parcialmente ocluidos [25].

2.3.4. Variantes y capacidades de YOLOv11

YOLOv11 está disponible en cinco variantes según su capacidad computacional, permitiendo adecuar el modelo al hardware disponible:

- **YOLOv11n** (*nano*): 2.6M parámetros — ideal para dispositivos *edge* de bajo consumo.
- **YOLOv11s** (*small*): 9.4M parámetros — equilibrio entre velocidad y precisión.
- **YOLOv11m** (*medium*): 20.1M parámetros — uso general en GPUs de consumo.
- **YOLOv11l** (*large*): 25.3M parámetros — alta precisión en hardware moderado.
- **YOLOv11x** (*extra-large*): 56.9M parámetros — máxima precisión en GPUs profesionales.

YOLOv11 alcanza la misma precisión que YOLOv8 con un 22 % menos de parámetros [26], lo cual representa una ventaja significativa para despliegues con restricciones de memoria o energía.

2.3.5. Función de pérdida en YOLOv11

YOLOv11 utiliza una función de pérdida compuesta que combina tres términos: pérdida de localización, pérdida de objetividad (*objectness*) y pérdida de clasificación:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{box}}\mathcal{L}_{\text{box}} + \lambda_{\text{obj}}\mathcal{L}_{\text{obj}} + \lambda_{\text{cls}}\mathcal{L}_{\text{cls}} \tag{2.6}$$

donde $\lambda_{\text{box}}, \lambda_{\text{obj}}, \lambda_{\text{cls}}$ son pesos que balancean las contribuciones de cada término. Para la pérdida de localización se utiliza CIoU (*Complete IoU*), que considera no solo el solapamiento entre cajas sino también el aspecto y la distancia entre centros:

$$\text{CIoU} = \text{IoU} - \frac{\rho^2(\mathbf{b}, \mathbf{b}^{\text{gt}})}{c^2} - \alpha \cdot v \tag{2.7}$$

donde $\rho^2(\cdot,\cdot)$ es la distancia euclidiana entre los centros, c es la diagonal de la caja envolvente mínima, v mide la consistencia de aspecto y α es un parámetro de balance [22].

2.4. Métricas de evaluación

La evaluación rigurosa del desempeño de un modelo de detección es esencial para validar su aplicabilidad en producción. Las métricas estándar utilizadas en este trabajo, derivadas de los protocolos PASCAL VOC [27] y COCO [28], se describen a continuación, siguiendo las formulaciones rigurosas presentadas por Padilla et al. [29].

2.4.1. Intersection over Union (IoU)

La métrica fundamental para evaluar la calidad de localización es el *Intersection over Union* (IoU), también conocida como índice de Jaccard:

$$\text{IoU} = \frac{|\mathbf{b}_{\text{pred}} \cap \mathbf{b}_{\text{gt}}|}{|\mathbf{b}_{\text{pred}} \cup \mathbf{b}_{\text{gt}}|} \tag{2.8}$$

donde $\mathbf{b}_{\text{pred}}$ es la caja predicha y $\mathbf{b}_{\text{gt}}$ la caja *ground truth* (verdad de referencia). El IoU varía entre 0 (sin solapamiento) y 1 (coincidencia perfecta). Una detección se considera positiva (TP, *True Positive*) si su IoU con alguna caja *ground truth* supera un umbral predefinido (típicamente 0.5).

2.4.2. Precisión y *recall*

A partir de la clasificación de detecciones según el umbral de IoU, se definen:

$$\text{Precisión} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.9}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.10}$$

donde TP, FP y FN denotan respectivamente *True Positives*, *False Positives* y *False Negatives*. La precisión mide qué proporción de las detecciones del modelo son correctas, mientras que el *recall* mide qué proporción de los objetos reales fueron detectados.

2.4.3. F1-Score

El F1-Score es la media armónica entre precisión y *recall*, proporcionando una única métrica balanceada:

$$\text{F1} = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}} \tag{2.11}$$

2.4.4. Average Precision (AP) y mean Average Precision (mAP)

El *Average Precision* (AP) se calcula como el área bajo la curva precisión-recall:

$$\text{AP} = \int_0^1 P(r)\, dr \tag{2.12}$$

donde $P(r)$ es la precisión en función del *recall*. En la práctica, se calcula mediante interpolación discreta sobre los valores de *recall*.

El *mean Average Precision* (mAP) promedia el AP sobre todas las clases:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{2.13}$$

donde N es el número de clases. Existen dos variantes principales:

- **mAP@0.5**: utiliza un único umbral de IoU = 0.5. Es la métrica más utilizada en aplicaciones industriales y la que se reporta con mayor frecuencia en la literatura [36, 35, 39].
- **mAP@0.5:0.95**: promedia el mAP sobre umbrales de IoU desde 0.5 hasta 0.95 con incrementos de 0.05. Es la métrica oficial del benchmark COCO [28] y proporciona una evaluación más rigurosa de la calidad de localización.

2.5. Estado del arte en monitoreo industrial

2.5.1. Detección de equipos de protección personal (EPP)

La verificación automática del uso de EPP mediante visión por computador ha sido un área de investigación activa, particularmente en el sector de la construcción. Los trabajos pioneros de Delhi et al. [34] aplicaron YOLOv3 para detectar el cumplimiento de protocolos de cascos y chalecos, sentando las bases metodológicas que serían refinadas en años posteriores.

Trabajos subsecuentes han explorado distintas variantes y mejoras: Ferdous y Ahsan [35] alcanzaron un mAP@0.5 de 89,84 % utilizando YOLOX-m sobre el dataset CHVG (cascos de cuatro colores, chaleco y gafas, 8 clases). Wang et al. [36] reportaron mAP@0.5 superiores al 92 % con YOLOv8x. Pham et al. [37] desplegaron YOLOv8 mejorado en dispositivos *edge* (Jetson Xavier NX) alcanzando 92,52 % de mAP@0.5 a 9,11 FPS, demostrando la viabilidad del despliegue en hardware modesto. Liu et al. [39] desarrollaron SD-YOLOv5s con un mecanismo de atención DilateFormer alcanzando AP=93,7 %. Park et al. [40] lograron un AP50 promedio de 87,32 % para detección multi-clase utilizando YOLOv10 con arquitecturas *Transformer*.

En el caso específico de YOLOv11, Karthik et al. [41] reportaron un mAP@0.5 de 88,5 % en sistemas integrados con modelos de lenguaje y razonamiento contextual. La Tabla 2.2 resume estos resultados.

Tabla 2.2: Comparación de trabajos previos sobre detección de EPP basada en YOLO.

Trabajo	**Modelo**	**Clases**	**mAP@0.5**	**Observación**
Delhi et al. [34]	YOLOv3	4	—	EPP construcción
Ferdous et al. [35]	YOLOX-m	8	0.898	Dataset CHVG
Hayat et al. [38]	YOLO-PPE	4	—	P=97.6 %, R=93.1 %
Wang et al. [36]	YOLOv8x	8	0.92	PPE construcción
Pham et al. [37]	YOLOv8+	5	0.925	9.11 FPS Jetson
Liu et al. [39]	SD-YOLOv5s	—	0.937	DilateFormer
Park et al. [40]	YOLOv10+ViT	5	0.873	Transformer
Karthik et al. [41]	YOLOv11s	—	0.885	+ NLP

2.5.2. Visión por computador en procesamiento de pescado

Las aplicaciones de visión por computador en procesamiento de pescado se han concentrado mayoritariamente en inspección de calidad del producto. Zhang et al. [42] presentaron un sistema de segmentación de partes de salmón y detección de defectos basado en una arquitectura híbrida U-Net+CBAM y YOLOv5, alcanzando un mAP de 96,87 % y mIoU de 94,33 % en tareas de corte. Zhang et al. [43] demostraron la viabilidad del conteo automático de poblaciones de salmón atlántico con una precisión de 95,06 %. Jang y Seo [44] predijeron puntos de corte óptimos para alcanzar pesos objetivo, mientras que Jareño et al. [45] clasificaron especies y calibres de pescado en subastas. Hamzaoui et al. [46] aplicaron deep learning a reconocimiento de especies subacuáticas en acuicultura.

Einarsdóttir et al. [10] ofrecen una revisión integral del estado actual de la automatización en la industria pesquera, identificando que las mayores oportunidades se encuentran en el monitoreo continuo de procesos productivos y en la verificación automática del cumplimiento de protocolos operativos —aspectos que el presente trabajo aborda directamente.

2.5.3. Monitoreo de productividad y reconocimiento de actividades

Los sistemas de monitoreo de productividad basados en visión por computador han sido explorados en diversos contextos industriales. Wang et al. [30] aplicaron una arquitectura combinada YOLOv3+VGG16 para el monitoreo automático de operaciones en talleres bajo el paradigma de Industria 4.0. Soltanzadeh et al. [31] desarrollaron un sistema basado en CNN, R-CNN y YOLOv3-Tiny para identificación automática de hasta 16 actividades industriales distintas. Tian et al. [32] demostraron la viabilidad de YOLOv5 para detección en tiempo real de herramientas en manufactura inteligente. Khan e Iqbal [33] mostraron que las arquitecturas YOLO pueden emplearse exitosamente para reconocimiento y localización de acciones humanas en tiempo casi real.

2.5.4. Análisis basado en regiones de interés (ROI)

El análisis basado en ROI ha demostrado ser efectivo para sistemas de monitoreo zonal. Islam et al. [47] desarrollaron un sistema de seguimiento multi-objeto consciente de ROI basado en YOLOv6 y SORT, alcanzando un mAP de 92,3 % para vigilancia inteligente. Alvar y Bajić [48] propusieron MV-YOLO, una técnica de seguimiento asistido por vectores de movimiento que utiliza ROIs predichos para reducir la carga computacional. Kumar et al. [49] reportan precisión de 96,78 % y *recall* de 96,89 % combinando YOLOv8 con mecanismos de atención y arquitecturas *Transformer*.

2.5.5. Reconocimiento de acciones temporales

Las arquitecturas especializadas en reconocimiento de acciones, capaces de procesar información temporal en secuencias de video, fueron consideradas como alternativa metodológica al enfoque basado en detección. SlowFast Networks [50], propuesta por Feichtenhofer et al. en 2019, utiliza dos ramas paralelas: una rama *Slow* que procesa información espacial detallada a baja frecuencia temporal, y una rama *Fast* que captura dinámica temporal a alta frecuencia. Otras arquitecturas como I3D [51] y *Temporal Segment Networks* (TSN) [52] también han sido evaluadas en contextos similares.

Si bien estas arquitecturas son técnicamente capaces de clasificar estados de actividad directamente sobre secuencias de video, presentan costos computacionales significativamente mayores que los detectores de una etapa, lo cual limita su despliegue en hardware industrial modesto.

2.6. Metodologías relevantes

2.6.1. Entrenamiento de modelos de aprendizaje profundo

El entrenamiento de modelos profundos sigue un proceso iterativo de optimización por gradiente. Para un dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ con N ejemplos etiquetados, el objetivo es encontrar los parámetros θ^* que minimizan la función de pérdida promedio:

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \tag{2.14}$$

donde f_θ es el modelo parametrizado por θ y $\mathcal{L}$ la función de pérdida. La optimización se realiza típicamente mediante variantes del descenso por gradiente estocástico (SGD), incluyendo Adam y AdamW.

2.6.2. Aumentación de datos (*data augmentation*)

La aumentación de datos consiste en aplicar transformaciones a las imágenes del conjunto de entrenamiento para incrementar artificialmente su variabilidad y mejorar la capacidad de generalización del modelo [14]. Las técnicas más utilizadas en YOLOv11 incluyen:

- **Transformaciones geométricas:** rotaciones, traslaciones, escalados, volteos horizontales.
- **Transformaciones fotométricas:** ajustes de brillo, contraste, saturación, tono.
- **Mosaic augmentation** [22]: combinación de cuatro imágenes en una sola, permitiendo al modelo aprender contextos diversos en cada iteración.
- **MixUp:** interpolación lineal entre dos imágenes y sus etiquetas.

Estas técnicas son particularmente importantes en aplicaciones industriales donde la recolección de datos etiquetados es costosa.

2.6.3. División del dataset y validación cruzada

La metodología estándar divide el dataset en tres conjuntos disjuntos: entrenamiento (típicamente 70-80 %), validación (10-15 %) y prueba (10-15 %). El conjunto de entrenamiento se utiliza para ajustar los parámetros, el de validación para seleccionar hiperparámetros y aplicar *early stopping*, y el de prueba para evaluar el desempeño final del modelo [14].

2.6.4. Etiquetado y plataformas de anotación

La calidad del etiquetado es uno de los factores determinantes en el desempeño final del modelo. En el presente trabajo se utilizó la plataforma Roboflow [53], una herramienta especializada en gestión de *datasets* de visión por computador que ofrece interfaces de anotación, control de versiones, exportación a múltiples formatos (incluyendo el formato YOLOv11) y aplicación automática de aumentaciones.

2.6.5. Frameworks de implementación

El desarrollo del sistema utiliza un *stack* tecnológico basado en software libre y de código abierto:

- **PyTorch** [54]: framework de aprendizaje profundo desarrollado por Meta AI, ampliamente adoptado por su flexibilidad y ecosistema maduro.
- **Ultralytics** [24]: implementación oficial y mantenida de la familia YOLO, con API de alto nivel para entrenamiento e inferencia.
- **OpenCV** [55]: biblioteca de visión por computador para captura, preprocesamiento y manipulación de imágenes y video.
- **Gradio** [56]: framework para construcción rápida de interfaces web para modelos de aprendizaje automático.
- **NVIDIA CUDA** [59]: plataforma de cómputo paralelo que permite acelerar el entrenamiento e inferencia mediante GPUs.

2.7. Procesamiento de video en tiempo real

2.7.1. Captura y transmisión de video mediante RTSP

El protocolo RTSP (*Real-Time Streaming Protocol*) es un estándar de la capa de aplicación diseñado para controlar la entrega de datos multimedia en tiempo real entre servidores y clientes [60]. En el contexto del sistema desarrollado, las cámaras

IP instaladas sobre las líneas de despinado transmiten sus *streams* de video mediante RTSP hacia el servidor de inferencia a través de la red local de la planta. La captura de fotogramas desde el *stream* RTSP se realiza con OpenCV [55] mediante la clase `VideoCapture`, que abstrae el protocolo de transmisión y proporciona acceso uniforme a los fotogramas individuales.

Las ventajas del uso de RTSP en este contexto son:

- **Transmisión en tiempo real con baja latencia:** adecuada para aplicaciones de monitoreo continuo.
- **Independencia de infraestructura externa:** opera completamente en la red local de la planta, sin dependencias de servicios *cloud*.
- **Compatibilidad universal:** prácticamente todas las cámaras IP industriales y de seguridad ofrecen soporte nativo RTSP.

2.7.2. Non-Maximum Suppression (NMS)

Un desafío inherente a los detectores de objetos es la generación de múltiples detecciones superpuestas para un mismo objeto. La técnica de *Non-Maximum Suppression* (NMS) resuelve este problema filtrando las cajas redundantes [19]. El algoritmo opera de la siguiente manera:

1. Ordenar todas las cajas detectadas por su puntaje de confianza en orden descendente.
2. Seleccionar la caja de mayor confianza y eliminar todas las cajas cuyo IoU con ella supere un umbral τ_{NMS} (típicamente 0.45).
3. Repetir el proceso con las cajas restantes hasta no quedar más cajas.

Formalmente, una caja $\mathbf{b}_j$ es suprimida si:

$$\text{IoU}(\mathbf{b}_i, \mathbf{b}_j) > \tau_{\text{NMS}} \quad \text{y} \quad \text{conf}(\mathbf{b}_i) > \text{conf}(\mathbf{b}_j) \tag{2.15}$$

donde $\mathbf{b}_i$ es la caja de mayor confianza. El resultado es un conjunto no redundante de detecciones que representan los objetos presentes en la imagen.

2.7.3. Umbral de confianza

Adicionalmente al NMS, se aplica un umbral de confianza τ_{conf} que filtra las detecciones cuya confianza no alcanza el nivel mínimo requerido. La confianza de una detección es el producto de la probabilidad de que la caja contenga un objeto y la probabilidad de clase condicional:

$$\text{conf}_{\text{final}} = P(\text{objeto}) \times P(\text{clase} \mid \text{objeto}) \tag{2.16}$$

La elección del umbral de confianza implica un compromiso entre precisión y *recall*: valores altos de τ_{conf} reducen los falsos positivos pero pueden aumentar los falsos negativos. En el sistema desarrollado, este umbral fue ajustado empíricamente durante la validación en producción.

2.7.4. Pipeline de inferencia intercalado

Dado que el sistema ejecuta dos modelos YOLOv11 independientes sobre el mismo *stream* de video, se implementó una arquitectura de *pipeline* intercalado (*interleaved inference*) para maximizar el uso de la GPU. En lugar de ejecutar ambos modelos secuencialmente sobre cada fotograma, el sistema alterna las inferencias de manera que el tiempo de espera de un modelo se aprovecha para la inferencia del otro, manteniendo una tasa de actualización efectiva de 30 FPS sobre el *display* de monitoreo.

2.8. Indicadores operativos y lógica de detención

2.8.1. Clasificación de estados por puesto

El sistema clasifica el estado de cada puesto de trabajo en tres categorías mutuamente excluyentes, determinadas por la combinación de detecciones dentro de la ROI correspondiente en cada fotograma:

- **DESPINANDO:** se detecta presencia de operario **y** presencia de al menos un filete dentro de la ROI. Indica que el operario está activamente procesando

material.

- **ESPERA:** se detecta presencia de operario pero **no** hay filetes dentro de la ROI. Indica que el operario está presente pero sin material para procesar, lo cual puede deberse a discontinuidades en el suministro de la línea.
- **PUESTO VACÍO:** no se detecta presencia de operario en la ROI, independientemente de si hay filetes o no. Indica que el puesto está desocupado.

Esta lógica determinística, fundamentada en la co-detección espacial dentro de ROIs definidas, permite inferir el estado productivo de cada puesto sin requerir el análisis de secuencias temporales [47].

2.8.2. Cálculo de los indicadores operativos

A partir de la clasificación de estados en tiempo real, el sistema acumula los siguientes indicadores operativos a lo largo del turno:

2.8.2.1. Indicadores de tiempo por puesto

Para cada puesto $p \in \{P01, P02, \ldots, P10\}$, el sistema mantiene contadores acumulados de tiempo en cada estado:

$$T_{\text{estado}}^{(p)} = \sum_{t=1}^{T} \mathbb{1}[\text{estado}(p,t) = \text{estado}] \cdot \Delta t \tag{2.17}$$

donde Δt es el intervalo de muestreo (inverso de la frecuencia de actualización), T es el número total de muestras del turno, y $\mathbb{1}[\cdot]$ es la función indicadora.

2.8.2.2. Conteo de filetes y velocidades

El conteo de filetes procesados por puesto se deriva de las detecciones de filetes asociadas a cada ROI durante períodos de estado DESPINANDO. Las velocidades de procesamiento se calculan como tasas instantáneas suavizadas exponencialmente:

$$v_{\text{inst}}(t) = \frac{\Delta F(t)}{\Delta t} \tag{2.18}$$

donde $\Delta F(t)$ es el incremento en el conteo de filetes en el intervalo Δt. La velocidad promedio del turno se calcula como:

$$\bar{v} = \frac{F_{\text{total}}}{T_{\text{activo}}} \tag{2.19}$$

donde F_{total} es el total de filetes procesados y T_{activo} el tiempo total en estado DESPINANDO.

2.8.2.3. Tiempo promedio por filete

El tiempo promedio de despinado por filete para cada puesto p se calcula como:

$$\bar{t}_{\text{filete}}^{(p)} = \frac{T_{\text{despinando}}^{(p)}}{F^{(p)}} \tag{2.20}$$

donde $F^{(p)}$ es el total de filetes procesados en el puesto p. El promedio global es la media ponderada sobre todos los puestos activos:

$$\bar{t}_{\text{filete}}^{\text{global}} = \frac{\sum_p T_{\text{despinando}}^{(p)}}{\sum_p F^{(p)}} \tag{2.21}$$

2.8.3. Detección automática de detenciones de línea

2.8.3.1. Definición de detención

Una detención de línea se define como un período continuo de duración mayor o igual a un umbral temporal $\tau_{\text{det}} = 40$ segundos durante el cual no se detecta movimiento de filetes en la zona analizada. Esta definición opera de manera independiente para tres zonas de la línea: entrada, salida y línea completa.

2.8.3.2. Tipos de detención

El sistema clasifica las detenciones en cuatro tipos no excluyentes:

Detención de entrada: período $\geq \tau_{\text{det}}$ sin ingreso de nuevos salmones a la línea. Puede indicar problemas en el suministro de materia prima, mantenimiento de equipos aguas arriba, o pausas de operación planificadas.

Detención de salida: período $\geq \tau_{\text{det}}$ sin egreso de filetes procesados. Puede indicar acumulación de material en la línea, problemas de transporte aguas abajo, o detenciones en etapas posteriores del proceso.

Detención general: período $\geq \tau_{\text{det}}$ en que tanto la entrada como la salida están simultáneamente detenidas. Indica una parada completa de la línea.

Detención total: suma de todas las detenciones registradas de todos los tipos, expresada en tiempo acumulado.

2.8.3.3. Algoritmo de detección de detenciones

El algoritmo de detección de detenciones opera sobre una ventana temporal deslizante de longitud τ_{det}. Para cada zona $z \in \{\text{entrada}, \text{salida}\}$:

$$\text{Det}_z(t) = \mathbb{1}\left[\sum_{k=t-\tau_{\text{det}}}^{t} \Delta F_z(k) = 0\right] \tag{2.22}$$

donde $\Delta F_z(k)$ es el incremento de filetes en la zona z en el instante k. Cuando $\text{Det}_z(t) = 1$ por primera vez, se registra el inicio de una detención. Cuando $\text{Det}_z(t)$ vuelve a 0, se registra el fin de la detención y se calcula su duración.

2.9. Seguridad alimentaria e industria salmonera

2.9.1. Estándares internacionales de inocuidad

La industria salmonera chilena opera bajo un marco regulatorio estricto de estándares internacionales de inocuidad alimentaria. Los principales estándares relevantes para el presente trabajo son:

- **BRC Global Standard for Food Safety (Issue 9)** [6]: estándar desarrollado por el *British Retail Consortium* que establece requisitos detallados para la gestión

de la inocuidad, higiene y seguridad operacional en plantas de procesamiento de alimentos. Incluye requisitos explícitos sobre el uso obligatorio de cofias y EPP en zonas de producción.

- **IFS Food Standard (v7)** [7]: estándar europeo de inocuidad alimentaria ampliamente adoptado por exportadores a mercados europeos. Establece requisitos similares al BRC sobre EPP y trazabilidad del personal.
- **ISO 22000:2018** [8]: norma internacional que especifica los requisitos para sistemas de gestión de la inocuidad alimentaria en cualquier organización de la cadena alimentaria.
- **GLOBALG.A.P. Aquaculture Standard** [9]: estándar específico para el sector acuícola que regula aspectos de bienestar, medioambiente y seguridad laboral, incluyendo el control del uso correcto de EPP.

El cumplimiento simultáneo de estos estándares impone la necesidad de verificación continua del uso correcto de EPP en todas las áreas productivas, lo cual constituye una de las motivaciones directas para el desarrollo del módulo de detección de cofias de este trabajo.

2.9.2. Equipos de protección personal en plantas de alimentos

En plantas de procesamiento de alimentos, el EPP cumple una doble función: protege la integridad física del trabajador y previene la contaminación del producto. Las cofias (mallas de protección capilar) son un elemento de EPP de uso obligatorio en todas las zonas de producción, ya que previenen la contaminación del producto por cabellos o materias extrañas. En Salmones Camanchaca, el sistema de cofias por colores cumple adicionalmente una función de identificación de roles, asignando un color específico a cada categoría de personal:

- Operarios de despinado
- Operarios de fileteado
- Supervisores de turno
- Personal de calidad

- Personal de mantenimiento
- Visitas autorizadas
- Personal de limpieza
- Personal de logística
- Personal externo/contratistas

Esta convención permite que un sistema de visión por computador identifique simultáneamente el cumplimiento del uso de cofia **y** el rol del personal en cámara, utilizando un único detector multiclase [34, 35].

2.10. Justificación del enfoque

A partir de la revisión del estado del arte y considerando las restricciones operativas y de hardware del proyecto, se justifica la elección metodológica de la siguiente forma:

2.10.1. ¿Por qué YOLOv11 y no otra arquitectura?

La elección de YOLOv11 sobre detectores de dos etapas (R-CNN, Faster R-CNN) se fundamenta en el requerimiento de inferencia en tiempo real sobre el *stream* de video continuo de la planta. Los detectores de dos etapas, si bien alcanzan precisiones marginalmente superiores en algunos benchmarks, presentan latencias incompatibles con la operación de 30 FPS requerida [26].

Frente a alternativas de una etapa (SSD, RetinaNet), YOLOv11 ofrece:

- **Mejor relación precisión/parámetros:** 22 % menos parámetros que YOLOv8 con precisión equivalente o superior [25].
- **Ecosistema maduro:** integración nativa con PyTorch, documentación extensa, comunidad activa.
- **Resultados validados en aplicaciones similares:** múltiples trabajos recientes [41] demuestran resultados competitivos con YOLOv11 en escenarios industriales.

- **Mecanismo de atención C2PSA:** mejor desempeño en detección de objetos parcialmente ocluidos, escenario común en plantas de procesamiento donde múltiples operarios y filetes coexisten en proximidad espacial.

2.10.2. ¿Por qué dos modelos independientes y no un único modelo multiclase?

La decisión de entrenar dos modelos YOLOv11 independientes (uno para operarios, otro para filetes) en lugar de un único modelo multiclase se fundamenta en tres consideraciones:

1. **Disponibilidad de datos asimétrica:** la cantidad y calidad de imágenes etiquetadas para cada clase difieren significativamente. Modelos independientes permiten ajustar los hiperparámetros y la composición del dataset de manera específica para cada tarea.
2. **Optimización por clase:** los filetes y operarios presentan características visuales muy distintas (escala, color, dinámica). Modelos especializados pueden alcanzar mayor precisión que un modelo generalista.
3. **Mantenibilidad:** permite actualizar o reentrenar un modelo independientemente del otro, facilitando la evolución incremental del sistema.

2.10.3. ¿Por qué análisis ROI y no *action recognition*?

La elección del enfoque basado en detección + ROI sobre arquitecturas de *action recognition* (SlowFast, I3D, TSN) se justifica por:

- **Costo computacional:** las arquitecturas de *action recognition* requieren procesar secuencias temporales completas, multiplicando el costo computacional respecto a la detección por fotograma. Esto las hace impracticables en hardware modesto como el RTX 4050 disponible para despliegue.
- **Suficiencia del estado actual:** para el problema específico (clasificar el estado de cada puesto en *Despinando*, *Espera* o *Vacío*), la información espacial

instantánea es suficiente. La presencia conjunta de operario y filete en un ROI es indicativa fiable de actividad productiva, sin requerir análisis temporal complejo.

- **Etiquetado más simple:** el etiquetado de *bounding boxes* es significativamente menos costoso que el etiquetado de acciones temporales, lo cual es crítico dada la limitada disponibilidad de personal para anotación.
- **Interpretabilidad:** la lógica determinística (operario + filete = despinando) es más interpretable y depurable que un modelo *end-to-end* de clasificación de acciones.

2.10.4. ¿Por qué hardware modesto (RTX 4050)?

La selección de hardware modesto para el despliegue (NVIDIA RTX 4050 Laptop GPU, 12 GB VRAM) responde a un requerimiento explícito de viabilidad económica del proyecto. Trabajos previos han demostrado que los modelos YOLO modernos pueden operar eficientemente en hardware similar [37], e incluso en dispositivos *edge* aún más restringidos como Jetson Xavier NX, alcanzando precisiones competitivas y latencias adecuadas para aplicaciones en tiempo real.

2.10.5. Síntesis

El enfoque adoptado —dos modelos YOLOv11 independientes para detección de operarios y filetes, combinados con análisis por regiones de interés y un módulo de detección automática de detenciones— constituye una solución técnicamente fundamentada, económicamente viable y operativamente efectiva para el problema planteado. Esta arquitectura integra las mejores prácticas reportadas en la literatura reciente [25, 47, 30] y las adapta al contexto específico del procesamiento de salmón en una planta de la industria salmonera chilena, abordando la brecha tecnológica identificada en la revisión bibliográfica.

Capítulo 3

Desarrollo

3.1. Introducción

El presente capítulo describe el proceso de desarrollo del sistema de monitoreo implementado en la planta de procesos de Salmones Camanchaca, Tomé. El trabajo se organizó en tres líneas tecnológicas: (1) evaluación de SlowFast Networks para reconocimiento de acciones en video; (2) sistema de monitoreo de productividad en tiempo real basado en dos modelos YOLOv11 con análisis por regiones de interés (ROI); y (3) modelo de detección multiclase de cofias como base del futuro verificador de EPP, complementado con un prototipo de verificación multimodal basado en modelos generativos. Cada sección documenta las decisiones técnicas adoptadas, los parámetros utilizados y los resultados intermedios obtenidos.

3.2. Descripción de la línea de despinado

La línea de despinado opera con hasta 10 puestos de trabajo simultáneos (P01 a P10), distribuidos simétricamente a ambos lados de una cinta transportadora central. Los filetes de salmón ingresan por un extremo de la cinta y avanzan hacia los operarios, quienes extraen manualmente las espinas *pin bones*, recolocan el filete procesado y continúan con el siguiente. La planta cuenta con aproximadamente cinco líneas de despinado activas durante los turnos de mayor producción, cada una con hasta diez

puestos de trabajo [10]. Los turnos de operación tienen una duración aproximada de cinco horas, durante las cuales el sistema opera de forma continua.

3.3. Infraestructura de captura de video

Para la captura de video se instaló una cámara IP en posición cenital sobre la línea de despinado, obteniendo una vista superior que permite visualizar simultáneamente los 10 puestos de trabajo y la totalidad de la cinta transportadora. La posición cenital minimiza las oclusiones entre operarios y proporciona una perspectiva consistente de las zonas de procesamiento [47]. El *stream* de video se transmite mediante protocolo RTSP a través de la red local de la planta, sin dependencia de servicios externos. Los parámetros técnicos se resumen en la Tabla 3.1.

Tabla 3.1: Parámetros técnicos del sistema de captura de video.

Parámetro	**Valor**
Resolución	1280 × 720 px (HD 720p)
Frecuencia de fotogramas	30 FPS
Protocolo de transmisión	RTSP
Infraestructura de red	Red local de planta (sin internet)
Posición de instalación	Cenital, perpendicular a la línea
Hardware de inferencia	NVIDIA RTX 4050 Laptop GPU, 12 GB VRAM

3.4. Construcción del dataset de operarios y filetes

3.4.1. Recolección de datos

Se capturaron segmentos de video directamente del *stream* RTSP durante turnos regulares de operación, en sesiones de 30 a 60 minutos. La recolección cubrió deliberadamente distintas condiciones operativas: variaciones de iluminación, cambios en la densidad de operarios, distintas velocidades de procesamiento, y situaciones de espera, puestos vacíos y detenciones de línea.

3.4.2. Etiquetado con Roboflow

El etiquetado se realizó en la plataforma Roboflow [53], identificando dos clases:

- **`operario`:** silueta de cada operario visible, etiquetada con una *bounding box* que incluye desde la cabeza hasta la cintura.
- **`filete`:** cada filete individual visible sobre la cinta o en la zona de trabajo del operario.

Se aplicaron técnicas de *data augmentation* [22] incluyendo volteo horizontal, ajuste de brillo (±20 %), contraste (±15 %), rotación leve (±5°) y recorte aleatorio, mejorando la robustez ante variaciones no vistas [14].

3.4.3. Composición del dataset

El dataset acumula 97.292 instancias etiquetadas entre las dos clases, divididas en los conjuntos indicados en la Tabla 3.2.

Tabla 3.2: Composición del dataset de entrenamiento para detección de operarios y filetes.

Clase	**Total instancias**	**Train (80 %)**	**Val (10 %)**	**Test (10 %)**
Operario	19.710	2.192 imgs	274 imgs	274 imgs
Filete de salmón	77.582			
Total	**97.292**	**2.192 (80 %)**	**274 (10 %)**	**274 (10 %)**

3.5. Evaluación de SlowFast Networks para reconocimiento de acciones

3.5.1. Motivación y arquitectura

Como primera aproximación metodológica, se evaluó la arquitectura SlowFast Networks [50] para clasificar directamente las acciones de los operarios a partir de secuencias de

video. SlowFast Networks, disponible a través de la librería PyTorchVideo, propone dos ramas paralelas:

- **Rama Slow:** procesa la secuencia a baja frecuencia temporal, capturando información espacial detallada (posiciones, texturas).
- **Rama Fast:** procesa a alta frecuencia temporal, capturando dinámica de movimiento con menor resolución espacial.

La fusión de ambas ramas permite distinguir entre estados de actividad que son visualmente similares en fotogramas individuales pero difieren en su dinámica temporal.

3.5.2. Implementación

Se realizó *fine-tuning* sobre el modelo preentrenado de reconocimiento de acciones de Facebook AI [50], adaptándolo al dominio específico del despinado. El proceso fue:

1. **Detección de personas:** YOLOv11 genera *bounding boxes* de los operarios en cada fotograma. Los recortes individuales centrados en cada operario son la entrada para SlowFast.
2. **Construcción de clips:** se generaron clips de 2–3 segundos por operario, etiquetados con una de cinco clases de acción: **despinando**, **espera**, **limpieza**, **conversación** y **otros**.
3. **Fine-tuning:** entrenamiento en Google Colab con GPU NVIDIA Tesla A100.
4. **Seguimiento por puesto:** el algoritmo BotSort [48] mantenía la identidad de cada operario a lo largo del tiempo, atribuyendo acciones a puestos sin necesidad de definir polígonos ROI estáticos.

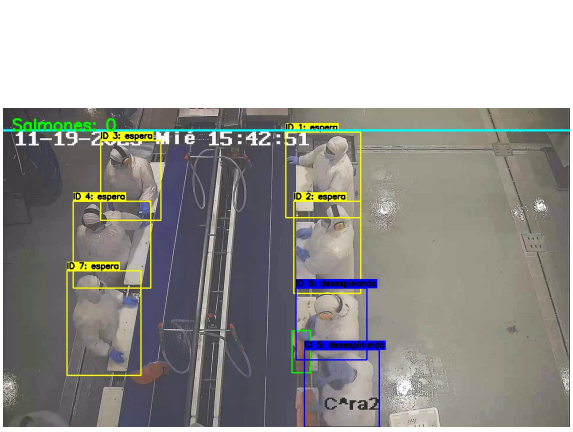
3.5.3. Resultados y decisión de descarte

Los resultados sobre el conjunto de prueba se presentan en la Tabla 3.3. Si bien el desempeño es razonable para las clases principales, la combinación YOLOv11 + SlowFast + BotSort generaba una carga computacional incompatible con la inferencia en tiempo real en el hardware de despliegue (RTX 4050, 12 GB VRAM): el

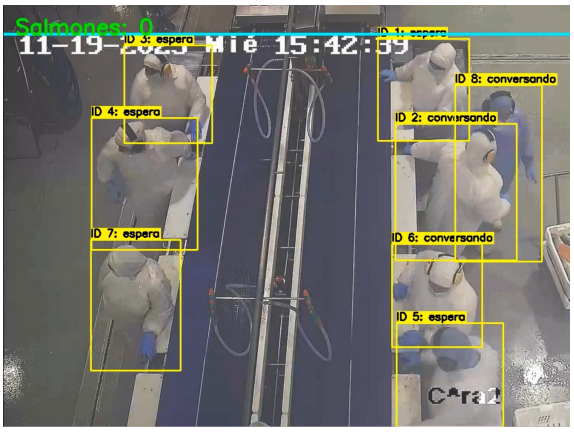
procesamiento de secuencias temporales de 2–3 segundos requería recursos de memoria y cómputo que producían latencias inaceptables para la operación continua. Por esta razón, SlowFast fue descartado como solución de producción y se optó por el enfoque alternativo basado en detección de objetos + ROI, que alcanza precisiones superiores con una fracción del costo computacional.

Tabla 3.3: Métricas de desempeño del modelo SlowFast para reconocimiento de acciones en línea de despinado.

Clase de acción	Precisión	Recall
Despinando	0,881	0,863
Espera	0,854	0,841



(a) Línea de despinado.



(b) Línea de despinado.

Fig. 3.1: Sistema SlowFast Networks + BotSort operando en condiciones reales. Las *bounding boxes* amarillas indican operarios en estado *espera* y las azules operarios en estado *desespinando*. BotSort asigna IDs persistentes a cada operario para atribuir las acciones a puestos específicos a lo largo del tiempo. Fuente: elaboración propia.

3.6. Entrenamiento de los modelos YOLOv11

3.6.1. Configuración del entrenamiento

Se seleccionó la variante YOLOv11L (*large*) con 25,3 millones de parámetros y 86,6 GFLOPs [25]. Esta variante ofrece mayor capacidad representacional para escenas con

alta densidad de objetos y oclusión parcial, sin exceder los recursos del hardware de despliegue. El entrenamiento se ejecutó en Google Colab con GPU NVIDIA Tesla A100 (80 GB VRAM) [61], empleando PyTorch [54] y la implementación oficial de Ultralytics [24]. Los hiperparámetros se detallan en la Tabla 3.4.

Tabla 3.4: Hiperparámetros de entrenamiento del modelo YOLOv11L.

Hiperparámetro	**Valor**
Arquitectura	YOLOv11L (25,3M parámetros)
Épocas máximas	120
Tamaño de imagen (*imgsz*)	640 × 640 px
Optimizador	SGD
Tasa de aprendizaje (*lr0*)	0,001
Momentum	0,937
Hardware de entrenamiento	NVIDIA Tesla A100 (80 GB VRAM)
Plataforma	Google Colab [61]

3.6.2. Resultados del entrenamiento

La Tabla 3.5 presenta las métricas obtenidas sobre el conjunto de prueba para cada clase y para el sistema global de despinado (promedio ponderado de ambas clases). Ambos modelos superaron los objetivos de precisión definidos (mAP@0.5 > 0,90).

Tabla 3.5: Métricas de desempeño de los modelos YOLOv11L entrenados.

Modelo / Clase	**Precisión**	**Recall**	**mAP@0.5**	**Observación**
YOLOv11 — Operario	0,934	0,912	0,951	Presencia/ausencia por puesto
YOLOv11 — Filete salmón	0,891	0,873	0,903	Conteo en línea
YOLOv11 — Global	**0,912**	**0,892**	**0,927**	Promedio ponderado 2 clases

El modelo de operarios alcanzó el mayor desempeño (mAP@0.5 = 0,951). El modelo de filetes obtuvo mAP@0.5 = 0,903, resultado ligeramente menor explicado por la mayor variabilidad visual de los filetes (distintos tamaños, posiciones y grados de acumulación en la cinta). Ambos resultados son comparables o superiores a los reportados en trabajos similares de detección de EPP y monitoreo industrial [36, 39].

Umbrales de confianza en inferencia: a partir del análisis de las curvas precisión-recall obtenidas durante la validación, se seleccionaron umbrales de confianza diferenciados para cada modelo: $\theta_{\text{operario}} = 0{,}50$ y $\theta_{\text{filete}} = 0{,}20$. El umbral más bajo para el modelo de filetes se justifica por las características del objeto detectado: los filetes presentan alta variabilidad visual (distintos tamaños, solapamientos y cambios de iluminación), por lo que un umbral conservador generaría un número excesivo de falsos negativos que comprometería el conteo de entrada y salida de la línea. En contraste, la silueta humana es más consistente y un umbral más estricto para operarios reduce eficazmente las detecciones espurias sobre fondos complejos. Esta asimetría de umbrales fue validada empíricamente durante las pruebas de operación continua.

3.7. Pipeline de inferencia en tiempo real

3.7.1. Arquitectura general

El *pipeline* fue implementado en Python con PyTorch [54], Ultralytics YOLOv11 [24] y OpenCV [55]. El flujo de procesamiento por fotograma se ilustra en la Figura 3.2 y comprende los pasos de la Tabla 3.6.

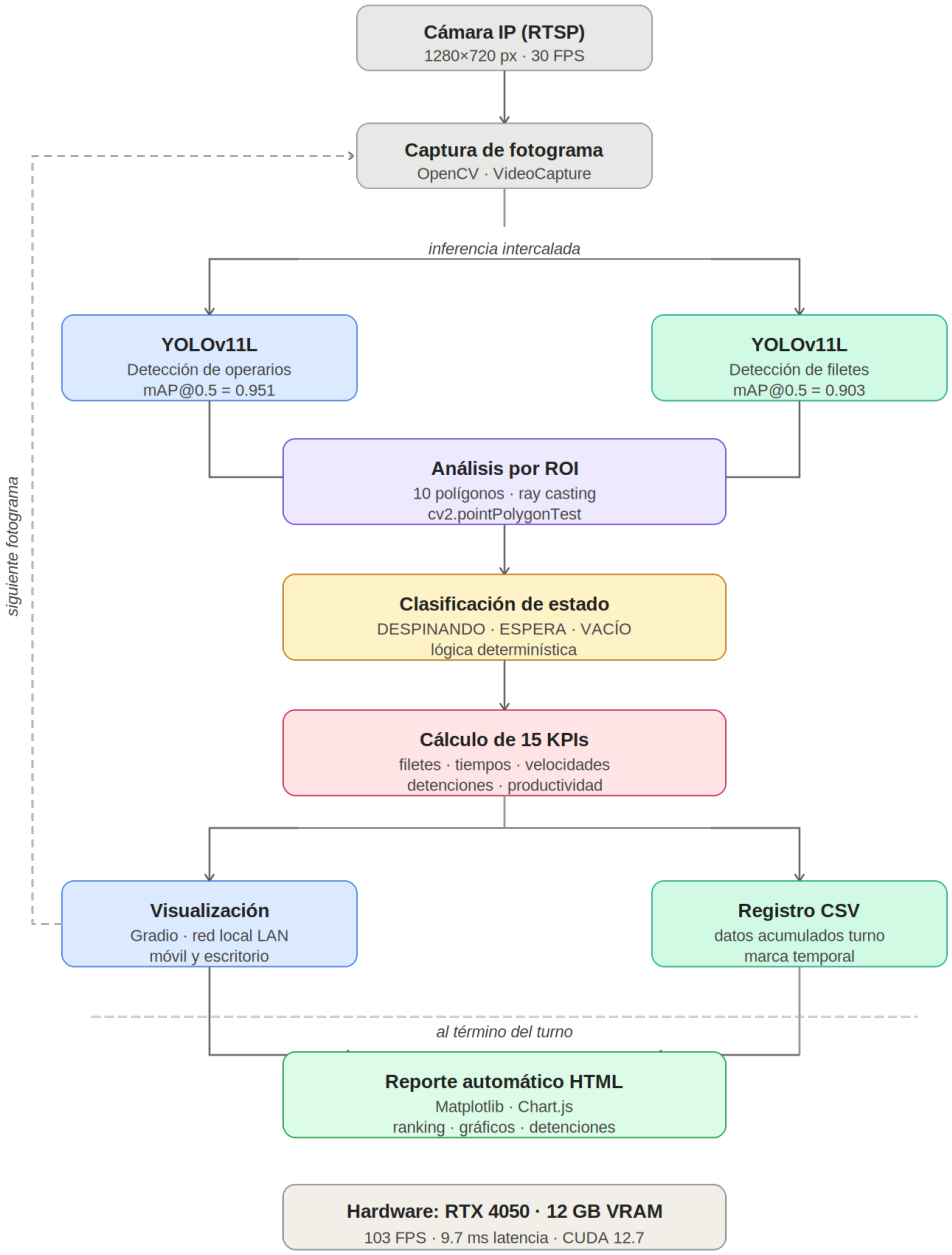


Fig. 3.2: Diagrama de flujo del *pipeline* de inferencia en tiempo real. Se capturan fotogramas vía RTSP y se procesan con dos modelos YOLOv11L de forma intercalada. Las detecciones se combinan en el módulo de análisis por ROI para clasificar el estado de cada puesto y calcular los 15 indicadores operativos. La línea discontinua representa el *loop* de retroalimentación continua. Al término del turno, los datos CSV se convierten en reporte HTML automático. Fuente: elaboración propia.

Tabla 3.6: Pasos del *pipeline* de inferencia en tiempo real.

Paso	**Descripción**
1	Captura del fotograma desde el *stream* RTSP
2	Inferencia del modelo YOLOv11 de operarios
3	Inferencia del modelo YOLOv11 de filetes
4	Análisis de co-detección por ROI: presencia de operario y filete por puesto
5	Clasificación del estado de cada puesto (DESPINANDO / ESPERA / PUESTO VACÍO)
6	Actualización de los 15 indicadores operativos
7	Verificación de condiciones de detención de línea (umbral $\tau_{\text{det}} = 40$ s)
8	Visualización con anotaciones superpuestas y panel de métricas
9	Escritura periódica de datos al archivo CSV del turno

3.7.2. Inferencia intercalada

Dado que el sistema ejecuta dos modelos YOLOv11 independientes sobre el mismo *stream*, se implementó una arquitectura de inferencia intercalada para maximizar el uso de la GPU. La latencia total en producción es de 9,7 ms por fotograma (captura + doble inferencia + análisis ROI + visualización), equivalente a aproximadamente 103 FPS sobre el hardware de despliegue. La Figura 3.3 muestra el sistema en operación real durante un turno regular.

Fig. 3.3: Sistema de monitoreo de productividad operando en tiempo real sobre la línea de despinado. Se visualizan los 10 puestos etiquetados con su estado (DESPINANDO en verde, PUESTO VACÍO en rojo) y el contador acumulado de entradas y salidas (ENTRADAS: 138, SALIDAS: 137). Fuente: elaboración propia.

Fig. 3.4: Fotograma del *pipeline* en tiempo real durante un período de alta actividad productiva. Los recuadros azules corresponden a detecciones de operarios; los recuadros verdes a detecciones de filetes de salmón. La mayoría de los puestos se encuentra en estado DESPINANDO. Fuente: elaboración propia.

3.8. Definición de regiones de interés (ROI)

3.8.1. Polígonos de puesto

Para cada puesto P01–P10 se definió manualmente un polígono convexo que delimita su zona de procesamiento: mesa de trabajo, sección de cinta adyacente y espacio donde el operario manipula filetes. Esta estrategia desacopla la detección de objetos de la atribución espacial: el modelo detecta en toda la imagen y la lógica ROI asigna cada detección al puesto correspondiente [47].

Las coordenadas de los polígonos se almacenan en `puestos_backup.json` en el directorio raíz del *pipeline*. La ruta se resuelve dinámicamente para evitar errores al ejecutar desde distintos directorios:

```
import os
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
PUESTOS_PATH = os.path.join(BASE_DIR, "puestos_backup.json")
```

Listing 3.1: Resolución dinámica de la ruta al archivo de polígonos ROI.

3.8.2. Herramientas de edición

Se desarrollaron dos herramientas complementarias:

- **`editor_puesto10.py`:** editor interactivo que permite agregar, mover o eliminar vértices mediante clics del ratón, con guardado automático en `puestos_backup.json`.
- **`ver_puestos.py`:** herramienta de visualización para verificar la cobertura de los ROI sobre fotogramas estáticos del *stream*.

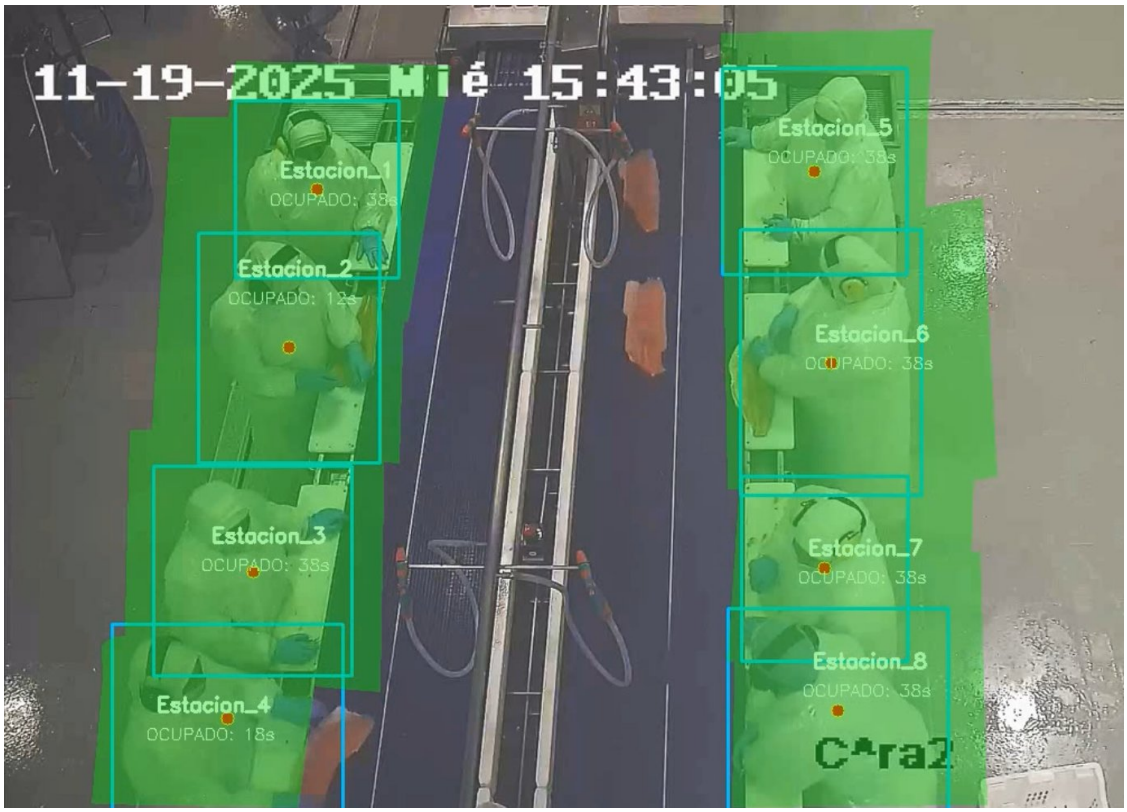


Fig. 3.5: Regiones de interés (ROI) poligonales superpuestas sobre el fotograma de la cámara cenital (19 de noviembre de 2025, 15:43:05). Cada estación (Estacion_1 a Estacion_8) tiene un polígono de color verde que delimita su zona de procesamiento, con el tiempo acumulado en estado OCUPADO y el centroide en naranja como referencia espacial. Fuente: elaboración propia.

3.9. Lógica de clasificación de estados

3.9.1. Algoritmo de co-detección

Para cada puesto p y fotograma t, el sistema evalúa la presencia de detecciones dentro del polígono ROI correspondiente mediante el algoritmo *ray casting* implementado con `cv2.pointPolygonTest` [55]. La clasificación se determina mediante la lógica:

```
def clasificar_estado(operarios_en_roi, filetes_en_roi):
    if operarios_en_roi > 0 and filetes_en_roi > 0:
        return "DESPINANDO"
    elif operarios_en_roi > 0 and filetes_en_roi == 0:
```

```
5           return "ESPERA"
6       else:
7           return "PUESTO␣VACIO"
```

Listing 3.2: Clasificación de estado por puesto basada en co-detección en ROI.

3.9.2. Código de colores en tiempo real

Cada puesto es anotado en el fotograma con su etiqueta y un código de color:

- **Verde** → DESPINANDO: puesto activo y produciendo.
- **Amarillo** → ESPERA: operario presente sin material.
- **Rojo** → PUESTO VACÍO: sin operario asignado.

Se muestran también los contadores globales de entradas y salidas de la línea, actualizados en cada fotograma.

3.10. Módulo de detección de detenciones de línea

3.10.1. Definición operativa

Una detención se define como un período continuo $\geq \tau_{\text{det}} = 40$ s sin movimiento de filetes en la zona analizada. Este umbral fue establecido empíricamente: en operación normal los intervalos entre filetes raramente superan los 10–15 segundos, por lo que 40 segundos discrimina efectivamente las paradas reales de las variaciones operativas normales.

3.10.2. Cuatro tipos de detención

Detención de entrada: no ingresan nuevos filetes durante ≥ 40 s. Puede indicar problemas en el suministro de materia prima aguas arriba.

Detención de salida: no egresan filetes procesados durante ≥ 40 s. Puede indicar acumulación en línea o problemas aguas abajo.

Detención general: entrada y salida simultáneamente detenidas durante $\geq$ 40 s. Indica parada completa de la línea.

Detención total: suma acumulada de tiempos de todos los tipos registrados en el turno.

Cada detención se registra con marca temporal de inicio, fin y duración, alimentando el reporte al cierre del turno.

3.11. Módulo de indicadores operativos en tiempo real

El sistema calcula y mantiene actualizados 15 indicadores operativos a lo largo del turno, acumulados fotograma a fotograma y exportados periódicamente a un archivo CSV con marca temporal:

1. Tiempo de presencia por puesto (con operario vs. vacío).
2. Filetes despinados por puesto.
3. Tiempo de espera por puesto (min).
4. Tiempo despinando por puesto (min).
5. Distribución temporal del trabajo por puesto (despinando vs. espera en %).
6. Tiempo promedio de despinado por filete por puesto (s/filete).
7. Tiempo promedio global de despinado por filete (s/filete).
8. Filetes procesados por minuto (fil/min).
9. Productividad vs. tiempo efectivo de trabajo por puesto.
10. Total de detenciones de línea registradas.
11. Total de detenciones generales.
12. Total de detenciones de entrada.
13. Total de detenciones de salida.

14. Tiempo acumulado en detenciones por tipo (hr:min).
15. Tiempo total acumulado en detenciones (hr:min).

Nota metodológica sobre la estimación de filetes por puesto: dado que la cámara cenital proporciona una vista única sin identificación individual de cada filete por puesto de destino, los filetes despinados asignados a cada puesto p se calculan mediante una distribución proporcional al tiempo activo acumulado:

$$\hat{F}_p = F_{\text{total}} \cdot \frac{t_{\text{desp},p}}{\sum_{p'} t_{\text{desp},p'}} \tag{3.1}$$

donde F_{total} es el total de filetes contabilizados en la línea y $t_{\text{desp},p}$ es el tiempo acumulado en estado DESPINANDO para el puesto p. Este enfoque de imputación proporcional es una aproximación válida bajo el supuesto de que la velocidad de despinado es homogénea entre puestos activos. Las diferencias individuales en velocidad de despinado se capturan de forma complementaria a través del indicador *tiempo promedio por filete* (indicador 6), que sí refleja la eficiencia individual de cada puesto.

3.12. Interfaz de monitoreo en tiempo real

La interfaz fue desarrollada con Gradio [56] y es accesible desde dispositivos móviles y escritorio dentro de la red local de la planta, sin instalación de software adicional. Muestra:

- **Vista en vivo:** fotograma procesado con detecciones y estados de los 10 puestos.
- **Indicador de estado global:** banner verde (EN FUNCIONAMIENTO) o rojo (DETENIDA – ENTRADA / SALIDA / GENERAL), con alerta automática al superar el umbral de 40 segundos.
- **Panel de métricas:** contadores de entradas y salidas, tiempo transcurrido del turno.
- **Gráficos de eficiencia:** eficiencia por puesto y curva horaria de producción acumulada.

Fig. 3.6: Interfaz de monitoreo con estado EN FUNCIONAMIENTO (banner verde). Múltiples puestos activos con filetes en procesamiento. Fuente: elaboración propia.

Fig. 3.7: Interfaz de monitoreo con estado DETENIDA – SALIDA (banner rojo). El sistema detecta y alerta automáticamente al superar los 40 segundos sin egreso de filetes. Fuente: elaboración propia.

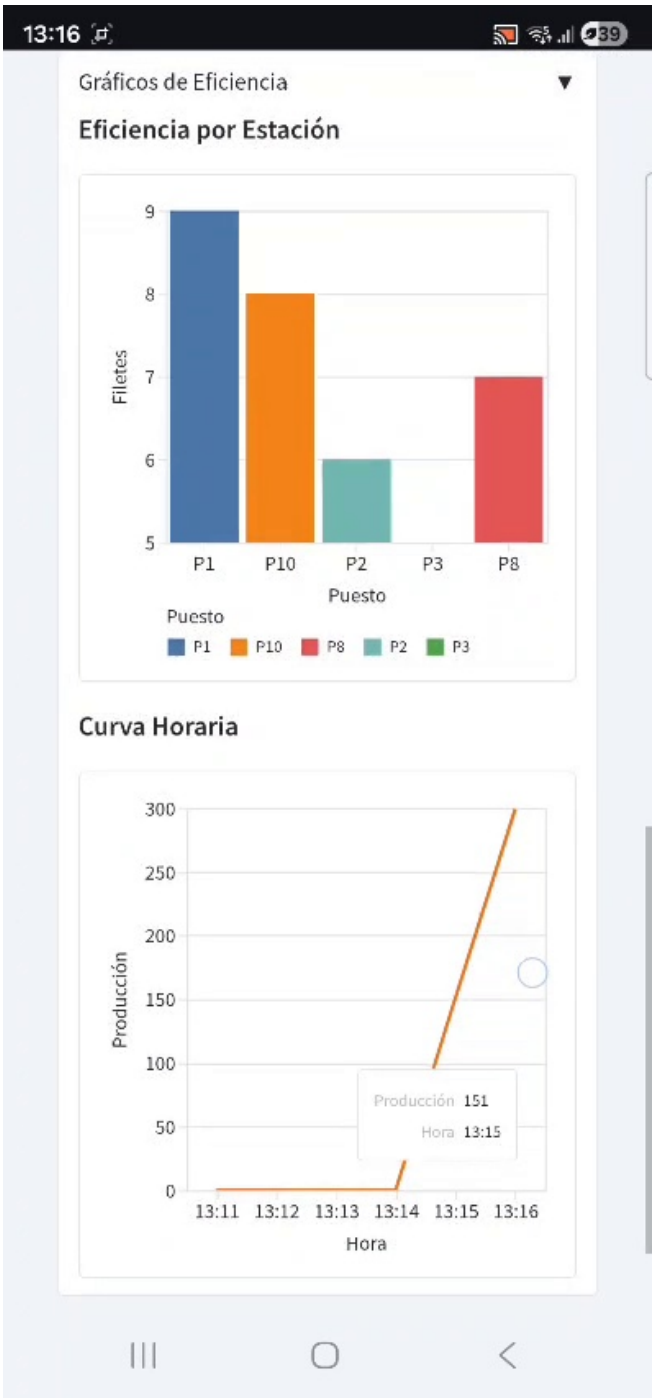


Fig. 3.8: Panel de gráficos de eficiencia de la interfaz de monitoreo accedido desde dispositivo móvil. Se muestran el gráfico de eficiencia por estación (filetes procesados por puesto) y la curva horaria de producción acumulada durante el turno. Fuente: elaboración propia.

3.13. Generación automática de reportes diarios

3.13.1. Flujo de generación

Al término de cada turno el sistema genera automáticamente un reporte HTML mediante el siguiente flujo:

1. Lectura del archivo CSV con datos acumulados del turno.
2. Cálculo de los 15 indicadores operativos finales del turno.
3. Generación de gráficos estáticos con Matplotlib [62].
4. Construcción de la página HTML con gráficos interactivos (Chart.js), tablas de resultados y registro de detenciones.
5. Escritura del archivo `reporte_AAAA-MM-DD.html` en el directorio de reportes.

3.13.2. Contenido del reporte

El reporte incluye: encabezado con fecha y horario del turno; resumen con KPIs globales; tabla de producción por puesto (filetes, tiempos, productividad); gráficos de barras por puesto y evolución temporal; registro completo de detenciones con tipo y duración; y ranking de productividad. La Figura 3.9 muestra el reporte correspondiente al turno del 9 de marzo de 2026, cuyo análisis detallado se presenta en el Capítulo 4.

A modo de ejemplo, los KPIs globales del reporte automático generado para el turno 2026-03-09 (13:29–18:39) fueron: **5.098 filetes procesados**, 5.098 entradas y 5.052 salidas a la línea, velocidad promedio de entrada de 29,7 fil/min y de salida de 29,2 fil/min, tiempo promedio por filete de 11,2 segundos, 950,9 minutos totales en estado despinando y solamente 2,5 minutos acumulados de detenciones de línea durante las cinco horas de operación.

RANKING DE PRODUCTIVIDAD POR PUESTO

#	PUESTO	FILETES	AVG S/FIL	T. TRABAJO (MIN)	T. ESPERA (MIN)	T. VACÍO (MIN)	PRODUCTIVIDAD
🥇	P10	759	13s	141.6 min	22.7 min	108.9 min	100%
🥈	P06	687	14.9s	128.1 min	42.8 min	102.4 min	91%
🥉	P02	622	15.1s	116.1 min	40 min	117.2 min	82%
4	P01	616	12.9s	114.9 min	17.2 min	141.1 min	81%
5	P08	600	13.6s	112 min	23.7 min	137.6 min	79%
6	P07	599	15.9s	111.7 min	46.6 min	114.9 min	79%
7	P05	530	16.3s	98.8 min	44.9 min	129.5 min	70%
8	P04	468	15.8s	87.2 min	35.9 min	150.1 min	62%
9	P03	152	17.2s	28.3 min	15.2 min	229.8 min	20%
10	P09	65	20.1s	12.2 min	9.5 min	251.5 min	9%

Fig. 3.9: Panel de productividad del reporte automático generado para el turno 2026-03-09 (13:29–18:39). Panel izquierdo: filetes procesados por puesto (P10 lideró con 759 filetes, seguido por P06 con 687 y P02 con 622; P09 registró sólo 65 filetes durante el turno). Panel derecho: distribución del tiempo por estado y puesto (verde = despinando, naranja = espera, rojo = vacío). Fuente: elaboración propia.

3.14. Dataset y modelo de detección de cofias (EPP)

3.14.1. Convención de colores de cofia en planta

En la planta de Salmones Camanchaca existe una convención en la que cada rol de personal utiliza una cofia de color específico. El modelo aprovecha esta convención para identificar simultáneamente el cumplimiento del uso de cofia obligatoria y el rol del personal en cámara, sin requerir ningún identificador adicional [34]. Las nueve clases y sus colores correspondientes se detallan en la Tabla 3.7.

Tabla 3.7: Clases del modelo de detección de cofias y su correspondencia con roles del personal.

Clase	Color de cofia	Rol del personal
1	Verde claro	Calidad
2	Verde oscuro	Comité paritario
3	Celeste	No identificado
4	Azul	Operador cofia azul
5	Blanco	Operador cofia blanca
6	Morado	Operador cofia morada
7	Naranja	Operador cofia naranja
8	Rojo	Personal de limpieza
9	Amarillo	Supervisor

Fig. 3.10: Leyenda de las nueve clases de cofia implementadas en el modelo de detección de EPP, con su color y rol asociado del personal: calidad (verde claro), comité paritario (verde oscuro), no identificado (celeste), operador cofia azul, operador cofia blanca, operador cofia morada, operador cofia naranja, personal de limpieza (rojo) y supervisor (amarillo). Fuente: elaboración propia.

3.14.2. Construcción del dataset de cofias

Se construyó un dataset de 10.197 imágenes capturadas en distintas zonas y condiciones de la planta: perspectivas cenitales y laterales en líneas de producción, pasillos, zona de vestuarios y accesos. El etiquetado se realizó en Roboflow [53]. La distribución fue

90/6/4 (entrenamiento/validación/prueba), como se detalla en la Tabla 3.8.

Tabla 3.8: Composición del dataset de entrenamiento del modelo de cofias (EPP).

Conjunto	**Train (90 %)**	**Val (6 %)**	**Test (4 %)**	**Total**
Imágenes	9.189	607	401	**10.197**
Clases	9 (un color de cofia por clase)			

3.14.3. Entrenamiento y resultados

El modelo YOLOv11 multiclase de 9 clases se entrenó en Google Colab con GPU NVIDIA Tesla A100 [61]. Alcanzó un mAP@0.5 global de 0,894, con precisión promedio de 0,887 y *recall* promedio de 0,872, superando el objetivo de mAP@0.5 $>$ 0,85. Este resultado es comparable a los reportados en la literatura para detección de EPP multiclase en entornos industriales [35, 36], validando la viabilidad técnica de la identificación automática de cofias por color.

Fig. 3.11: Modelo de detección de color de cofia operando sobre la línea de fileteado. Las *bounding boxes* de color magenta identifican a los operarios con su clase de cofia detectada: *operador cofia morada* (ID 0, ID 1, ID 2) y *operador cofia azul* (ID 3). Fuente: elaboración propia.

Fig. 3.12: Modelo de detección de EPP operando en condiciones reales en distintas zonas de la planta. Las *bounding boxes* de distintos colores identifican a los operarios según el color de su cofia y, por tanto, su rol: naranja (operador cofia naranja), rojo (personal de limpieza), blanco (operador cofia blanca) y azul (operador cofia azul). Fuente: elaboración propia.

3.14.4. Prototipo de verificación multimodal de EPP

Como extensión del modelo de cofias, se desarrolló un prototipo de verificador integral de EPP basado en una arquitectura híbrida de dos etapas, documentada en un trabajo de investigación asociado [1]. El flujo completo del sistema se ilustra en la Figura 3.13.

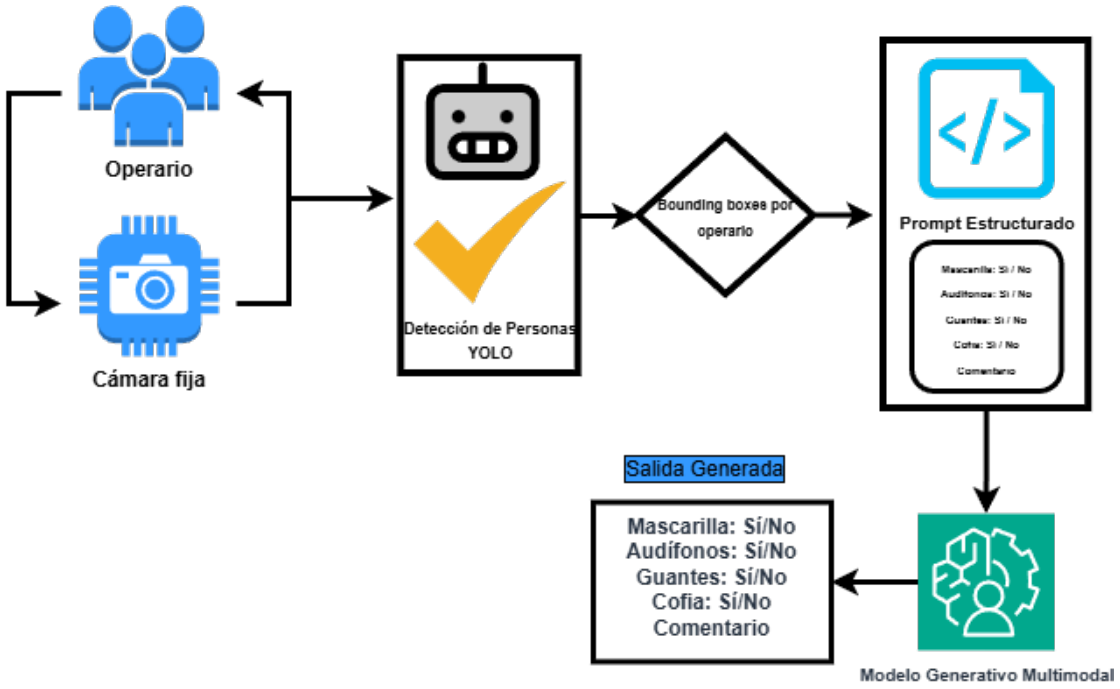


Fig. 3.13: Diagrama del *pipeline* de verificación multimodal de EPP propuesto en [1]. La cámara fija captura imágenes de los operarios; un detector YOLO genera *bounding boxes* individuales por operario; cada recorte se envía junto con un *prompt* estructurado a un modelo generativo multimodal, que produce una salida con la verificación binaria del uso de mascarilla, audífonos, guantes y cofia, más un comentario descriptivo. Fuente: [1].

El *pipeline* opera en dos etapas sucesivas:

1. **Detección (YOLOv11):** se detecta cada operario y se genera un recorte individual de su *bounding box*.
2. **Análisis multimodal:** cada recorte se envía a un modelo generativo multimodal con un *prompt* estructurado que solicita verificar de forma binaria (sí/no) el uso de mascarilla, audífonos de protección, guantes y cofia, más un comentario descriptivo.

Se compararon tres modelos generativos bajo un protocolo experimental común [63]: OpenAI GPT-4.1 [64], Google Gemini Pro [65] y LLaVA (*open-source*) [66]. Los resultados sobre 50 imágenes de operarios en condiciones reales se resumen en la Tabla 3.9.

Tabla 3.9: Comparación de modelos generativos multimodales para verificación de EPP sobre 50 imágenes en condiciones reales de planta [1].

Modelo	**Precisión**	**Coherencia**	**Detalle**	**Consistencia**
OpenAI GPT-4.1	100 %	100 %	100 %	100 %
Google Gemini Pro	90 %	92 %	96 %	94 %
LLaVA (*open-source*)	78 %	80 %	86 %	82 %

Los modelos propietarios presentan mayor estabilidad y precisión descriptiva. Los errores de Gemini Pro se concentraron en la detección de audífonos de protección (4 imágenes) y guantes (1 imagen). LLaVA, si bien muestra un desempeño inferior, constituye una alternativa viable para despliegues a gran escala donde el costo por consulta API y el control sobre los datos son factores críticos [1].

3.15. Stack tecnológico del sistema

La Tabla 3.10 resume las herramientas y bibliotecas utilizadas en el desarrollo de todos los componentes.

Tabla 3.10: Herramientas y bibliotecas tecnológicas del sistema desarrollado.

Herramienta	**Versión**	**Función**
Python	3.10+	Lenguaje de desarrollo principal
PyTorch [54]	2.x	Framework de deep learning
Ultralytics YOLOv11 [24]	11.x	Detección de objetos en tiempo real
PyTorchVideo / SlowFast [50]	—	Reconocimiento de acciones (evaluado)
GPT-4.1 / Gemini Pro / LLaVA	—	Verificación multimodal de EPP
Roboflow [53]	Cloud	Etiquetado y gestión de datasets
Google Colab + A100 [61]	NVIDIA A100	Entrenamiento (80 GB VRAM)
OpenCV [55]	4.x	Captura y preprocesamiento de video
Gradio [56]	3.x	Interfaz web en tiempo real
Matplotlib [62]	3.x	Generación de gráficos
HTML + Chart.js	4.4.0	Reportes diarios interactivos
NVIDIA CUDA [59]	12.7	Cómputo paralelo en GPU

3.16. Resumen del sistema implementado

La Tabla 3.11 presenta un resumen consolidado de los tres componentes del sistema con sus tecnologías y resultados alcanzados.

Tabla 3.11: Resumen del sistema de monitoreo implementado en Salmones Camanchaca, Tomé.

Componente	**Tecnología**	**Resultado**
Detección operarios	YOLOv11L	mAP@0.5 = 0,951 · P = 0,934
Detección filetes	YOLOv11L	mAP@0.5 = 0,903 · P = 0,891
Análisis por ROI	10 polígonos · ray casting	10 puestos simultáneos
Clasificación estados	Lógica de co-detección	3 estados por puesto
Detenciones de línea	Ventana temporal 40 s	4 tipos clasificados
Indicadores operativos	Cálculo en tiempo real	15 KPIs
Interfaz monitoreo	Gradio · red local LAN	Móvil y escritorio
Reportes automáticos	HTML · Matplotlib · Chart.js	Al término de cada turno
Modelo de cofias	YOLOv11 · 9 clases	mAP@0.5 = 0,894 · P = 0,887
Verificador EPP	YOLOv11 + modelos generativos	Prototipo validado [1]
SlowFast (evaluado)	PyTorchVideo + BotSort	Descartado por costo cómputo
Hardware despliegue	RTX 4050 · 12 GB VRAM	103 FPS · 9,7 ms latencia

Capítulo 4

Resultados

4.1. Introducción

El presente capítulo reporta los resultados obtenidos en la validación del sistema de monitoreo desarrollado. La evaluación se organiza en tres líneas: (1) desempeño de los modelos de detección YOLOv11 en términos de métricas de precisión; (2) resultados operativos del sistema de monitoreo de productividad durante un turno real en la planta de Salmones Camanchaca; y (3) resultados del módulo de verificación de EPP basado en modelos generativos multimodales.

4.2. Resultados del modelo de detección YOLOv11

4.2.1. Desempeño de los modelos entrenados

Ambos modelos YOLOv11L fueron evaluados sobre sus respectivos conjuntos de prueba. La Tabla 4.1 resume las métricas obtenidas. El modelo de operarios alcanzó una precisión de 0,934 y un mAP@0.5 de 0,951, mientras que el modelo de filetes obtuvo una precisión de 0,891 y un mAP@0.5 de 0,903. El desempeño global del sistema combinado, expresado como el promedio ponderado de ambas clases, es de mAP@0.5 = 0,927.

Tabla 4.1: Métricas de desempeño de los modelos YOLOv11L sobre el conjunto de prueba.

Modelo	**Precisión**	**Recall**	**mAP@0.5**	**mAP@0.5:0.95**
YOLOv11 — Operario	0,934	0,912	0,951	—
YOLOv11 — Filete	0,891	0,873	0,903	—
Global (prom.)	**0,912**	**0,892**	**0,927**	**0,866**

El menor desempeño del modelo de filetes respecto al de operarios se explica por la mayor variabilidad visual de los filetes de salmón: distintos tamaños, grados de procesamiento, posiciones sobre la cinta y acumulaciones en la zona de trabajo. En contraste, los operarios presentan una silueta más consistente, favoreciendo la detección. Ambos modelos superaron el umbral de mAP@0.5 $>$ 0,90 definido como criterio de aceptación.

4.2.2. Desempeño del modelo de cofias

El modelo YOLOv11 de 9 clases para detección de cofias fue evaluado sobre el conjunto de prueba (401 imágenes). Los resultados se presentan en la Tabla 4.2.

Tabla 4.2: Métricas de desempeño del modelo YOLOv11 de detección de cofias (9 clases).

Métrica	**Valor**	**Objetivo**	**Resultado**
Precisión promedio	0,887	$>$ 0,85	Cumplido
Recall promedio	0,872	$>$ 0,80	Cumplido
mAP@0.5 global	0,894	$>$ 0,85	Cumplido

El modelo superó los objetivos de desempeño en las tres métricas evaluadas, validando la viabilidad de identificar automáticamente el rol del personal a través del color de cofia en condiciones reales de planta.

4.2.3. Comparación con la literatura

La Tabla 4.3 compara los modelos desarrollados con trabajos relacionados de la literatura en detección de EPP y monitoreo de personal en entornos industriales.

Tabla 4.3: Comparación del sistema desarrollado con trabajos relacionados de detección de EPP y monitoreo industrial.

Trabajo	**Modelo**	**Tarea**	**mAP@0.5**	**Precisión**
Este trabajo	**YOLOv11L**	**Operarios + Filetes**	**0,927**	**0,912**
Este trabajo	**YOLOv11L**	**Cofias (9 clases)**	**0,894**	**0,887**
Wang et al. [36]	YOLOv8x	EPP construcción	0,920	—
Ferdous et al. [35]	YOLOX-m	EPP (8 clases)	0,898	—
Liu et al. [39]	SD-YOLOv5s	EPP	0,937	—
Zhang et al. [67]	U-Net+YOLOv5	Salmón segmentación	0,969	—
Islam et al. [47]	YOLOv6+ROI	Vigilancia ROI	0,923	—

Los resultados obtenidos son comparables o superiores a los trabajos de la literatura en detección de EPP y monitoreo industrial. El sistema propuesto destaca por integrar simultáneamente detección de dos clases, análisis por ROI y clasificación de estado operativo, lo que no se encuentra en ninguno de los trabajos comparados de forma integrada.

4.3. Resultados operativos del sistema de monitoreo

4.3.1. Descripción del turno validado

La validación del sistema de monitoreo de productividad se realizó durante el turno operativo del 9 de marzo de 2026 en la planta de procesos de Salmones Camanchaca. Las condiciones generales del turno se detallan en la Tabla 4.4.

Tabla 4.4: Características del turno validado (2026-03-09).

Parámetro	**Valor**
Fecha	9 de marzo de 2026
Hora de inicio	13:29
Hora de término	18:39
Duración total	5 horas 10 minutos
Puestos monitoreados	10 (P01 – P10)
Total de filetes procesados	5.098
Total de entradas a la línea	5.098
Total de salidas de la línea	5.052

4.3.2. Indicadores globales del turno

La Tabla 4.5 presenta los indicadores operativos globales calculados por el sistema al término del turno.

Tabla 4.5: Indicadores operativos globales del turno 2026-03-09.

Indicador	**Valor**
Filetes procesados (total)	5.098
Velocidad promedio de entrada	29,7 fil/min
Velocidad promedio de salida	29,2 fil/min
Tiempo promedio por filete	11,2 seg/filete
Tiempo total en estado despinando	950,9 min
Detenciones registradas	3
Tiempo total en detenciones	2,5 min

El sistema registró únicamente 2,5 minutos acumulados de detenciones de línea durante las cinco horas de operación, lo que equivale a un índice de disponibilidad del 99,2 %, indicando una operación altamente continua durante el turno analizado. Cabe destacar que el horario monitoreado (13:29–18:39) incluye tanto los períodos de procesamiento activo como los intervalos no productivos asociados a colación del personal y a las labores de limpieza de la línea posteriores al término de la producción, situaciones

que se reflejan en los tiempos en estado PUESTO VACÍO presentados en las secciones siguientes y que no corresponden a detenciones operativas sino a pausas planificadas dentro del turno.

4.3.3. Productividad por puesto

La Figura 4.1 muestra el gráfico de filetes procesados por puesto generado automáticamente por el sistema. La diferencia entre el puesto más productivo (P10, 759 filetes) y el menos productivo (P09, 65 filetes) es significativa, evidenciando una marcada heterogeneidad operacional entre puestos durante el turno.

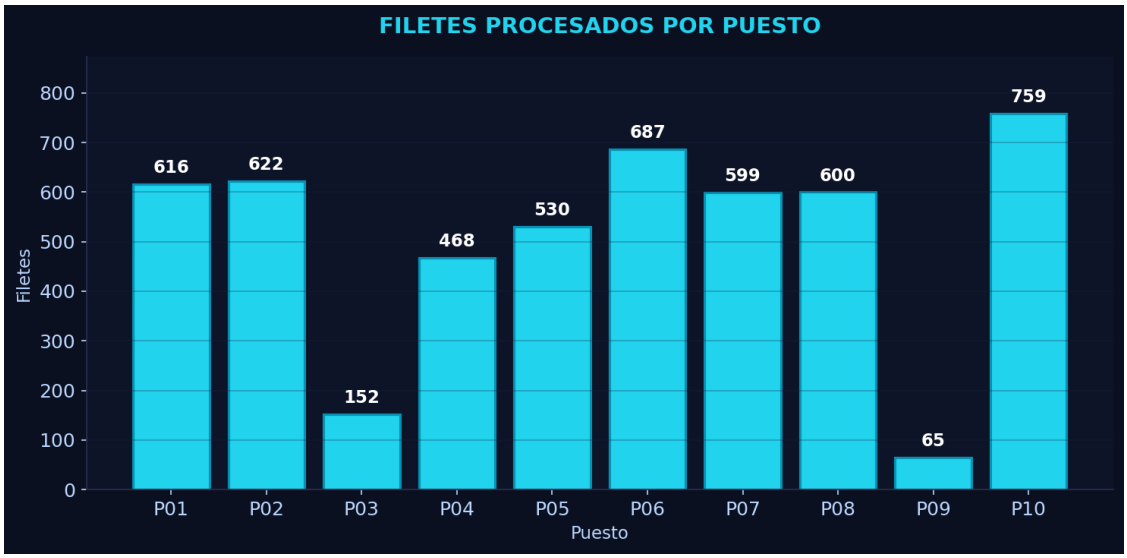


Fig. 4.1: Filetes procesados por puesto durante el turno 2026-03-09. P10 lidera con 759 filetes, seguido por P06 (687) y P02 (622). P03 y P09 presentan los valores más bajos (152 y 65 respectivamente). Fuente: reporte automático del sistema, elaboración propia.

La Figura 4.2 complementa el análisis presentando el tiempo promedio por filete en cada puesto, métrica que refleja la velocidad individual de procesamiento independientemente del tiempo total operado. Un valor menor indica mayor rapidez de despinado por unidad.

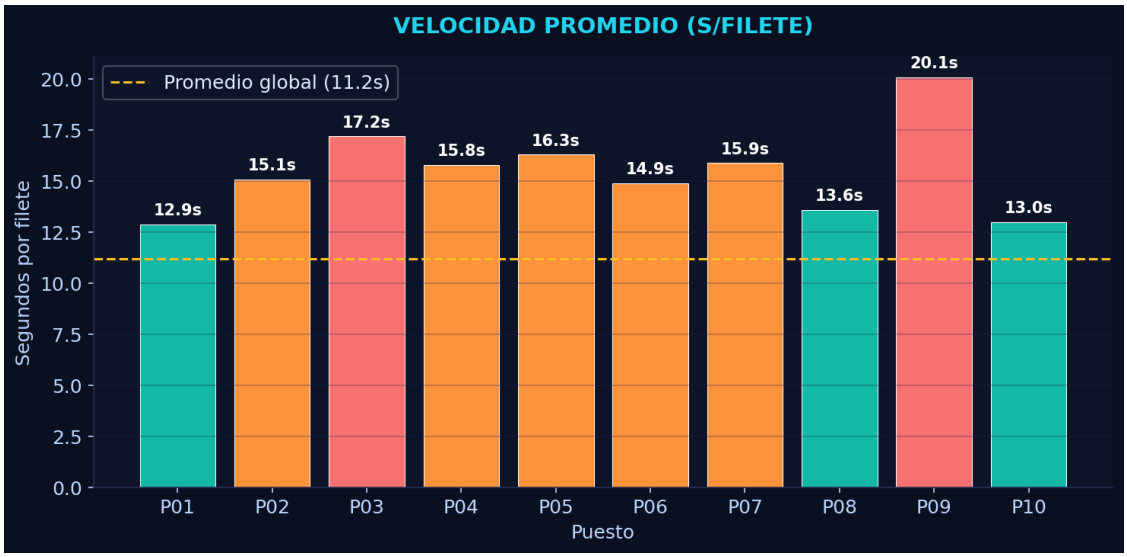


Fig. 4.2: Velocidad promedio de procesamiento por puesto, expresada en segundos por filete (menor valor = mayor rapidez individual). Los puestos P01 (12,9 s/filete) y P10 (13,0 s/filete) presentan los menores tiempos unitarios, indicando mayor rapidez de despinado. La línea horizontal indica el promedio global del turno (11,2 s/filete). Fuente: reporte automático del sistema, elaboración propia.

La Tabla 4.6 presenta el ranking completo de productividad de los 10 puestos con los valores numéricos exactos.

Tabla 4.6: Ranking de productividad por puesto, turno 2026-03-09.

Posición	Puesto	Filetes procesados	Tiempo prom. (seg/filete)
1	P10	759	13,0
2	P06	687	14,9
3	P02	622	15,1
4	P01	616	12,9
5	P08	600	13,6
6	P07	599	15,9
7	P05	530	16,3
8	P04	468	15,8
9	P03	152	17,2
10	P09	65	20,1
Total	—	**5.098**	**11,2**

El puesto P10 lideró la producción con 759 filetes y un tiempo promedio de 13,0 seg/filete. Los puestos con menores tiempos promedio por filete (P01: 12,9 s; P10: 13,0 s; P08: 13,6 s) son los de mayor rapidez individual de despinado. Nótese que el promedio global del turno es 11,2 s/filete: este valor es inferior al de cualquier puesto individual porque refleja el promedio ponderado total de la línea en momentos de alta actividad, no necesariamente la velocidad sostenida de cada operario a lo largo del turno completo. P10 es el único puesto que combina la mayor producción total (759 filetes) con una velocidad individual competitiva (13,0 s/filete).

4.3.4. Distribución de estados por puesto

La Figura 4.3 muestra la distribución temporal de los tres estados (DESPINANDO, ESPERA, PUESTO VACÍO) para cada puesto durante el turno. Los puestos P03 y P09 presentan barras dominadas casi completamente por estado PUESTO VACÍO (rojo), confirmando que estuvieron sin operario asignado la mayor parte del turno.

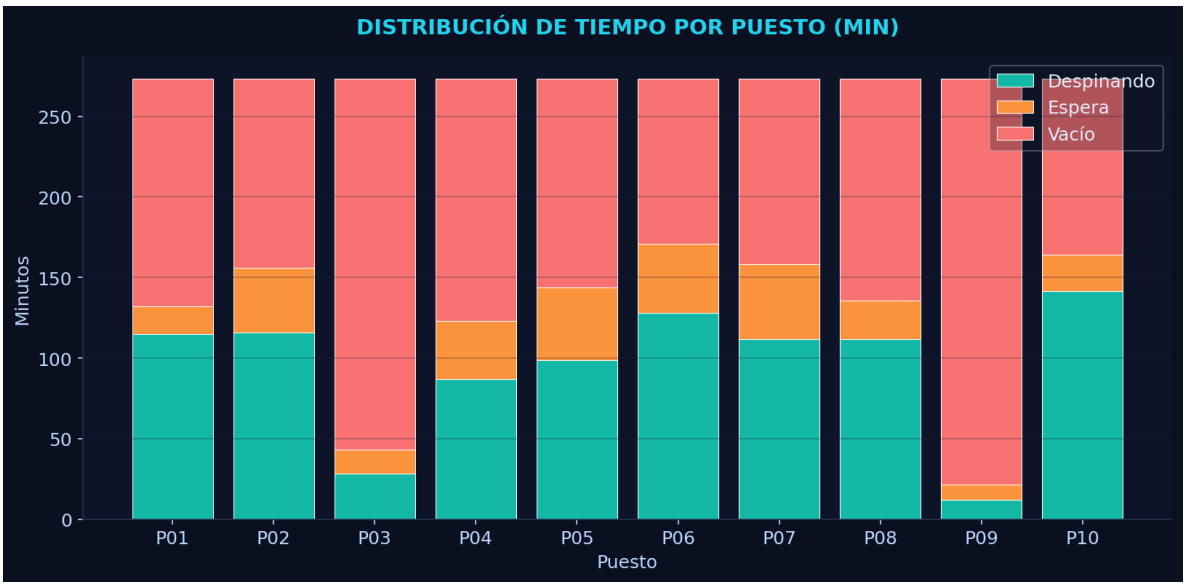


Fig. 4.3: Distribución temporal de estados por puesto durante el turno 2026-03-09 (escala en minutos). Verde: DESPINANDO; naranja: ESPERA; rojo: PUESTO VACÍO. P10 y P06 presentan los mayores tiempos en DESPINANDO; P03 y P09 muestran predominancia de PUESTO VACÍO. Fuente: reporte automático del sistema, elaboración propia.

La Figura 4.4 resume la distribución global de los tres estados, agregando los datos de los 10 puestos durante todo el turno.

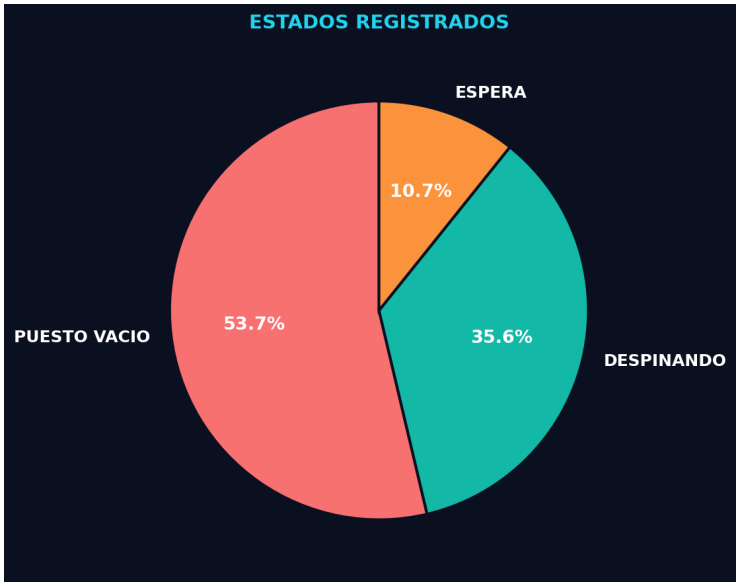


Fig. 4.4: Distribución global de estados registrados durante el turno 2026-03-09. PUESTO VACÍO concentra el 53,7 % del tiempo agregado de los 10 puestos, DESPINANDO el 35,6 % y ESPERA el 10,7 %. Fuente: reporte automático del sistema, elaboración propia.

Es importante interpretar estos porcentajes en contexto. El elevado porcentaje del estado PUESTO VACÍO (53,7 %) no debe leerse como tiempo no productivo de la línea, sino como la consecuencia directa de tres factores: (i) el horario monitoreado abarca el turno completo de 5 horas 10 minutos, lo que incluye los **períodos de colación del personal** y las **labores de limpieza de la línea** ejecutadas al término de la producción; (ii) durante la mayor parte del turno los puestos P03 y P09 no estuvieron asignados a operario alguno (84 % y 92 % en estado vacío respectivamente, según la Tabla 4.7); y (iii) la línea opera con capacidad parcial cuando la demanda de producción no requiere los 10 puestos simultáneamente. Estos elementos son habituales en la operación real de la planta y forman parte natural de la jornada laboral, por lo que el sistema los registra fielmente sin clasificarlos como detenciones de línea.

La Tabla 4.7 presenta los valores exactos por puesto en minutos y porcentaje, complementando la información visual de las figuras anteriores.

Tabla 4.7: Distribución temporal de estados por puesto, turno 2026-03-09 (273,2 min totales por puesto).

Puesto	Despinando (min)	Desp. (%)	Espera (min)	Esp. (%)	Vacío (min)	Vac. (%)
P01	114,9	42 %	17,2	6 %	141,1	52 %
P02	116,1	43 %	40,0	15 %	117,2	43 %
P03	28,3	10 %	15,2	6 %	229,8	84 %
P04	87,2	32 %	35,9	13 %	150,1	55 %
P05	98,8	36 %	44,9	16 %	129,5	47 %
P06	128,1	47 %	42,8	16 %	102,4	37 %
P07	111,7	41 %	46,6	17 %	114,9	42 %
P08	112,0	41 %	23,7	9 %	137,6	50 %
P09	12,2	4 %	9,5	3 %	251,5	92 %
P10	141,6	52 %	22,7	8 %	108,9	40 %
Global	**950,9**	**35 %**	**298,5**	**11 %**	**1.483,0**	**54 %**

4.3.5. Evolución temporal del turno

La Figura 4.5 presenta la evolución de las entradas y salidas acumuladas a la línea de despinado durante todo el turno. La pendiente prácticamente lineal entre las 13:29 y las 16:30 evidencia un flujo de procesamiento constante, mientras que el plateau posterior corresponde al cierre paulatino del flujo de materia prima al final del turno.

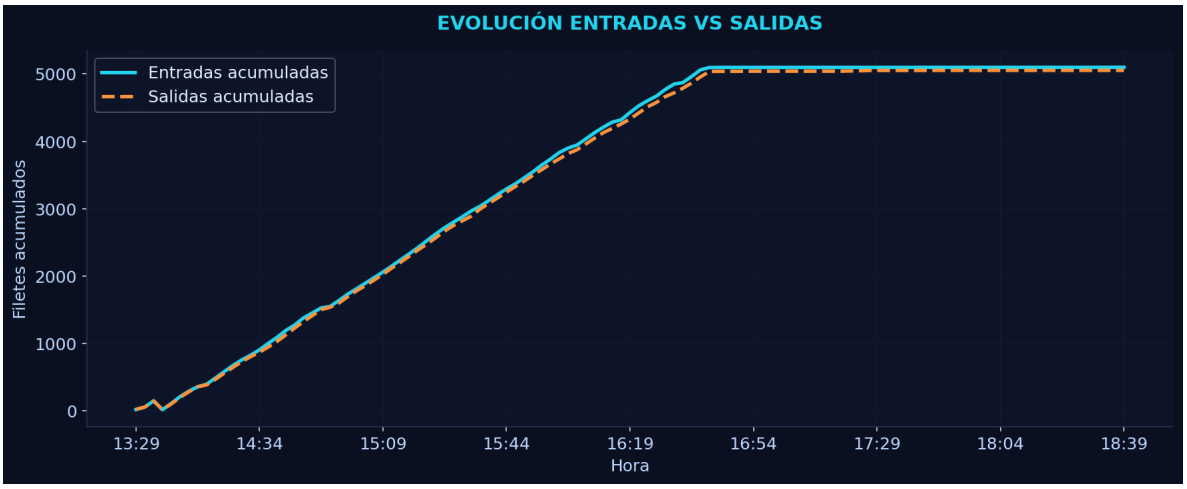


Fig. 4.5: Evolución de las entradas y salidas acumuladas durante el turno 2026-03-09. Curva celeste: entradas a la línea (5.098 totales); curva naranja: salidas (5.052 totales). La diferencia constante de 46 filetes corresponde al material en tránsito en la línea. Fuente: reporte automático del sistema, elaboración propia.

La Figura 4.6 muestra las velocidades instantáneas de entrada y salida (fil/min) a lo largo del turno, métrica que permite identificar variaciones en el ritmo de la línea más allá de los promedios globales.

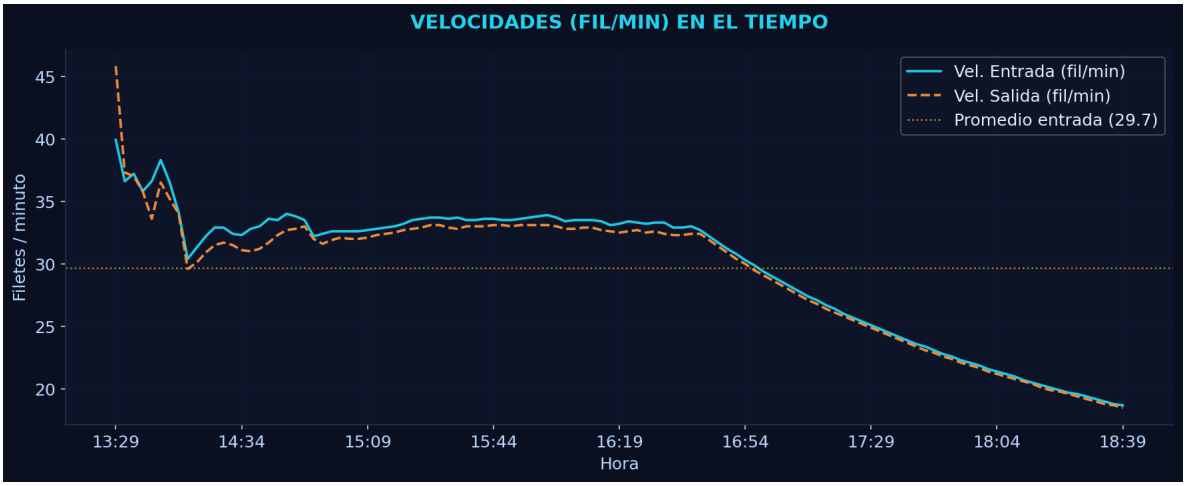


Fig. 4.6: Velocidades instantáneas de entrada y salida (fil/min) durante el turno 2026-03-09. La velocidad se mantuvo estable en torno a 33 fil/min entre las 14:00 y las 16:30, descendiendo gradualmente hacia el final del turno. La línea horizontal indica el promedio global de entrada (29,7 fil/min). Fuente: reporte automático del sistema, elaboración propia.

4.3.6. Detenciones de línea registradas

El sistema registró tres detenciones de línea durante el turno, todas de tipo ENTRADA, detalladas en la Tabla 4.8.

Tabla 4.8: Registro de detenciones de línea, turno 2026-03-09.

N°	**Inicio**	**Fin**	**Tipo**	**Duración**
1	16:01:56	16:02:26	ENTRADA	30 s
2	16:15:26	16:15:56	ENTRADA	30 s
3	16:32:27	16:33:57	ENTRADA	1 min 30 s
Tiempo total en detenciones				**2 min 30 s**

Las tres detenciones ocurrieron en un intervalo de aproximadamente 31 minutos (16:01–16:33), todas asociadas a interrupciones en el flujo de entrada de filetes a la línea. La detención de mayor duración (1 min 30 s) ocurrió a las 16:32. Ninguna detención superó el umbral de 2 minutos, y el tiempo total acumulado de 2 min 30 s sobre un turno de 5 horas representa un impacto mínimo sobre la producción total.

4.4. Resultados del sistema de verificación de EPP

4.4.1. Desempeño comparativo de modelos generativos

La evaluación comparativa de los tres modelos generativos multimodales para la verificación automática de EPP se realizó sobre un conjunto de 50 imágenes de operarios individuales, capturadas en condiciones reales de la planta [1]. Los resultados se presentan en la Tabla 4.9.

Tabla 4.9: Resultados de la evaluación comparativa de modelos generativos para verificación de EPP [1].

Modelo	**Precisión**	**Coherencia**	**Detalle**	**Consistencia**
OpenAI GPT-4.1	100 %	100 %	100 %	100 %
Google Gemini Pro	90 %	92 %	96 %	94 %
LLaVA (*open-source*)	78 %	80 %	86 %	82 %

4.4.2. Análisis de errores por modelo

El modelo OpenAI GPT-4.1 alcanzó un desempeño perfecto en todas las métricas evaluadas, sin errores de clasificación sobre las 50 imágenes analizadas. Google Gemini Pro presentó errores puntuales en la detección de audífonos de protección (4 imágenes) y en la detección de guantes (1 imagen), lo que impactó principalmente su precisión descriptiva (90 %). El modelo LLaVA mostró errores más distribuidos: 4 fallos en la identificación de cofias, 1 fallo en guantes y 1 fallo en mascarilla.

Cabe precisar que la evaluación se realizó sobre un conjunto de 50 imágenes, tamaño que permite establecer tendencias comparativas entre modelos pero que, en el caso de GPT-4.1, no es estadísticamente suficiente para confirmar un desempeño perfecto de manera generalizable. Una evaluación más robusta sobre cientos de imágenes es recomendable antes de un despliegue permanente en producción.

4.4.3. Selección del modelo para despliegue

Si bien GPT-4.1 obtuvo el mejor desempeño, el análisis para el despliegue en producción debe considerar factores económicos y operacionales además de la precisión. La Tabla 4.10 compara los tres modelos desde una perspectiva de viabilidad práctica para un sistema que operaría con cientos o miles de consultas diarias.

Tabla 4.10: Comparación de viabilidad para despliegue en producción del sistema de verificación de EPP.

Criterio	**GPT-4.1**	**Gemini Pro**	**LLaVA**
Precisión global	100 %	90 %	78 %
Costo por consulta	Alto	**Bajo**	Nulo (local)
Dependencia externa	Sí	Sí	No
Latencia API	Alta	**Baja**	Variable
Req. hardware local	No	No	Alto
Control de datos	No	No	Sí
Viabilidad producción	Limitada	**Recomendada**	Alternativa

En virtud de este análisis, se concluye que **Google Gemini Pro es el modelo más adecuado para el despliegue en producción**. Si bien GPT-4.1 supera a Gemini en precisión, su costo por *token* es significativamente mayor, lo que lo hace inviable económicamente para un sistema que procesa múltiples operarios a lo largo de un turno completo. Gemini Pro, en cambio, ofrece un costo por *token* menor, latencias reducidas y un desempeño de 90 % que resulta suficiente para la aplicación de supervisión de EPP en planta. La alternativa LLaVA, ejecutada localmente, eliminaría el costo por consulta pero requeriría hardware dedicado en la planta y presenta el menor desempeño de los tres modelos, lo que lo descarta como opción principal en el contexto de seguridad industrial donde los errores de clasificación tienen consecuencias directas.

4.5. Discusión general de resultados

Los resultados obtenidos demuestran que el sistema de monitoreo desarrollado es técnicamente viable y operacionalmente útil en condiciones reales de planta. Los modelos YOLOv11L entrenados superaron los objetivos de precisión definidos (mAP@0.5 $>$ 0,90) y operan a 103 FPS en el hardware de despliegue, muy por encima de los 30 FPS de la cámara, garantizando que no existe cuello de botella computacional en la captura.

La validación en el turno del 9 de marzo de 2026 permitió registrar 5.098 filetes procesados en 5 horas, con diferencias significativas de productividad entre puestos:

P10 procesó 759 filetes (14,9 % del total) mientras que P09 procesó apenas 65 (1,3 % del total). Esta variabilidad evidencia el valor del sistema para identificar asimetrías operativas que no serían detectables mediante supervisión humana convencional. Las detenciones de línea fueron escasas (3 eventos, 2,5 min acumulados), pero el sistema las registró con precisión de segundo, habilitando su trazabilidad completa para análisis posteriores. El elevado porcentaje agregado de estado PUESTO VACÍO (53,7 %) se explica principalmente por los períodos de colación del personal y las labores de limpieza de línea incluidos dentro de las cinco horas del turno monitoreado, sumados a la operación habitual con capacidad parcial de la línea, no constituyendo en ningún caso un indicador de baja eficiencia productiva.

En cuanto al módulo de EPP, los resultados confirman la factibilidad del enfoque generativo multimodal para la supervisión automática del cumplimiento de equipos de protección. La selección de Gemini Pro como modelo de producción equilibra adecuadamente precisión, costo operacional y simplicidad de integración, sin requerir hardware adicional en la planta.

4.6. Análisis económico y de viabilidad

La viabilidad económica de un sistema de monitoreo basado en visión por computador depende de su estructura de costos de implementación y operación, y de los beneficios que genera en términos de ahorro de recursos. A continuación se presenta un análisis preliminar de costos e indicadores de rentabilidad para el despliegue del sistema desarrollado en Salmones Camanchaca.

4.6.1. Estructura de costos

El análisis distingue entre costos de inversión inicial (*CAPEX*) y costos de operación mensual (*OPEX*).

4.6.1.1. Inversión inicial (CAPEX)

La Tabla 4.11 resume los componentes de inversión requeridos para implementar el sistema en una línea de producción. El equipo de cómputo (notebook con GPU NVIDIA RTX 4050) corresponde a un activo preexistente en la empresa, por lo que su costo de adquisición no se imputa al proyecto.

Tabla 4.11: Inversión inicial (CAPEX) para despliegue del sistema en una línea.

Ítem	**Costo (CLP)**
Cámara IP cenital (1080p, protocolo RTSP) + instalación[a]	$150.000
Notebook con GPU NVIDIA RTX 4050[b]	$0
Roboflow — etiquetado y gestión de datasets[c]	$0
API Google Gemini — desarrollo y pruebas[c]	$0
Total CAPEX	**$150.000**

[a] Adquirida e instalada por Salmones Camanchaca S.A.

[b] Activo preexistente; costo no imputado al proyecto.

[c] Utilizados bajo plan gratuito (*free tier*).

4.6.1.2. Costos de desarrollo

El desarrollo del sistema se ejecutó durante tres meses en modalidad de práctica profesional. La Tabla 4.12 detalla los costos asociados. La beca de práctica profesional corresponde al monto total del período de tres meses.

Tabla 4.12: Costos de desarrollo del sistema (3 meses).

Ítem	**Costo total (CLP)**
Beca de práctica profesional (3 meses)	$650.000
Google Colab Pro — entrenamiento en GPU A100 (3 meses)	$28.500
Herramientas de desarrollo (software)	$0
Total desarrollo	**$678.500**

4.6.1.3. Costos de operación mensual (OPEX)

Una vez desplegado el sistema, los costos operativos son significativamente bajos, como se resume en la Tabla 4.13. El consumo eléctrico de la GPU RTX 4050 en modo de inferencia es aproximadamente 35 W, lo que representa un costo despreciable a la tarifa industrial chilena (≈ $120/kWh). El uso de la API Gemini en producción depende del volumen de verificaciones de EPP realizadas por turno.

Tabla 4.13: Costos de operación mensual estimados en producción.

Ítem	**Costo mensual (CLP)**
Electricidad GPU (35 W × 8 h × 22 días)[a]	$702
API Google Gemini Pro (verificación EPP)[b]	$47.500
Mantenimiento y actualizaciones	$0
Total OPEX mensual	**$48.202**

[a] Tarifa industrial eléctrica chilena (≈ $120/kWh).

[b] Estimado en ≈50 USD/mes según volumen de verificaciones.

4.6.2. Estimación de beneficios

Los principales beneficios económicos del sistema provienen del ahorro en supervisión manual de productividad y en la generación de reportes operativos. Sin el sistema automatizado, estas tareas requieren dedicación diaria de personal de supervisión.

La Tabla 4.14 estima el ahorro mensual considerando que la supervisión manual de la línea de despinado demanda aproximadamente 2 horas diarias de tiempo de supervisor para registro de actividad y elaboración de reportes al cierre del turno, valorizado a la tarifa horaria de supervisión de planta (≈ $8,000 CLP/hora, equivalente a ≈ 8 USD/h).

4.6.3. Indicadores de rentabilidad

Con base en los datos anteriores, la Tabla 4.15 sintetiza los principales indicadores de rentabilidad del proyecto.

Tabla 4.14: Estimación de beneficios mensuales del sistema automatizado.

Fuente de ahorro	**Ahorro mensual (CLP)**
Registro manual de productividad (2 h/día)[a]	$334.400
Elaboración de reportes de turno (1 h/día)[a]	$167.200
Reducción de errores de registro (estimado conservador)	$50.000
Total beneficio mensual estimado	**$551.600**

[a] Valorizado a $8.000 CLP/h × 22 días hábiles/mes.

Tabla 4.15: Indicadores de rentabilidad del sistema de monitoreo.

Indicador	**Valor**
Inversión total (CAPEX + desarrollo)	$828.500 CLP
OPEX mensual en producción	$48.202 CLP
Beneficio mensual estimado	$551.600 CLP
Beneficio neto mensual	$503.398 CLP
Período de recuperación (*payback*)	< 2 meses
ROI al primer año de operación	$> 200\,\%$

Es importante destacar que estos valores corresponden a una estimación conservadora para una sola línea de producción. Salmones Camanchaca opera múltiples líneas simultáneas; el escalamiento del sistema a cinco líneas —propuesto como trabajo futuro en la Sección 5.2— implicaría una inversión marginal adicional (principalmente cámaras adicionales a $\approx$ \$114,000 CLP cada una), con beneficios que escalan proporcionalmente, mejorando sustancialmente los indicadores de rentabilidad.

El análisis de costo-beneficio completo, incluyendo proyecciones a múltiples líneas y el impacto cuantificado en la reducción de tiempos de inactividad no planificada, queda sujeto a evaluación por parte del equipo de gestión de Salmones Camanchaca con datos operativos de largo plazo.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

El presente trabajo desarrolló un sistema de monitoreo basado en inteligencia artificial para la supervisión de productividad y seguridad ocupacional en una línea de despinado de salmón de la planta de procesos de Salmones Camanchaca, Tomé. A partir de los resultados obtenidos, se pueden extraer las siguientes conclusiones, organizadas en correspondencia con los objetivos específicos planteados.

Los objetivos específicos 1 y 2 —diseño del sistema de captura y construcción del *dataset* etiquetado— fueron cumplidos mediante la instalación de una cámara IP cenital con *stream* RTSP y la construcción de un *dataset* de 5.754 imágenes de entrenamiento y 549 de validación en Roboflow, con técnicas de *data augmentation*. Los objetivos 3 y 4 —entrenamiento de los modelos YOLOv11 e implementación de la lógica ROI— fueron satisfechos con un mAP@0.5 global de 0,927, superando el umbral de 0,90, y con el algoritmo de co-detección implementado para clasificar los tres estados por puesto. El objetivo 5 —detección automática de detenciones— fue cumplido con cuatro tipos de detención registrados con precisión de segundo. El objetivo 6 —cálculo de 15 indicadores operativos— fue implementado completamente y validado durante el turno del 9 de marzo de 2026. El objetivo 7 —modelo de cofias con mAP $> 0{,}85$— fue superado con un mAP@0.5 $= 0{,}894$ sobre 9 clases. Los objetivos 8 y 9 —reporte HTML automático y validación en producción— fueron completados con el sistema operando durante un turno completo y generando reportes sin intervención manual. El objetivo

10 —documentación técnica— se entrega a través de los apéndices de esta memoria.

El sistema de monitoreo de productividad desarrollado demostró ser técnicamente viable y operacionalmente útil en condiciones reales de producción. Los dos modelos YOLOv11L entrenados para la detección de operarios y filetes de salmón alcanzaron un mAP@0.5 global de 0,927 y una latencia de inferencia de 9,7 ms por fotograma, equivalente a 103 FPS, lo que supera ampliamente la frecuencia de captura de la cámara instalada (30 FPS) y garantiza que no existe cuello de botella computacional en el despliegue. Estos resultados son comparables con los reportados en trabajos recientes de detección de EPP y monitoreo industrial basados en arquitecturas YOLO [36, 35, 39].

La validación del sistema durante el turno del 9 de marzo de 2026 permitió registrar 5.098 filetes procesados en 5 horas y 10 minutos, con una velocidad promedio de entrada de 29,7 fil/min y un tiempo promedio por filete de 11,2 segundos. El sistema identificó una variabilidad significativa entre puestos, con P10 procesando 759 filetes (14,9 % del total) frente a P09 con apenas 65 filetes (1,3 % del total), diferencias que son prácticamente imposibles de cuantificar con precisión mediante supervisión humana convencional. Se registraron únicamente 3 detenciones de línea, con una duración acumulada de 2 minutos y 30 segundos, lo que equivale a un índice de disponibilidad del 99,2 % durante el turno monitorizado.

La evaluación de la arquitectura SlowFast Networks [50] como alternativa para el reconocimiento de acciones evidenció que, si bien es técnicamente capaz de clasificar actividades con precisiones razonables, su carga computacional combinada (detección + acción + seguimiento) resulta incompatible con los recursos de hardware disponibles para despliegue en la planta. Este resultado justifica la decisión de adoptar el enfoque determinístico basado en co-detección dentro de ROI, que alcanza desempeños superiores con una fracción del costo computacional.

El modelo de detección multiclase de cofias alcanzó un mAP@0.5 de 0,894 sobre 9 clases de rol del personal, validando la viabilidad técnica de identificar automáticamente el cumplimiento del uso de cofia y el rol del operario a partir del color de la misma sin ningún identificador adicional. La evaluación comparativa de modelos generativos multimodales para la verificación de EPP [1] demostró que Google Gemini Pro constituye la alternativa más adecuada para el despliegue en producción, equilibrando un desempeño del 90 % con un costo por *token* significativamente menor al de GPT-4.1,

sin requerir infraestructura local adicional.

En términos globales, el sistema propuesto sienta las bases tecnológicas para una transformación profunda de la supervisión operativa en plantas de procesamiento de salmón, avanzando desde una gestión reactiva basada en observación humana periódica hacia una gestión continua, trazable y basada en datos.

5.2. Trabajo futuro

Los resultados obtenidos abren diversas líneas de trabajo futuro que permitirían expandir el alcance, la robustez y el impacto del sistema desarrollado. A continuación se presentan las más relevantes.

5.2.1. Escalamiento a múltiples líneas de producción

El sistema fue desarrollado y validado sobre una única línea de despinado. Sin embargo, la arquitectura implementada es intrínsecamente escalable: la lógica de análisis por ROI, la clasificación de estados y los 15 indicadores operativos son independientes del número de líneas monitoreadas. Una extensión natural consiste en desplegar el sistema de forma simultánea sobre las cinco líneas de despinado activas en la planta, centralizando los datos en un servidor local que integre los registros CSV de cada línea y genere reportes consolidados por turno. Este escalamiento permitiría comparar el desempeño entre líneas, identificar patrones globales de producción y detectar asimetrías sistemáticas entre grupos de trabajadores.

5.2.2. Extensión a otras áreas de procesamiento

El enfoque basado en visión por computador y ROI no es exclusivo del área de despinado. Otras zonas de la planta, tales como las líneas de corte, fileteado, envasado y despacho, presentan características operativas igualmente susceptibles de monitoreo automático. Cada área requeriría el entrenamiento de modelos específicos adaptados a las clases de objetos y acciones relevantes, pero la infraestructura de captura, análisis por ROI, generación de KPIs y reporte automático sería reutilizable sin modificaciones

sustanciales. A largo plazo, esto permitiría construir un sistema de inteligencia operativa integral para toda la planta de procesos.

5.2.3. Infraestructura de servidor centralizado con múltiples cámaras

El despliegue a escala de planta requiere una arquitectura de servidor centralizado capaz de procesar simultáneamente los *streams* RTSP de múltiples cámaras. Un servidor equipado con GPU de gama media-alta (p. ej., NVIDIA RTX 4090 o equivalente con 24 GB VRAM) podría ejecutar de forma concurrente instancias independientes del *pipeline* de inferencia, una por línea o área, almacenando los datos de todos los turnos en una base de datos estructurada. Esta infraestructura habilitaría análisis históricos longitudinales, comparaciones entre turnos y entre líneas, y la construcción de modelos predictivos de productividad basados en series temporales.

5.2.4. Verificación de EPP en producción con Gemini Pro

El prototipo de verificación de EPP desarrollado puede trasladarse a producción de forma directa aprovechando la API de Google Gemini Pro [65]. En este esquema, el servidor centralizado recibiría los fotogramas de cada cámara, ejecutaría el detector YOLOv11 para obtener los recortes individuales de cada operario y enviaría cada recorte a la API de Gemini Pro con el *prompt* estructurado de verificación. Las respuestas (mascarilla Sí/No, audífonos Sí/No, guantes Sí/No, cofia Sí/No) se almacenarían en la base de datos junto con la marca temporal y el identificador del operario, construyendo un registro trazable y auditable del cumplimiento de EPP por turno. El bajo costo por *token* de Gemini Pro hace que este esquema sea económicamente viable incluso a escala de cientos de consultas por turno [65, 1].

5.2.5. Cámaras PTZ con zoom automático para verificación de EPP

Una mejora significativa para el módulo de EPP consiste en integrar cámaras PTZ (*Pan-Tilt-Zoom*) en la infraestructura de captura. En este esquema, el *pipeline* de detección ejecutaría un ciclo de cuatro pasos por operario detectado: (i) detección de la persona en la imagen cenital de amplio campo; (ii) comando automático de *zoom* y reencuadre de la cámara PTZ hacia el operario identificado; (iii) captura de una secuencia de 4 imágenes consecutivas del operario en alta resolución desde distintos ángulos del ciclo PTZ; y (iv) envío de las cuatro imágenes a la API de Gemini Pro con un *prompt* que solicita la verificación de EPP tomando en cuenta la secuencia completa. Este enfoque de verificación secuencial reduciría significativamente los errores de clasificación asociados a oclusiones parciales, ángulos desfavorables o elementos de EPP fuera del campo de visión, que constituyen la principal fuente de fallos del enfoque con imagen única.

5.2.6. Detección de acciones y conductas de riesgo

A más largo plazo, la arquitectura multimodal basada en modelos generativos abre la posibilidad de extender el análisis más allá del cumplimiento de EPP hacia la detección de acciones y conductas de riesgo en tiempo real. En este contexto, el *prompt* enviado a Gemini Pro junto con la secuencia de imágenes podría solicitar adicionalmente la identificación de situaciones como manipulación incorrecta de herramientas, posturas ergonómicamente inadecuadas, acumulación de material peligroso en zonas de tránsito, o presencia de personal no autorizado en áreas restringidas. La combinación de detección visual (YOLOv11) con razonamiento semántico (modelo generativo multimodal) permitiría construir un sistema de seguridad ocupacional proactivo, capaz de generar alertas en tiempo real antes de que una situación de riesgo derive en un accidente, trascendiendo el enfoque reactivo de la supervisión tradicional [68, 1].

5.3. Reflexión final

El trabajo desarrollado demuestra que la inteligencia artificial aplicada a la visión por computador puede transformar de manera concreta y medible la forma en que se gestionan la productividad y la seguridad en entornos industriales de alta exigencia como las plantas de procesamiento de salmón. Los modelos y sistemas construidos no son prototipos teóricos: fueron diseñados, entrenados y validados en condiciones reales de producción, con datos reales, en una planta real y operando en tiempo real. Este carácter aplicado constituye, en sí mismo, una de las contribuciones más relevantes del trabajo.

Desde una perspectiva de proceso, este trabajo representó un aprendizaje significativo sobre las brechas que existen entre la investigación académica y el despliegue industrial real. La literatura de visión por computador ofrece resultados notables en condiciones controladas, pero trasladar esos resultados a una planta de procesamiento de alimentos —con iluminación variable, vapor de agua, movimiento continuo de personas y la presión de no interrumpir la producción— impone desafíos que no están documentados en los papers. Aprender a lidiar con esa brecha, negociar tiempos de grabación con los supervisores de línea, ajustar parámetros en tiempo real durante el turno y entregar un sistema que los propios operarios pudieran consultar desde el móvil, fue una formación que ningún curso pudo haber entregado de forma equivalente.

Una lección técnica concreta fue la evaluación y descarte de SlowFast Networks: la arquitectura era técnicamente correcta para la tarea, pero incompatible con las restricciones de hardware disponibles para despliegue. Esta decisión de abandonar una solución más sofisticada en favor de una más simple y efectiva —la co-detección por ROI— enseñó que en ingeniería aplicada, la solución óptima no siempre es la más compleja, sino la que mejor se adapta a las restricciones del problema real.

La escalabilidad de la arquitectura propuesta, la modularidad de sus componentes y la accesibilidad de las herramientas empleadas (YOLOv11, Roboflow, Google Colab, Gemini API) hacen que las soluciones desarrolladas sean replicables en otros contextos de la industria acuícola y alimentaria, con adaptaciones menores. El camino hacia una planta de procesos completamente instrumentada con inteligencia artificial para monitoreo continuo de productividad, trazabilidad de operaciones y verificación

automatizada de seguridad ocupacional está, a partir de este trabajo, significativamente más cerca.

Bibliografía

[1] B. L. Durán, "Evaluación automática de EPP en planta usando modelos generativos multimodales," *Proyecto de Investigación, Ingeniería Civil en Telecomunicaciones*, 2025, trabajo no publicado.

[2] SalmonChile, "Discover the main markets of chilean salmon," 2024, exportaciones 2024: USD 6.366 mil millones; Chile como segundo productor mundial (31 % global). [Online]. Available: https://www.salmonchile.cl/blog/markets-chilean-salmon/

[3] USDA Foreign Agricultural Service, "Chile: Salmon overview," 2022. [Online]. Available: https://www.fas.usda.gov/data/chile-salmon-overview

[4] SeafoodSource, "Obstacles both domestic and abroad take toll on Chile's 2024 salmon exports," 2025, 782.076 toneladas exportadas en 2024 por USD 6.37 mil millones. [Online]. Available: https://www.seafoodsource.com/news/supply-trade/obstacles-both-domestic-and-abroad-take-toll-on-chile-s-2024-salmon-exports

[5] UNCTAD, "A case study of the salmon industry in Chile," *United Nations Conference on Trade and Development*, 2006. [Online]. Available: https://unctad.org/system/files/official-document/iteiit200512_en.pdf

[6] British Retail Consortium, "BRC global standard for food safety – issue 9," BRC Trading Ltd., London, UK, Industry Standard, 2022.

[7] International Featured Standards, "IFS food standard – version 7," IFS Management GmbH, Berlin, Germany, Industry Standard, 2020.

[8] International Organization for Standardization, "ISO 22000:2018 food safety management systems – requirements for any organization in the food chain," ISO,

Geneva, Switzerland, International Standard, 2018.

[9] GLOBALG.A.P., "GLOBALG.A.P. aquaculture standard v6.0," GLOBALG.A.P. c/o FoodPLUS GmbH, Cologne, Germany, Industry Standard, 2022.

[10] H. Einarsdottir, B. Gudmundsson, and V. Omarsson, "Automation in the fish industry," *Animal Frontiers*, vol. 12, no. 1, pp. 32–39, 2022.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, pp. 1097–1105, 2012.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587.

[16] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[17] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016, pp. 21–37.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[20] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7263–7271.

[21] ——, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[22] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[23] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolo," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[24] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[25] R. Khanam and M. Hussain, "Yolov11: An overview of the key architectural enhancements," *arXiv preprint arXiv:2410.17725*, 2024.

[26] A. Vijayakumar and S. Vasu, "Yolo-based object detection models: A review and its applications," *Multimedia Tools and Applications*, vol. 83, pp. 83 535–83 574, 2024.

[27] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[28] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014, pp. 740–755.

[29] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237–242.

[30] Wang and Others, "YOLO V3 + VGG16-based automatic operations monitoring and analysis in a manufacturing workshop under industry 4.0," *Journal of Manufacturing Systems*, vol. 62, pp. 325–340, 2022, reconocimiento de acciones de operarios en fabricas con YOLOv3 + VGG16.

[31] Soltanzadeh and Mohammadfam, "An automated recognition of work activity in industrial manufacturing using convolutional neural networks," *Electronics*, vol. 10, no. 23, p. 2946, 2021, yOLOv3 + R-CNN para reconocer 16 actividades industriales.

[32] Tian and Others, "Real-time tool detection in smart manufacturing using You-Only-Look-Once (YOLO)v5," *Manufacturing Letters*, vol. 35, pp. 1192–1199, 2023.

[33] M. U. Khan and M. Iqbal, "YOLO based human action recognition and localization," *Procedia Computer Science*, vol. 133, pp. 831–838, 2018.

[34] V. S. K. Delhi, R. Sankarlal, and A. Anand, "Detection of personal protective equipment (PPE) compliance on construction site using computer vision based deep learning techniques," *Frontiers in Built Environment*, vol. 6, p. 136, 2020.

[35] M. Ferdous and S. M. M. Ahsan, "PPE detector: a YOLO-based architecture to detect personal protective equipment (PPE) for construction sites," *PeerJ Computer Science*, vol. 8, p. e999, 2022, dataset CHVG con 8 clases, alcanzo mAP@0.5 de 89.84 % con YOLOX-m.

[36] Wang and Others, "Personal protective equipment detection using YOLOv8 architecture on object detection benchmark datasets: a comparative study," *Cogent Engineering*, vol. 11, no. 1, p. 2333209, 2024, reporta mAP@0.5 de hasta 92 % para detección de cascos, chalecos y guantes con YOLOv8x.

[37] V. H. S. Pham, L. A. Tran, B. D. Khoa, and Q. T. Nguyen, "Enhancing worker safety: Real-time automated detection of personal protective equipment to prevent falls from heights at construction sites using improved YOLOv8 and edge devices," *Journal of Construction Engineering and Management*, vol. 151, no. 1, 2025, yOLOv8 mejorado en Jetson Xavier NX, mAP@0.5=92.52 % a 9.11 FPS.

[38] Hayat and Others, "Personal protective equipment detection: A deep-learning-based sustainable approach," *Sustainability*, vol. 15, no. 18, p. 13990, 2023, precision 97.64 %, recall 93.11 %, 7.96 FPS para deteccion de PPE.

[39] Liu and Others, "SD-YOLOv5: a rapid detection method for personal protective equipment on construction sites," *Frontiers in Built Environment*, vol. 11, p. 1563483, 2025, dataset CHV, AP=93.7 % con YOLOv5s mejorado.

[40] Park and Others, "Automated non-PPE detection on construction sites using YOLOv10 and transformer architectures for surveillance and body worn cameras with benchmark datasets," *Scientific Reports*, vol. 15, p. 12468, 2025, aP50 promedio 87.32 % para deteccion de cascos, mascarillas, chalecos, guantes y zapatos.

[41] Karthik and Others, "Real-time safety detection on construction sites using a vision-language and NLP-based model," *Advanced Engineering Informatics*, 2025, yOLOv11-s para deteccion de PPE, mAP@0.5=88.5 %.

[42] C. Zhang, Y. Zhao, W. Yang, L. Gao, W. Zhang, Y. Liu, X. Zhang, and H. Wang, "Computer vision-based deep learning modeling for salmon part segmentation and defect identification," *Foods*, vol. 14, no. 20, p. 3529, 2025, u-Net mejorado con CBAM + YOLOv5 para corte de salmon, mAP=96.87 %, mIoU=94.33 %.

[43] L. Zhang, J. Wang, and Q. Duan, "Automatic fish population counting by machine vision and a hybrid deep neural network model," *Animals*, vol. 10, no. 2, p. 364, 2020, conteo automatico de salmones atlanticos con precision 95.06 %.

[44] Y. Jang and Y.-S. Seo, "Machine vision based fish cutting point prediction for target weight," *Computers, Materials and Continua*, vol. 75, no. 1, pp. 2247–2263, 2023.

[45] J. Jareno, G. Barcena-Gonzalez, J. Castro-Gutierrez, R. Cabrera-Castro, and P. L. Galindo, "Enhancing fish auction with deep learning and computer vision: Automated caliber and species classification," *Fishes*, vol. 9, no. 4, p. 133, 2024.

[46] M. Hamzaoui, M. O.-E. Aoueileyine, L. Romdhani, and R. Bouallegue, "An improved deep learning model for underwater species recognition in aquaculture," *Fishes*, vol. 8, no. 10, p. 514, 2023.

[47] M. T. Islam, T. S. Ripa, S. A. Udoy, and M. Jahan, "ROI-Aware multi-object tracking with YOLOv6 and SORT for intelligent video surveillance," *SSRN Electronic Journal*, 2025, yOLOv6 con modulo ROI alcanza mAP=92.3 % para vigilancia inteligente.

[48] S. R. Alvar and I. V. Bajic, "MV-YOLO: Motion vector-aided tracking by semantic object detection," *arXiv preprint arXiv:1805.00107*, 2018.

[49] Kumar and Others, "Object detection in real-time video surveillance using attention based transformer-YOLOv8 model," *Alexandria Engineering Journal*, vol. 99, pp. 1–14, 2025, precision 96.78 %, recall 96.89 %, mAP=89.67 %.

[50] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "SlowFast networks for video recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6202–6211.

[51] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and large-scale datasets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6299–6308.

[52] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool, "Temporal segment networks: Towards good practices for deep action recognition," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016, pp. 20–36.

[53] Roboflow, "Roboflow: Computer vision tools for developers," 2024. [Online]. Available: https://roboflow.com

[54] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 8024–8035.

[55] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, 2000.

[56] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, and J. Zou, "Gradio: Hassle-free sharing and testing of ML models in the wild," 2019, arXiv:1906.02569. [Online]. Available: https://www.gradio.app

[57] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed. Springer, 2022.

[58] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 2nd ed. Prentice Hall, 2011.

[59] NVIDIA Corporation, "CUDA toolkit documentation," 2024. [Online]. Available: https://docs.nvidia.com/cuda/

[60] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[61] Google LLC, "Google colaboratory: A free jupyter notebook environment," 2024, plataforma de *notebooks* Jupyter con soporte GPU en la nube. [Online]. Available: https://colab.research.google.com

[62] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[63] A. Kadiyala, O. Mermer, D. J. Samuel, Y. Sermet, and I. Demir, "A comparative study of GPT-4 vision, Gemini, LLaVA, and Multimodal-GPT," *Sensors*, vol. 24, no. 9, 2024.

[64] OpenAI, "GPT-4o system card," 2024. [Online]. Available: https://openai.com/index/gpt-4o-system-card/

[65] Google DeepMind, "Gemini: A family of highly capable multimodal models," 2024. [Online]. Available: https://deepmind.google/technologies/gemini/

[66] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "LLaVA: Large language and vision assistant," 2023. [Online]. Available: https://llava-vl.github.io

[67] Y. Zhang *et al.*, "Salmon fillet segmentation and quality inspection using U-Net and YOLOv5," *Computers and Electronics in Agriculture*, vol. 210, 2025.

[68] Z. Chen *et al.*, "Vision-language model for interpretable and fine-grained detection of safety compliance in diverse workplaces," *arXiv preprint arXiv:2408.07146*, 2024.

Anexos

Apéndice A

Anexo A: Parámetros de configuración del sistema

A.1. Polígonos ROI de mesas (zonas de cinta transportadora)

El archivo `mesas.json` define las regiones de interés correspondientes a las zonas de la cinta transportadora frente a cada puesto de trabajo. Estas regiones se utilizan para detectar la presencia de filetes de salmón en la zona de procesamiento activa de cada operario.

```
[
  {
    "id_puesto": 1,
    "nombre": "Mesa␣1",
    "coordenadas": [[86,51],[81,79],[101,83],[104,55],[104,52]]
  },
  {
    "id_puesto": 2,
    "nombre": "Mesa␣2",
    "coordenadas": [[79,94],[78,114],[77,118],[82,122],
                    [93,121],[98,91],[98,87],[91,86],
                    [86,86],[82,88]]
  }
  // ... (10 mesas en total)
```

```
]
```

Listing A.1: Extracto del archivo `mesas.json` con las coordenadas de las regiones ROI de la cinta transportadora (Mesa 1 y Mesa 2).

A.2. Polígonos ROI de puestos (zonas de trabajo de operarios)

El archivo `puestos.json` define las regiones de interés correspondientes a las zonas de trabajo individuales de cada operario. Estas regiones, más amplias que las de mesas, permiten detectar la presencia del operario en su puesto y atribuirle el estado correspondiente.

```
[
  {
    "id_puesto": 1,
    "nombre": "Puesto␣1",
    "coordenadas": [[103,44],[66,44],[61,78],[60,84],[99,84]]
  },
  {
    "id_puesto": 10,
    "nombre": "Puesto␣10",
    "coordenadas": [[175,49],[176,80],[192,80],[204,82],
                    [212,82],[221,82],[223,72],[223,59],
                    [225,42],[223,25],[218,22],[198,28],
                    [192,27],[187,38]]
  }
  // ... (10 puestos en total)
]
```

Listing A.2: Extracto del archivo `puestos.json` con las coordenadas de los polígonos de puesto (Puesto 1 y Puesto 10).

A.3. Script de arranque del sistema

El archivo `run_alimentacion.bat` permite iniciar el pipeline principal con un doble clic en Windows, configurando el directorio de trabajo y el intérprete de Python correctos.

```
@echo off
TITLE Monitoreo Planta
cd /d "d:\Downloads\IA_SALMONESCAMANCHACA"
cls
echo Iniciando ALIMENTACIONGRADIO.py...
"C:\Users\...\Python311\python.exe" "ALIMENTACIONGRADIO.py"
pause
```

Listing A.3: Script de arranque `run_alimentacion.bat` para ejecución directa del pipeline en Windows.

Apéndice B

Anexo B: Código fuente del sistema de monitoreo

B.1. Pipeline principal de inferencia en tiempo real (`ALIMENTACIONGRADIO.py`)

El pipeline principal implementa la lógica completa de captura RTSP, inferencia dual con dos modelos YOLOv11, análisis por ROI, clasificación de estados, cálculo de indicadores operativos y escritura de datos. A continuación se presentan las secciones más relevantes del código.

B.1.1. Configuración y parámetros del sistema

```
# Input: Camara RTSP
RTSP_URL = "rtsp://admin:Nimda.12@10.8.9.13:554/Streaming/Channels/102
    "

# Modelos / poligonos
MODEL_SALMON_PATH    = os.path.join(BASE_DIR, "salmonestiemporeal.pt")
MODEL_OPERARIOS_PATH = os.path.join(BASE_DIR, "operariostiemporeal.pt"
    )
PUESTOS_JSON         = os.path.join(BASE_DIR, "puestos_backup.json")
MESAS_JSON           = os.path.join(BASE_DIR, "mesas.json")
```

```
9
10 # Outputs para Gradio
11 ESTADO_JSON_OUT    = os.path.join(BASE_DIR, "estado_planta.json")
12 CSV_DIR            = os.path.join(BASE_DIR, "csv_logs")
13 WRITE_JSON_EVERY_S = 1.0
14 WRITE_CSV_EVERY_S  = 30.0
15
16 # ROI / lineas de conteo
17 ROI_X, ROI_Y, ROI_W, ROI_H = 206, 0, 280, 357
18 LINEA_ENTRADA = 31
19 LINEA_SALIDA  = 327
20
21 # Umbrales de confianza
22 CONF_TH_SALMON    = 0.20
23 CONF_TH_OPERARIOS = 0.50
24
25 # Detenciones: umbral en segundos
26 UMBRAL_DETENCION_S = 40.0
27
28 # Suavizado de presencia: ciclos consecutivos para confirmar
29 OPER_PRESENCIA_MIN_CICLOS = 3
30
31 # Tasas de inferencia intercalada
32 TARGET_FPS_SALMON    = 10
33 TARGET_FPS_OPERARIOS = 5
```

Listing B.1: Configuración global del pipeline: rutas, modelos, umbrales y tasas de inferencia.

B.1.2. Funciones utilitarias: ROI y escritura atómica

```
1 def atomic_write_json(path: str, payload: dict):
2     """Escritura␣atomica␣de␣JSON␣(evita␣leer␣archivo␣corrupto)."""
3     tmp = path + ".tmp"
4     with open(tmp, "w", encoding="utf-8") as f:
5         json.dump(payload, f, ensure_ascii=False, indent=2)
6     for _ in range(3):
7         try:
8             os.replace(tmp, path)
9             return
```

```
        except PermissionError:
            time.sleep(0.05)

def cargar_poligonos(json_path: str):
    """Carga poligonos ROI desde JSON y los convierte a arrays numpy."""
    with open(json_path, "r", encoding="utf-8") as f:
        data = json.load(f)
    poligonos = {}
    for item in data:
        pid    = str(item["id_puesto"])
        coords = np.array(item["coordenadas"], dtype=np.int32)
        poligonos[pid] = coords
    return poligonos

def punto_en_poligono(punto, poligono):
    """Retorna True si el punto cae dentro del poligono (ray casting)."""
    return cv2.pointPolygonTest(poligono, punto, False) >= 0
```

Listing B.2: Funciones para carga de polígonos ROI, detección de punto en polígono y escritura atómica de JSON.

B.1.3. Lógica de clasificación de estados por puesto

```
for pid, poly_puesto in puestos_poly.items():
    poly_mesa = mesas_poly.get(pid)

    # Suavizado de presencia: incrementar/decrementar contador
    if pid in puestos_con_operario:
        presencia_contador[pid] = min(
            presencia_contador[pid] + 1, OPER_PRESENCIA_MIN_CICLOS + 2)
    else:
        presencia_contador[pid] = max(presencia_contador[pid] - 1, 0)

    operario_presente = (presencia_contador[pid] >= OPER_PRESENCIA_MIN_CICLOS)

    # Detectar salmon en la zona de mesa del puesto
```

```
    salmon_presente = False
    if poly_mesa is not None:
        for sid, scenter in last_salmon_ids_map.items():
            if punto_en_poligono(scenter, poly_mesa):
                salmon_presente = True
                break

    # Clasificacion deterministica de estado
    if operario_presente:
        if salmon_presente:
            nuevo_estado = "DESPINANDO"
            stats_puestos[pid]["t_despinando"] += dt
        else:
            nuevo_estado = "ESPERA"
            stats_puestos[pid]["t_espera"] += dt
    else:
        if tiempo_inicio_vacio_logico[pid] is None:
            tiempo_inicio_vacio_logico[pid] = now
        dur_vacio = now - tiempo_inicio_vacio_logico[pid]
        if dur_vacio < 5.0:    # gracia de 5s antes de marcar VACIO
            nuevo_estado = estado_actual_puesto[pid]
        else:
            nuevo_estado = "PUESTO␣VACIO"
            stats_puestos[pid]["t_vacio"] += dt

    estado_actual_puesto[pid] = nuevo_estado
```

Listing B.3: Clasificación del estado de cada puesto mediante co-detección en ROI, con suavizado de presencia y período de gracia temporal.

B.1.4. Cálculo de KPIs y escritura del payload JSON

```
# KPIs globales de linea
elapsed_min      = max((now - t0) / 60.0, 0.0001)
rate_entrada_min = entradas / elapsed_min
rate_salida_min  = salidas  / elapsed_min

total_time_desp_all = sum(s["t_despinando"] for s in stats_puestos.
    values())

```

```
# Filetes proporcionales por puesto (segun tiempo despinando)
for pid in puestos_poly:
    st = stats_puestos[pid]
    if total_time_desp_all > 0:
        proporcion        = st["t_despinando"] / total_time_desp_all
        filetes_estimados = round(entradas * proporcion)
    else:
        filetes_estimados = 0
    avg_time = (st["t_despinando"] / filetes_estimados) \
               if filetes_estimados > 0 else 0.0

    puestos_output_data[pid] = {
        "estado":              estado_actual_puesto[pid],
        "filetes_procesados":  filetes_estimados,
        "tiempo_despinando_m": round(st["t_despinando"] / 60, 2),
        "tiempo_espera_m":     round(st["t_espera"]     / 60, 2),
        "tiempo_vacio_m":      round(st["t_vacio"]      / 60, 2),
        "avg_seg_por_filete":  round(avg_time, 2)
    }

payload = {
    "timestamp":           now_str(),
    "estado_linea":        estado_linea,
    "entradas":            entradas,
    "salidas":             salidas,
    "rate_entrada_min":    round(rate_entrada_min, 2),
    "rate_salida_min":     round(rate_salida_min,  2),
    "global_avg_desp_sec": round(global_avg_desp_sec, 2),
    "detenciones": {
        "general": {"cantidad": det_stats["GENERAL"]["cantidad"],
                    "tiempo_s": round(det_stats["GENERAL"]["tiempo"],
                        1)},
        "entrada": {"cantidad": det_stats["ENTRADA"]["cantidad"],
                    "tiempo_s": round(det_stats["ENTRADA"]["tiempo"],
                        1)},
        "salida":  {"cantidad": det_stats["SALIDA"]["cantidad"],
                    "tiempo_s": round(det_stats["SALIDA"]["tiempo"],
                        1)},
        "total_tiempo_s":   round(total_det_sec, 1)
    },
    "puestos": puestos_output_data
```

```
}

# Escritura periodica al disco
if (now - last_json_write) >= WRITE_JSON_EVERY_S:
    atomic_write_json(ESTADO_JSON_OUT, payload)
    last_json_write = now
if ENABLE_CSV and (now - last_csv_write) >= WRITE_CSV_EVERY_S:
    append_csv(payload, list(puestos_poly.keys()))
    last_csv_write = now
```

Listing B.4: Cálculo de los 15 indicadores operativos y construcción del payload JSON para el dashboard en tiempo real.

B.2. Generador de reportes diarios HTML (REPORTE_DIARIO.py)

```
def generar_reporte(fecha: str):
    csv_puestos = os.path.join(CSV_DIR, f"registro_puestos_{fecha}.csv
        ")
    csv_linea   = os.path.join(CSV_DIR, f"registro_linea_{fecha}.csv")

    df_p = pd.read_csv(csv_puestos, parse_dates=["timestamp"])
    df_l = pd.read_csv(csv_linea,   parse_dates=["timestamp"])

    # Filtrar filas vacias (reinicios del pipeline donde todo es 0)
    df_p = df_p[~((df_p["filetes_procesados"]   == 0) &
                  (df_p["tiempo_despinando_min"] == 0) &
                  (df_p["tiempo_espera_min"]     == 0) &
                  (df_p["tiempo_vacio_min"]      == 0))]
    df_l = df_l[df_l["entradas"] > 0].copy()

    # Ultimo snapshot acumulado por puesto (datos finales del turno)
    ultimo = df_p.groupby("puesto_id").last().reset_index()

    # KPIs globales
    total_filetes    = int(ultimo["filetes_procesados"].sum())
    avg_seg_filete   = round(ultimo["avg_seg_por_filete"].mean(), 1)
    total_despinando = round(ultimo["tiempo_despinando_min"].sum(), 1)
```

```
22
23      if not df_l.empty:
24          last_linea      = df_l.iloc[-1]
25          total_entradas  = int(last_linea["entradas"])
26          total_salidas   = int(last_linea["salidas"])
27          avg_rate_in     = round(df_l["rate_in_min"].mean(), 1)
28          avg_rate_out    = round(df_l["rate_out_min"].mean(), 1)
29          det_total       = round(float(last_linea.get("det_total_s", 0))
                / 60, 1)
30
31      rango_hora = (f"{df_p['timestamp'].min().strftime('%H:%M')} - "
32                    f"{df_p['timestamp'].max().strftime('%H:%M')}")
33
34      # Generar y guardar archivo HTML
35      out_path = os.path.join(BASE_DIR, f"reporte_{fecha}.html")
36      with open(out_path, "w", encoding="utf-8") as f:
37          f.write(build_html(fecha, rango_hora, total_entradas,
38                             total_salidas, avg_rate_in, avg_rate_out,
39                             avg_seg_filete, total_despinando, det_total
                                 ,
40                             ultimo, df_l))
41      print(f"Reporte generado: {out_path}")
42
43  if __name__ == "__main__":
44      fecha = sys.argv[1] if len(sys.argv) > 1 \
45              else datetime.now().strftime("%Y-%m-%d")
46      generar_reporte(fecha)
```

Listing B.5: Función principal del generador de reportes: carga de CSVs, cálculo de KPIs finales del turno y construcción del reporte HTML.

Apéndice C

Anexo C: Entrenamiento de modelos y pipelines EPP

C.1. Entrenamiento del modelo YOLOv11 (operarios y filetes)

El entrenamiento de los modelos YOLOv11L se realizó en Google Colab con GPU NVIDIA Tesla A100 (80 GB VRAM). El flujo comprende la descarga del dataset desde Roboflow y la ejecución del comando de entrenamiento mediante la CLI de Ultralytics.

C.1.1. Descarga del dataset y entrenamiento

```
# Instalacion de dependencias en Google Colab
!pip install -U ultralytics
!pip install roboflow

from roboflow import Roboflow

rf      = Roboflow(api_key="<API_KEY>")
project = rf.workspace("procesamiento-de-peces").project("
   personasoperarios")
version = project.version(1)
dataset = version.download("yolov11")
```

```

# Entrenamiento YOLOv11L
!yolo detect train \
  model='/content/detectoroperarios.pt'                \
  data='/content/personasoperarios-1/data.yaml'        \
  imgsz=640       epochs=100      batch=32             \
  device=0        workers=8       cache=True            \
  amp=True        optimizer='SGD' lr0=0.001             \
  close_mosaic=10 patience=10     save=True             \
  save_period=10                                        \
  project='/content/drive/MyDrive/entrenamientos_salmon' \
  name='OPERARIOS_A100'
```

Listing C.1: Descarga del dataset etiquetado desde Roboflow y entrenamiento del modelo YOLOv11L para detección de operarios.

C.1.2. Reanudación desde checkpoint

```
!yolo detect train \
  model=/content/drive/MyDrive/entrenamientos_salmon/ \
        SALMON_A100_ROBOFLOW/weights/epoch30.pt       \
  data=/content/drive/MyDrive/dataset_salmon/data.yaml \
  resume=True
```

Listing C.2: Reanudación del entrenamiento desde un checkpoint intermedio (época 30).

C.2. Entrenamiento del modelo SlowFast Networks

El entrenamiento de SlowFast Networks para reconocimiento de acciones se realizó con la librería PyTorchVideo. El modelo base `slowfast_r50` fue ajustado para las cinco clases de acción definidas en la línea de despinado.

C.2.1. Dataset y transformaciones

```
import torch
from torch.utils.data import Dataset
```

```
from PIL import Image
import torchvision.transforms as T
import numpy as np, os

NUM_FRAMES_FAST = 32    # frames por clip (camino Fast)
IMG_SIZE        = 224   # tamano espacial

frame_transform = T.Compose([
    T.Resize((IMG_SIZE, IMG_SIZE)),
    T.ToTensor(),
    T.Normalize(mean=[0.45, 0.45, 0.45],
                std =[0.225, 0.225, 0.225]),
])

class FramesDataset(Dataset):
    def __init__(self, csv_path, root):
        self.root    = root
        self.samples = []
        with open(csv_path) as f:
            for line in f:
                parts = line.strip().split()
                if parts:
                    self.samples.append((parts[0], int(parts[1])))

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        folder, label = self.samples[idx]
        frames_dir    = os.path.join(self.root, folder)
        frame_files   = sorted(os.listdir(frames_dir))

        # Submuestreo uniforme a NUM_FRAMES_FAST frames
        idxs   = np.linspace(0, len(frame_files)-1,
                             NUM_FRAMES_FAST, dtype=int)
        frames = [frame_transform(
                    Image.open(os.path.join(frames_dir, frame_files[i
                        ]))
                          .convert("RGB"))
                  for i in idxs]

```

```
        video = torch.stack(frames, dim=1)  # [C, T, H, W]
        return video, label
```

Listing C.3: Clase `FramesDataset` con transformaciones ImageNet y submuestreo uniforme de frames para SlowFast.

C.2.2. Definición del modelo y ciclo de fine-tuning

```
!pip install "git+https://github.com/facebookresearch/pytorchvideo.git
    "

from pytorchvideo.models.hub import slowfast_r50
import torch.nn as nn, torch.optim as optim

device      = "cuda" if torch.cuda.is_available() else "cpu"
num_classes = len(label_map)   # 5: despinando, espera, limpieza,
                               #    conversacion, otros

# Adaptar clasificador al numero de clases
model = slowfast_r50(pretrained=False)
model.blocks[-1].proj = nn.Linear(
    model.blocks[-1].proj.in_features, num_classes)
model = model.to(device)

ALPHA     = 4       # razon temporal Slow/Fast
EPOCHS    = 50
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)

for epoch in range(EPOCHS):
    model.train()
    total_loss = 0.0
    for video, labels in train_loader:
        video, labels = video.to(device), labels.to(device)

        fast_path = video                  # [B, C, T,       H, W]
        slow_path = video[:, :, ::ALPHA]  # [B, C, T/ALPHA, H, W]

        loss = criterion(model([slow_path, fast_path]), labels)
        optimizer.zero_grad()
```

```
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    print(f"Epoch {epoch+1}/{EPOCHS} | "
          f"Loss: {total_loss/len(train_loader):.4f}")

torch.save(model.state_dict(), "slowfast_accion_espera.pth")
```

Listing C.4: Carga del modelo SlowFast R50, adaptación de la capa de clasificación y ciclo de entrenamiento con rutas Slow y Fast.

C.2.3. Evaluación del modelo SlowFast

```
model.eval()
correct, total = 0, 0

with torch.inference_mode():
    for video, labels in val_loader:
        video, labels = video.to(device), labels.to(device)
        fast_path     = video
        slow_path     = video[:, :, ::ALPHA]

        preds   = model([slow_path, fast_path]).argmax(dim=1)
        total   += labels.size(0)
        correct += (preds == labels).sum().item()

print(f"Accuracy: {correct/total:.3f}")
```

Listing C.5: Evaluación de la exactitud del modelo SlowFast sobre el conjunto de validación.

C.3. Configuración del tracker ByteTrack

El archivo `bytetrack.yaml` define los parámetros del algoritmo de seguimiento multi-objeto ByteTrack, utilizado en los pipelines de EPP para mantener identidades persistentes de los operarios a lo largo del video.

```yaml
tracker_type:      bytetrack
track_high_thresh: 0.55   # confianza minima para detecciones primarias
track_low_thresh:  0.15   # confianza minima para recuperar trayectorias
new_track_thresh:  0.55   # confianza para crear nueva trayectoria
track_buffer:      120    # frames que se retiene trayectoria sin deteccion
match_thresh:      0.9    # umbral de asociacion por IoU
fuse_score:        true   # fusionar confianza de deteccion con IoU
min_box_area:      40     # area minima de bounding box (pixeles^2)
frame_rate:        30
mot20:             false
```

Listing C.6: Parámetros del tracker ByteTrack (`bytetrack.yaml`).

C.4. Pipeline de verificación de EPP con OpenAI GPT-4.1

El pipeline de OpenAI integra el modelo YOLOv11 de cofias con ByteTrack y la API de GPT-4.1, verificando periódicamente el uso de EPP de cada operario y generando una bitácora trazable.

C.4.1. Configuración y prompt estructurado

```python
VIDEO_PATH       = ""           # ruta al video de entrada
YOLO_MODEL_PATH  = "best.pt"   # modelo YOLOv11 de cofias
OUTPUT_EPP_CSV   = "bitacora_epp.csv"
OPENAI_API_KEY   = ""           # clave de API
EPP_INTERVAL_MIN = 1            # intervalo de evaluacion (minutos)

CLASS_COLORS = {
    "operador cofia azul":   (255,   0,   0),
    "operador cofia blanca": (255, 255, 255),
    "operador cofia morada": (255,   0, 255),
    "operador cofia naranja":(  0, 140, 255),
```

```
    "supervisor":             (  0, 255, 255),
    "no identificado":        (128, 128, 128),
}

PROMPT_EPP = """
Analiza la imagen del operario y evalua el uso de EPP.
Responde EXACTAMENTE en este formato:
Mascarilla: si/no
Audifonos: si/no
Guantes: si/no
Cofia: si/no
Comentario:
""".strip()
```

Listing C.7: Configuración del pipeline OpenAI: rutas, colores por clase de cofia y prompt estructurado de verificación EPP.

C.4.2. Función de evaluación con GPT-4.1 y bucle principal

```
from openai import OpenAI
import base64

client = OpenAI(api_key=OPENAI_API_KEY)

def evaluar_epp_openai(crop_bgr):
    """Envia recorte de operario a GPT-4.1 y retorna evaluacion EPP."""
    _, buffer = cv2.imencode(".jpg",
                    cv2.cvtColor(crop_bgr, cv2.COLOR_BGR2RGB))
    img_b64 = base64.b64encode(buffer).decode("utf-8")

    response = client.responses.create(
        model="gpt-4.1",
        input=[{"role": "user", "content": [
            {"type": "input_text",  "text":         PROMPT_EPP},
            {"type": "input_image", "image_base64": img_b64}
        ]}]
    )
    return response.output_text

```

```
# Bucle: YOLO + ByteTrack + evaluacion periodica
results      = yolo_model.track(source=VIDEO_PATH, stream=True,
                                tracker="bytetrack.yaml", verbose=False
                                   )
frames_eval = int(fps * 60 * EPP_INTERVAL_MIN)
frame_idx   = 0

for r in results:
    frame = r.orig_img
    if r.boxes is None or r.boxes.id is None:
        frame_idx += 1
        continue

    boxes   = r.boxes.xyxy.cpu().numpy().astype(int)
    cls_ids = r.boxes.cls.cpu().numpy().astype(int)
    bt_ids  = r.boxes.id.cpu().numpy().astype(int)

    for i in range(len(boxes)):
        x1, y1, x2, y2 = boxes[i]
        bt_id  = bt_ids[i]
        clase  = yolo_model.names[cls_ids[i]]

        if bt_id not in mapa_ids:
            mapa_ids[bt_id] = len(mapa_ids)
        id_real = mapa_ids[bt_id]

        historial_cofias[id_real][clase] += 1
        cofia = max(historial_cofias[id_real],
                    key=historial_cofias[id_real].get)
        crop  = frame[max(0,y1):y2, max(0,x1):x2]

        if frame_idx - last_eval_frame.get(id_real, -1e9) >=
           frames_eval:
            resultado = evaluar_epp_openai(crop)
            bitacora_epp.append({
                "timestamp": datetime.now().strftime("%Y-%m-%d␣%H:%M:%
                   S"),
                "id": id_real, "cofia": cofia, "resultado": resultado
            })
            last_eval_frame[id_real] = frame_idx

```

```
        color = CLASS_COLORS.get(cofia, DEFAULT_COLOR)
        cv2.rectangle(frame, (x1,y1), (x2,y2), color, 2)
        cv2.putText(frame, f"ID {id_real} | {cofia}",
                    (x1, max(20,y1-10)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
    frame_idx += 1
```

Listing C.8: Función de evaluación EPP con GPT-4.1 y bucle principal de detección con YOLOv11 + ByteTrack.

C.5. Pipeline de verificación de EPP con LLaVA (ejecución local)

El pipeline con LLaVA replica la arquitectura anterior, sustituyendo la API de OpenAI por el modelo *open-source* LLaVA 1.5 (7B parámetros) ejecutado localmente mediante la librería `transformers` de Hugging Face, eliminando la dependencia de servicios externos.

C.5.1. Carga del modelo y función de evaluación

```
from transformers import AutoProcessor, AutoModelForVision2Seq
from PIL import Image

LLAVA_MODEL = "llava-hf/llava-1.5-7b-hf"
device      = "cuda" if torch.cuda.is_available() else "cpu"

llava_processor = AutoProcessor.from_pretrained(LLAVA_MODEL)
llava_model     = AutoModelForVision2Seq.from_pretrained(
    LLAVA_MODEL,
    torch_dtype=torch.float16 if device == "cuda" else torch.float32
).to(device)
llava_model.eval()

def evaluar_epp(crop_bgr):
    """Evalua EPP localmente usando LLaVA 1.5-7B."""
```

```
    pil    = Image.fromarray(cv2.cvtColor(crop_bgr, cv2.COLOR_BGR2RGB)
        )
    inputs = llava_processor(images=pil, text=PROMPT_EPP,
                             return_tensors="pt").to(device)
    with torch.no_grad():
        out = llava_model.generate(**inputs, max_new_tokens=200)
    return llava_processor.batch_decode(out, skip_special_tokens=True)
        [0]
```

Listing C.9: Carga de LLaVA 1.5-7B en GPU local y función de evaluación EPP por recorte de operario.

C.5.2. Escritura de bitácora EPP

Ambos pipelines (OpenAI y LLaVA) generan al finalizar una bitácora de verificaciones en formato TXT y CSV:

```
# Bitacora TXT (formato legible para auditoria)
with open(OUTPUT_EPP_TXT, "w", encoding="utf-8") as f:
    for r in bitacora_epp:
        f.write(f"{r['timestamp']} | ID {r['id']} | {r['cofia']}\n")
        f.write(r["resultado"] + "\n")
        f.write("-" * 40 + "\n")

# Bitacora CSV (para analisis de datos posterior)
with open(OUTPUT_EPP_CSV, "w", newline="", encoding="utf-8") as f:
    writer = csv.writer(f)
    writer.writerow(["timestamp", "id", "cofia", "resultado"])
    for r in bitacora_epp:
        writer.writerow([r["timestamp"], r["id"],
                         r["cofia"],    r["resultado"]])

print(f"Bitacora guardada: {OUTPUT_EPP_TXT} | {OUTPUT_EPP_CSV}")
```

Listing C.10: Escritura de la bitácora de verificaciones EPP en formato TXT y CSV al término del procesamiento.