Instituto Tecnológico de Costa Rica

Carrera de Ingeniería Mecatrónica

Diseño de una aplicación de entrenamiento maestro-estudiante en redes neuronales para uso en sistema embebido

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Mecatrónica con el grado académico de Licenciatura

Ricardo González Víquez

Cartago, 15 de junio de 2023

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Cartago, 15 de junio de 2023

Ricardo González Víquez

Céd: 9-0118-0848

INSTITUTO TECNOLÓGICO DE COSTA RICA
PROGRAMA DE LICENCIATURA EN INGENIERÍA MECATRÓNICA
PROYECTO FINAL DE GRADUACIÓN
ACTA DE APROBACIÓN

El profesor asesor del presente trabajo final de graduación, indica que el documento presentado por el estudiante cumple con las normas establecidas por el programa de Licenciatura en Ingeniería Mecatrónica del Instituto Tecnológico de Costa Rica para ser defendido ante el jurado evaluador, como requisito final para aprobar el curso Proyecto Final de Graduación y optar así por el título de Ingeniero(a) en Mecatrónica, con el grado académico de Licenciatura.

Estudiante: Ricardo González Víquez

Proyecto: Diseño de una aplicación de entrenamiento maestro-estudiante en redes neuronales para uso en sistema embebido

Dr -Ing. Carlos Adrián, Salazar García

Asesor

Cartago, 15 de junio de 2023

INSTITUTO TECNOLÓGICO DE COSTA RICA
PROGRAMA DE LICENCIATURA EN INGENIERÍA MECATRÓNICA
PROYECTO FINAL DE GRADUACIÓN
ACTA DE APROBACIÓN

Proyecto final de graduación defendido ante el presente jurado evaluador como requisito para optar por el título de Ingeniero(a) en Mecatrónica con el grado académico de Licenciatura, según lo establecido por el programa de Licenciatura en Ingeniería Mecatrónica, del Instituto Tecnológico de Costa Rica.

Estudiante: Ricardo González Víquez

Proyecto: Diseño de una aplicación de entrenamiento maestro-estudiante en redes neuronales para uso en sistema embebido

Miembros del jurado evaluador

Msc -Ing. Felipe, Meza Obando

Jurado

Ing. Ana María, Murillo Morgan

Jurado

Ing. Eduardo José, Asenjo Castillo

Jurado

Los miembros de este jurado dan fe de que el presente proyecto final de graduación ha sido aprobado y cumple con las normas establecidas por el programa de Licenciatura en Ingeniería Mecatrónica.

Cartago, 15 de junio de 2023

Resumen

Este trabajo consiste en la implementación de una técnica de entrenamiento maestro-estudiante para llevar a cabo la reducción de una red neuronal de gran tamaño, capaz de detectar automáticamente a personas por medio de cuadros delimitadores, sin que esta red pierda su precisión mAP. Con esto se busca disminuir el tiempo de inferencia y el consumo de la memoria del modelo para poder eventualmente, en un futuro, llevar a cabo la implementación del modelo resultante en un sistema embebido de bajo coste.

Además, se aplican distintas pruebas por medio de la validación cruzada con el fin de darle una confiabilidad a los resultados. El modelo final es capaz de reducir el tiempo de inferencia y el consumo de memoria en más de un 10 % mientras que disminuye su precisión en un valor menor al 8 %.

Palabras clave: CPU, cuadros delimitadores, entrenamiento maestro-estudiante, GPU, mAP, tiempo de inferencia

Abstract

This work consists of the implementation of a master-student training technique to carry out the reduction of a large neural network, capable of automatically detecting people by means of bounding boxes, without this network losing its mAP accuracy. The aim is to reduce the inference time and memory consumption of the model in order to eventually implement the resulting model in a low-cost embedded system in the future.

In addition, different tests are applied by means of cross validation in order to give reliability to the results. The final model is able to reduce the inference time and memory consumption by more than 10 % while reducing its accuracy by less than 8 %.

Keywords: CPU, bounding boxes, master-student training, GPU, mAP, inference time

A mi persona favorita: mamá

Agradecimientos

En primera instancia le agradezco al laboratorio de investigación SIPLAB por abrirme las puertas para llevar a cabo este trabajo. Seguidamente, le agradezco a mi profesor asesor: Carlos Adrián por recibirme como estudiante para desarrollar este trabajo y por ser persona antes que profesor.

Luego, quiero agradecerle a personajes que me acompañaron desde el principio de la carrera, como lo son: Javier, Eduardo y David, con quienes formé un excelente grupo de trabajo desde el inicio de la carrera. Posteriormente, quisiera agradecerle a Daniel y Carlos, que aparecieron en los siguientes años de carrera y con quienes generé un vínculo inimaginable.

Además, quiero agradecerles a Fernanda y Luis Pedro, por ser confidentes y con quienes conviví casi todos los días en épocas de pre-pandemia, y por último quisiera agradecerles a Esteban, Angelie y Cano por ser mentores de vida y claves de mi éxito y adaptación en el TEC.

Ricardo González Víquez

Cartago, 15 de junio de 2023

Índice general

Índice de figuras

Índice de tablas

Lista de abreviaciones

- SIPLAB: Laboratorio de Procesamiento de Señales e Imágenes
- ITCR: Instituto Tecnológico de Costa Rica
- VIE: Vicerrectoría de Investigación y Extensión
- IA: Inteligencia Artificial
- AA: Aprendizaje Automático
- DL: Aprendizaje Profundo
- RN: Redes Neuronales
- CNN: Redes Neuronales Convolucionales
- va: valor actual
- vo: valor objetivo
- MAE: Error Medio Promedio
- IoU: Intersección sobre la unión
- TP: verdadero positivo
- FN: falso negativo
- FP: falso positivo
- TN: verdadero negativo
- mAP: precisión media promedio
- YOLO: You Only Look Once
- NMS: filtro de supresión no máxima
- SSD: Single Shot Detection
- RPN: Red de Propuesta de Región
- Faster R-CNN: Faster Regional-Convolutional Neuronal Network

- Mask R-CNN: Mask Regional-Convolutional Neuronal Network
- RF-Next: Next generation Receptive Field
- GPU: unidad de procesamiento gráfico
- CPU: unidad central de procesamiento
- SGD: descenso de gradiente estocástico
- ADAM: optimizador de momento adaptable
- COCO: Common Object in Context
- KD: destilación de conocimiento
- ME: modelo estudiante
- MM: modelo maestro
- Memoria RAM: memoria de acceso aleatorio
- Memoria SWAP: memoria virtual

Capítulo 1

Introducción

1.1. Antecedentes del proyecto

El Laboratorio de Procesamiento de Señales e Imágenes (SIPLAB) es un laboratorio de investigación perteneciente a la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica (ITCR). Su gestación data del año 2005, pero existe como tal desde el 2008 y tiene como propósito desarrollar soluciones de problemas nacionales o regionales referentes al ámbito de procesamiento, análisis y reconocimiento de información transmitida en señales temporales y espaciales [1]. Actualmente poseen el proyecto de investigación: "Detección automática in situ de aforo en video para retorno a presencialidad debido a la pandemia de la COVID19" ligado con la Vicerrectoría de Investigación y Extensión (VIE) del ITCR.

La VIE se encarga de brindar condiciones para que tanto estudiantes como profesores puedan: generar, adaptar y validar conocimientos de índole científico, proponer desarrollos tecnológicos y participar en distintos sistemas económicos, sociales y productivos del país [2].

Por lo tanto, el presente proyecto se desarrolla en SIPLAB por medio del asesoramiento técnico de los profesores pertenecientes a dicho laboratorio de investigación, con el fin de llevar a cabo un avance en el ámbito de detección automática de personas en sitio. En específico, se propone diseñar una herramienta que permita reducir el coste computacional necesario para realizar la tarea, mientras que se conserve su precisión. Esto con el fin de que los futuros trabajos por realizarse en el laboratorio puedan desarrollarse en sistemas embebidos de bajo coste.

1.2. Descripción del problema a resolver

Este proyecto nace como respuesta a la necesidad de desarrollar por medio de Redes Neuronales Convolucionales un detector automático de personas con el fin de llevar el control del aforo en diversos espacios. Con el inconveniente de que debe ser aplicado en dispositivos de bajo consumo debido a las posibles limitantes en términos de recursos físicos, económicos y espaciales para implementar este tipo de tecnologías tanto en el laboratorio de investigación SIPLAB como en muchos otros potenciales espacios aplicables.

Asimismo, la ausencia de herramientas que permitan reducir la cantidad de parámetros que poseen este tipo de algoritmos generan un interés grande por parte de SIPLAB en poder desarrollar un método que sea capaz de reducir el coste computacional requerido para correr los algoritmos a la vez que se preserva su rendimiento, y de esta forma poder utilizar dispositivos de bajo consumo en la aplicación requerida.

Para ello, se tiene una base de datos de acceso confidencial con imágenes colocadas de tal manera que sean lo menos invasivas en términos de la privacidad de las personas y deben ser utilizadas para llevar a cabo la detección de las personas, ya que SIPLAB pretende utilizarlas de base para su eventual sistema físico.

1.3. Síntesis del problema

Dificultad de controlar por medio de un algoritmo de inteligencia artificial el aforo de un lugar al utilizar un sistema embebido de bajo coste debido a las características de coste computacional que se necesitan para llevarlo a cabo.

1.4. Enfoque de la solución

Para llevar a cabo este proyecto, se plantea seguir una metodología específica y de esta manera poder identificar las necesidades de los clientes, luego a raíz de las necesidades se plantea encontrar métricas para poder cuantificar al final del proyecto el cumplimiento de las necesidades del cliente, posteriormente, se propone generar propuestas de conceptos solución para llevar a cabo el problema y una vez realizado esto se pretende seleccionar los conceptos más adecuados para la solución con base en información de fuentes bibliográficas y conocimiento experto. Finalmente, se busca validar la solución implementada por medio de pruebas de validación.

1.5. Objetivos

1.5.1. Objetivo General

Diseñar un detector automático de personas mediante entrenamiento maestro-estudiante para su eventual uso en sistemas embebidos.

1.5.2. Objetivos Específicos

1. Determinar el algoritmo de detección automático de personas para ser usado como red maestra.

2. Definir el algoritmo de entrenamiento maestro-estudiante a utilizar para la detección automática de personas a partir del modelo previamente definido.

3. Desarrollar un programa computacional que permita implementar un algoritmo de entrenamiento maestro-estudiante para la detección automática de personas.

4. Validar el funcionamiento y rendimiento del detector automático de personas con base en las características del eventual sistema embebido a utilizar.

1.6. Estructura del documento

Este documento presenta comienza con el capitulo actual introductorio. Luego por medio del *Capítulo 2. Marco Teórico:* detalla la teoría necesaria para la comprensión del diseño planteado y muestra información respecto al estado del arte de las técnicas utilizadas para llevar a cabo el trabajo.

Luego en el *Capítulo 3. Metodología:* expone las fases metodológicas tomadas en consideración para llevar a cabo el diseño, desarrollo y validación de este proyecto y seguidamente, en el *Capítulo 4. Propuesta de Diseño:* muestra detalladamente los pasos desarrollados para la elaboración de la propuesta de diseño final.

Finalmente, en el *Capítulo 5. Resultados y Análisis:* se exponen, analizan y validan los resultados del diseño plantado y en el *Capítulo 6. Conclusiones y Recomendaciones:* se encuentran las conclusiones surgidas luego de la investigación llevada a cabo y menciona recomendaciones respecto a puntos que se considera importante tomar en cuenta para los futuros trabajos por realizar sobre esta línea investigativa.

Capítulo 2

Marco teórico

Este capítulo abarca la explicación teórica de los fundamentos de este trabajo, por lo tanto, como se desea implementar un algoritmo de visión por medio de inteligencia artificial, se explican las bases del sistema de visión, del aprendizaje automático, de las redes neuronales y las redes neuronales convolucionales, así como el modelo a utilizar para la detección de personas y la técnica de entrenamiento maestro-estudiante, entre otros temas. A continuación se detallan todos estos aspectos teóricos.

2.1. Sistemas de Visión: clasificadores

Los sistemas de visión por computadora son todos aquellos que procesan y analizan imágenes provenientes de una cámara con el fin de generar alguna descripción de su contenido. Existen aplicaciones específicas que pertenecen al campo de visión conocidas como clasificadores. Estas tienen la misión de distinguir objetos entre un conjunto predefinido de ejemplos, donde al conjunto predefino de ejemplos se le conoce como universo de trabajo y a los grupos de objetos distinguibles dentro del universo de trabajo se denominan clases. [3][4]

El reconocimiento automático de las clases se da por medio de la evaluación de características discriminantes que permitan distinguir una clase de las demás, para ello, se deben identificar características medibles que permitan de forma cuantitativa distinguir una clase de otra [3]. Por ejemplo: los objetos presentes en la Fig. 2.1 pueden ser distinguidos y clasificados según el número de lados de su contornos exteriores.

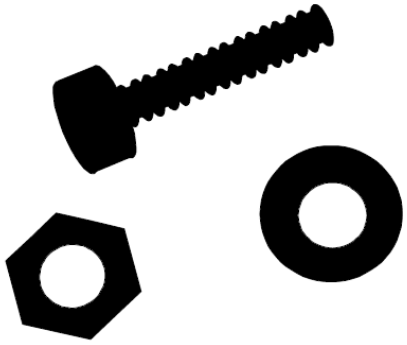


Figura 2.1: Elementos mecánicos a clasificar. Fuente: [4]

Para llevar a cabo tareas en sistemas de clasificación de objetos por medio de características discriminantes [5] expone el que se muestra en la Fig. 2.2.

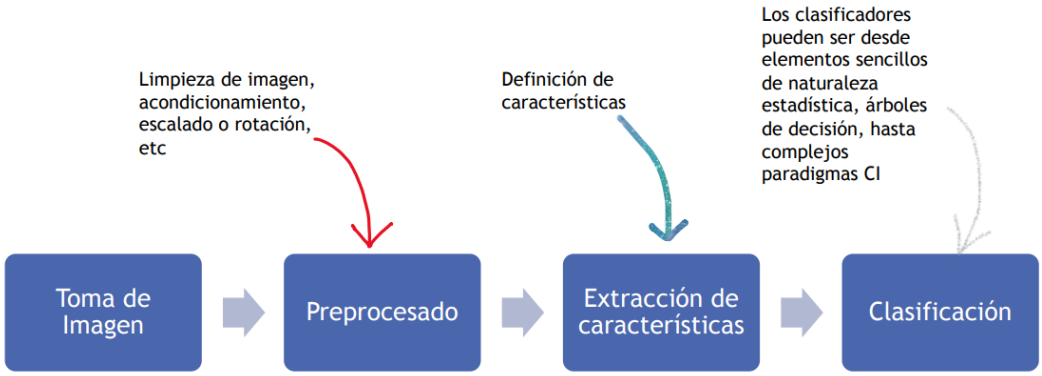


Figura 2.2: Flujo de trabajo para una clasificación de objetos. Fuente: [5]

Ahora, si bien es cierto que por medio del flujo de trabajo de la Fig. 2.2 es posible realizar la clasificación de objetos, se presentan algunos inconvenientes en el momento de aplicar este proceso debido a la alta subjetividad presente a la hora de responder las siguientes preguntas: ¿cómo definir las características discriminantes?, ¿cuál es el número adecuado de características discriminantes necesarias para identificar las clases?, ¿cómo diseñar el clasificador? y ¿cómo validar el funcionamiento del mismo? [3] Por lo tanto, el funcionamiento de una aplicación de esta índole esta sujeto a la experiencia e imaginación del diseñador, lo cual lo convierte en un tarea complicada de desarrollar.

2.2. Inteligencia Artificial: conceptos base

La Inteligencia Artificial (IA) es el área que se encarga de brindarle a los sistemas computacionales la habilidad de realizar, por si solos, actividades que requieren de inteligencia

humana, tales como: la habilidad de comprender, entender y razonar con base en la experiencia, de tener capacidad creativa, de sentir emociones o de tener intuición. Algunos ejemplos de sus aplicaciones son: el reconocimiento de imágenes estáticas, el análisis del desempeño de estrategias comerciales, el procesamiento eficiente y escalable de pacientes, la detección y clasificación de objetos, entre otros. [6][7]

2.2.1. Aprendizaje Automático

El Aprendizaje Automático (AA) es una área de la IA que se encarga de implementar técnicas y algoritmos para resolver problemas complejos de solucionar por medio de métodos convencionales de programación. Comúnmente para implementar un programa convencional se debe realizar un diseño con los pasos para llevar a cabo la tarea y luego se debe trasladar ese diseño a un lenguaje de programación; mientras que los algoritmos del AA aprenden a solucionar el problema a través de datos. [8]

La idea base del Aprendizaje Automático es generar modelos que describan el conjunto de datos con los que fue entrenados para luego predecir correctamente datos con los que no fueron entrenados. Por ello, mientras más grande sea el conjunto de datos de entrenamiento, más precisos serán los modelos planteados para solucionar los problemas. [8]

En aplicaciones donde encontrar relaciones entre los datos y lo que se desea obtener no es tan obvio es donde el AA toma mayor fuerza y sus soluciones comúnmente se dividen en las siguientes tres categorías:

- Clasificación: son los problemas en donde se clasifica a un objeto en alguna categoría [8]. Como lo es clasificar la foto de un animal entre carnívoro o herbívoro.
- Agrupación: corresponde a las aplicaciones en donde de un conjunto de datos de entrada, se dividen en grupos que contengan características en común y a diferencia de la clasificación, no se conoce el número de grupos en los que se debe dividir el problema de antemano [8].
- Predicción: Basado en el historial de los datos, se predicen los futuros valores de un problema [8]. Un ejemplo de este tipo puede ser predecir el precio de la gasolina tomando en consideración el historial de sus factores de influencia durante los últimos 3 meses.

Entrenamiento

El entrenamiento de un algoritmo de AA se puede dividir en dos etapas: una etapa de selección de los hiperparámetros del modelo, que consiste de una intervención manual

de los valores de los hiperparámetros con el fin de determinar los más apropiados para el entrenamiento. Luego, la segunda etapa utiliza los valores de hiperparámetros encontrados para entrenar el algoritmo para reducir el error del mismo. [9]

Dentro de los tipos de entrenamientos existentes se recalcan los siguientes:

- Aprendizaje Supervisado: se le presentan los datos tanto a la entrada como a la salida de la red y esta ajusta sus pesos internos durante el entrenamiento. [9]
- Aprendizaje No-Supervisado: en este tipo de entrenamiento no se le otorga ningún dato de salida, por lo que la red con los datos de entrada trata de encontrar algún patrón y genera grupos o clases entre las correlaciones que halle. [9]
- Aprendizaje por Refuerzo: se le otorga una puntuación a la salida de la red y esta aprende mediante prueba y error (recordando decisiones pasadas) a ajustar sus pesos según las puntuaciones otorgadas. [9]

Sobreajuste y subajuste

Los algoritmos de AA, en especial los de Aprendizaje Supervisado, buscan generalizar la información brindada en su entrenamiento para predecir de manera precisa datos a los que no fueron expuestos. Uno de los errores más comunes a la hora de entrenar un modelo es ajustarlo en exceso a los datos de entrenamiento. Esto ocasiona grandes problemas a la hora de exponerlo a nuevos datos debido a que el modelo memoriza peculiaridades en lugar de generalizar la información. A este problema se le conoce como sobreajuste y se muestra en la Fig. 2.3. [10]

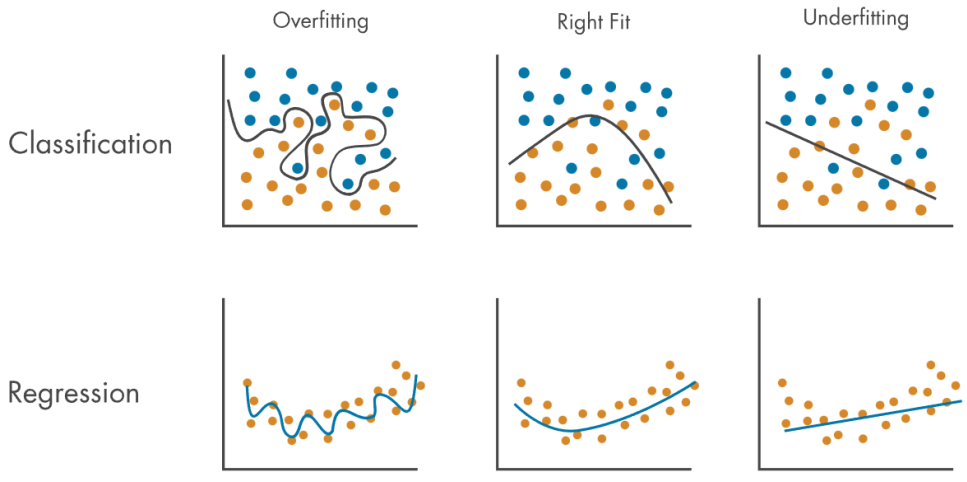


Figura 2.3: Sobreajuste y subajuste de modelos. Fuente: [11]

Los modelos sobreajustados tienen un buen rendimiento en el conjunto de datos de entrenamiento pero un mal rendimiento con en el de prueba. Existe un punto en el entre-

namiento a partir del cual el modelo comienza a sobreentrenarse, a modo de ejemplo se observa que en la Fig. 2.4 la gráfica roja (validación) comienza a subir a partir de cierto instante a pesar de la línea azul (entrenamiento) continúa en descenso. Esto significa que el modelo se está sobreentrenando. [12][13]

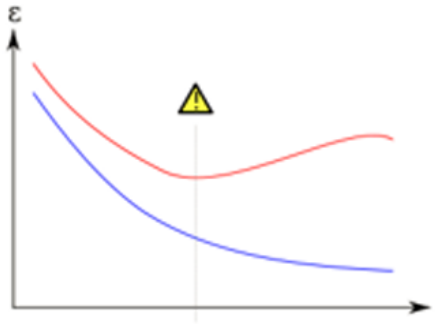


Figura 2.4: Error del modelo en función de las épocas de entrenamiento. Fuente: [13]

La causa de que suceda este fenómeno puede deberse a que el modelo aprende el ruido presente en el conjunto de datos, ya sea porque existe gran cantidad de ruido en los datos o porque la cantidad de datos es muy pequeña y por lo tanto el modelo aprende su comportamiento preciso, incluyendo el ruido. También puede ser debido a la complejidad del modelo y su relación con la varianza y el sesgo, ya que, un modelo con alta precisión en la mayoría de los casos posee un sesgo bajo y una varianza alta, lo cual significa que al exponerse a otro conjunto de datos su rendimiento va a disminuir debido a que es muy sensible a los cambios en su entrada. [13]

Por otro lado, el se dice que un modelo está subentrenado cuando no es capaz de generalizar la información, por lo que tiene una baja precisión al exponerse al conjunto de datos de validación porque no ha sido capaz de aprender la tendencia de los datos. [14]

División del conjunto de datos

En el AA se desea construir modelos computacionales que sean capaces de abstraer características generales de los datos con los que fueron entrenados, para que a la hora exponerse a datos con los que no se entrenaron, sean capaces de realizar la tarea para la que fueron implementados con el mínimo error. Una de las razones más comunes por las que los algoritmos generalizan la información de manera inadecuada es debido al sobreentrenamiento, por lo tanto, se debe recurrir a métodos de la validación cruzada. [15]

La validación cruzada es un técnica que se utiliza para evaluar y probar el rendimiento de un modelo de AA, su funcionamiento se basa en dividir el conjunto de datos T en dos secciones: uno para su entrenamiento y otro para la evaluar el rendimiento del modelo

final. Esto con el fin de brindarle estabilidad y confiabilidad al modelo resultante . Entre los métodos existentes se destacan la validación cruzada *hold-out* y *K-fold*, que son explicadas a continuación. [15][16]

La validación hold-out consiste en separar el conjunto de datos en tres secciones: una para el entrenamiento (T_{tr}), otra para la validación (T_v) y una última para el testeo (T_t). Los datos de entrenamiento es utilizado para entrenar el modelo, el de validación para ajustar (tunear) sus hiperparámetros y el de testeo para probar su funcionamiento. En caso de que un modelo no alcance las especificaciones requeridas a la hora de usar el conjunto de datos de validación T_v, debe ser entrenado con los datos de entrenamiento T_{tr} nuevamente hasta alcanzar el rendimiento deseado. Debido a que los datos de validación son utilizados para ajustar los hiperparámetros del modelo, se dice que estos poseen una influencia indirecta en el rendimiento del algoritmo, por lo que se necesita de un tercer conjunto para realizar pruebas (T_t). [15][17] En la Fig. 2.5 se aprecia un ejemplo de la configuración típica del set de datos al utilizar esta técnica.

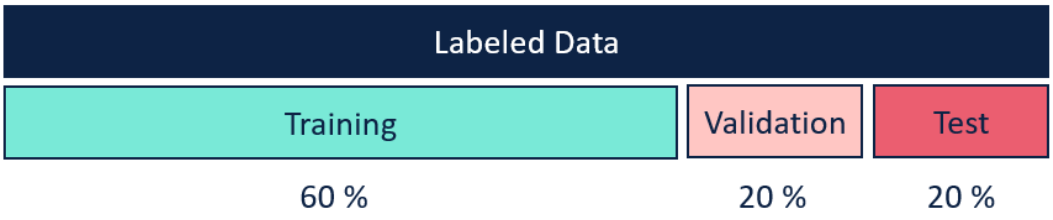


Figura 2.5: Ejemplo de la validación cruzada *hold out*. Fuente: [17]

Por otro lado, la validación cruzada K-fold utiliza la combinación de varias pruebas para obtener una estimación más estable del error. Se trata de un método que divide el conjunto de datos en K secciones iguales, donde K-1 partes se utilizan para el entrenamiento T_{tr} y la otra para la validación T_v. A su vez, la utilizada para validar el modelo cambia durante las K iteraciones y el resultado final es el promedio de todas las pruebas. [15][16] La Fig. 2.6 detalla un ejemplo de una validación cruzada 10-fold.

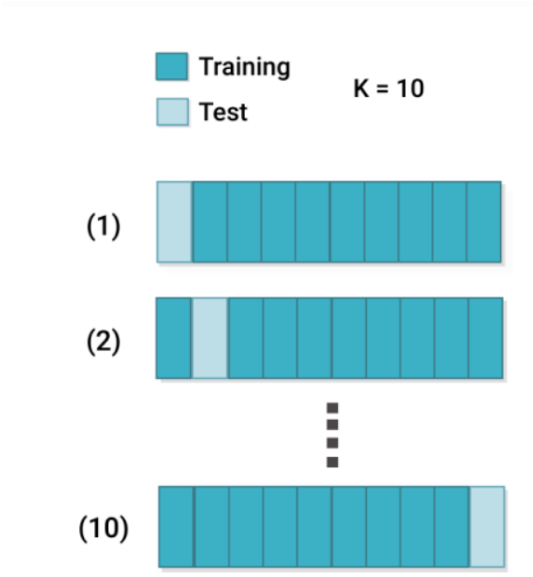


Figura 2.6: Ejemplo de la validación cruzada *K-fold*. Fuente: [16]

2.2.2. Aprendizaje Profundo

El Aprendizaje Profundo (DL, por sus siglas en inglés) es una subtécninca del AA (observar la Fig. 2.7) que se basa en utilizar gran cantidad de datos para llevar a cabo el entrenamiento de redes neuronales artificiales con gran complejidad. A través de capas de procesamiento no lineales procesa los datos ingresados para extraer características con el fin de reconocer y aprender patrones. [18]

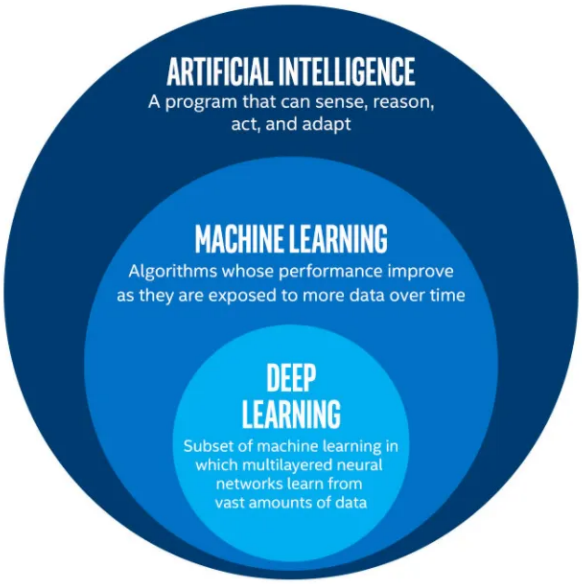


Figura 2.7: Familia del aprendizaje profundo. Fuente: [19]

Generalmente necesita de gran cantidad de recursos computacionales para ser implementado, por lo que se tiende a utilizar unidades de procesamiento gráfico (GPU) para su procesamiento, ya que son excelentes para manejar cantidades masivas de cálculos. [18]

2.2.3. Redes Neuronales

Las Redes Neuronales (RN) son un tipo de modelo existente dentro del campo de la Inteligencia Artificial que buscan emular el comportamiento del cerebro humano por medio de una computadora. Su funcionamiento se basa en la construcción de conexiones cerebrales artificiales con el fin de memorizar y asociar situaciones de un labor en específico. [20]

Neuronas humanas

El cerebro humano está compuesto por la unión de miles de neuronas que juntas al recibir una señal permiten generar una acción compleja en consecuencia, por lo tanto, se debe entender el comportamiento de las neuronas para comprender su funcionamiento. Las funciones básicas de las neuronas son: recibir señales eléctricas por medio de las dendritas, integrar las entradas para excitar o inhibir un señal de salida según la fuerza de las entradas y comunicar la señal de salida a través del axón. [21] La Fig. 2.8 muestra las partes de las neuronas recién mencionadas.

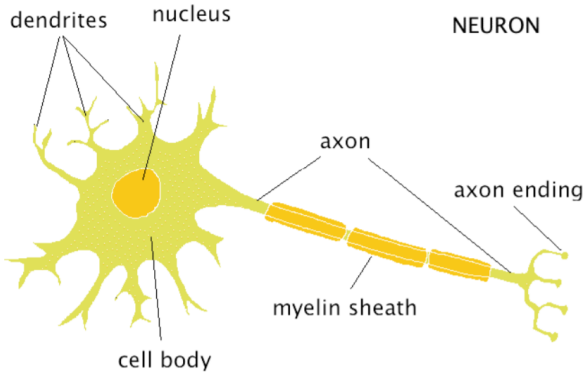


Figura 2.8: Neurona humana. Fuente: [22]

Neurona computacional

Las neuronas humanas son representadas de manera computacional por medio del modelo presente en la Fig. 2.9. Donde las entradas (P_R) emulan a las dendritas, los pesos de las entradas (W_R) corresponde a la fuerza de la señal de entrada en cada dendrita y la sumatoria ($\sum$) junto a la función de activación (f) es el cuerpo de la neurona donde se inhibe o exhibe la señal. La salida (a) compete la señal del axón. [23]

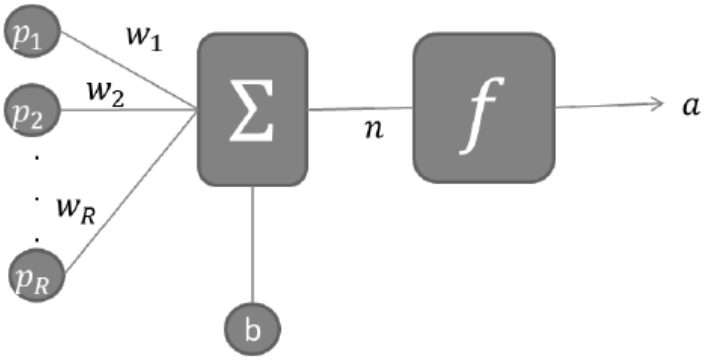


Figura 2.9: Modelo de una neurona computacional. Fuente: [23]

Para calcular la señal de salida la neurona recibe R señales de entrada que provienen de la salida de otra neurona o de una entrada del modelo, por lo que, primeramente, calcula la señal de entrada neta (n) tomando en cuenta el valor de cada entrada (P) y su respectivo peso (W) y sesgo (b). [23] Este cálculo se representa por medio de la Ec. 2.1.

$$n = b + \sum_{i=1}^{R} P_i W_i \tag{2.1}$$

Luego, se usa una función de activación para inhibir o excitar la salida según el valor neto. Dependiendo de la función que se utilice el comportamiento de la red neuronal varía. En los casos donde la función de activación no es lineal la red incrementa su capacidad de aprender datos, ya que, puede generar asociaciones no lineales, lo que significa que puede solucionar problemas complejo (no lineales); caso contrario, la red solo es capaz de generar asociaciones lineales. A continuación se muestran algunas de las funciones de activación más comunes. [24]

Sigmoide:

$$a = f(n) = \frac{1}{1 + e^{-n}} \tag{2.2}$$

Tangente hiperbólica:

$$a = f(n) = \frac{1 - e^{-2n}}{1 + e^{-2n}} \tag{2.3}$$

ReLU:

$$a = f(n) = \begin{cases} n & \text{si } n > 0 \\ 0 & \text{si } n \leq 0 \end{cases} \tag{2.4}$$

Redes computacionales

Las Redes Neuronales computacionales permiten mediante la unión de neuronas desarrollar aplicaciones específicas que presentan un alto grado de complejidad. En términos generales las RN son un conjunto de neuronas unidas en distintas capas, donde unas se encuentran en la capa de entrada, otras en las capas ocultas y otras en la capa de salida. Para conectar neuronas basta con unir la salida de una con la entrada de otra y agregar un peso a dicha conexión. Cuando todas las neuronas de dos capas consecutivas están conectadas entre sí se conoce a la red como *densa*. [7] La Fig. 2.10 muestra el ejemplo de las conexiones y partes de una RN densa.

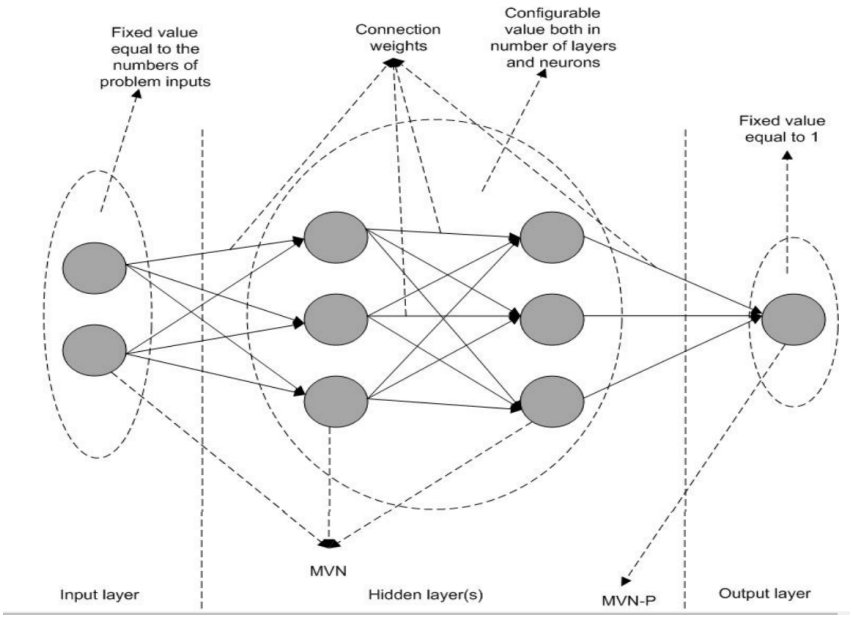


Figura 2.10: Partes de una Red Neuronal densa. Fuente: [25]

Función de coste

Al entrenar una RN se le deben proporcionar datos debidamente etiquetados para que pueda comparar el valor actual (va) proporcionado por la salida de la red respecto al valor objetivo (vo) brindado de los datos etiquetados y calcular lo que se conoce como función de coste (también conocida como función de error), que es una manera de medir la precisión, capacidad o calidad de un modelo. [26] A modo de ejemplo se muestra la Ec. 2.5 que es una función de coste que toma en consideración el promedio del error, también conocida como Error Medio Promedio (MAE, por sus siglas en inglés).

$$\varepsilon = MAE = \sum_{i=1}^{R} \frac{|vo - va|}{R} \tag{2.5}$$

Optimizador

El fin del entrenamiento de una RN es minimizar el error presente dentro de la misma, por lo tanto, busca cuantificar el error de alguna manera para luego reducirlo. A raíz de esto es que surgen los métodos de optimización. Estos buscan ajustar los pesos de las neuronas durante cada iteración por medio de una retropropagación para reducir el error del modelo. Un optimizador muy común es el Descenso de Gradiente (∇f), que consiste en utilizar derivadas parciales sobre la función de coste para igualar el resultado a cero y así encontrar mínimos locales o globales. Para ello se calculan las derivadas y se mueve el vector de pesos en dirección contraria a la pendiente, tal y como se muestra en la Fig. 2.11. [27] [28]

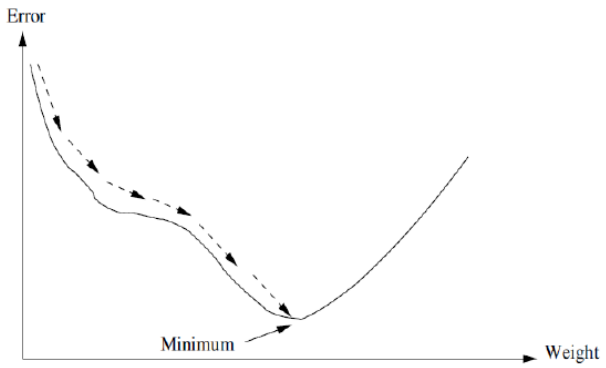


Figura 2.11: Ejemplo del optimizador Descenso del Gradiente. Fuente: [7]

Además, al Descenso de Gradiente se le añade un hiperparámetro conocido como tasa de aprendizaje (α) que consiste en un hiperparámetro que escala la afectación del gradiente al minimizar el error y por ende, permite que los pesos se ajusten de manera más rápida

durante cada iteración. La Ec. 2.6 y 2.7 son las ecuaciones del Descenso del Gradiente y de la tasa de aprendizaje, respectivamente. Donde w^i es el valor de los pesos de la iteración actual y w^{i-1} el de la siguiente época. [29]

$$\nabla f = \frac{\partial \varepsilon}{\partial w_R^i} \tag{2.6}$$

$$w^{i+1} = w^i - \alpha \nabla f \tag{2.7}$$

Mientras mayor sea la tasa menos iteraciones son necesarias para minimizar el error pero existe la posibilidad de que nunca converja, ya que este puede brincar de un mínimo local a otro constantemente, caso contrario, mientras menor sea la tasa, más épocas son necesarias para optimizar el error y esto puede que reduzca la posibilidad de encontrar un mejor mínimo local. Actualmente muchos algoritmos de optimización utilizan tasas de aprendizaje variables para solucionar estos inconvenientes. [29]

2.2.4. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) son un tipo específico de RN que presentan dos fases. La primera fase, también conocida como la fase de extracción de características, contiene capas que aplican filtros convolucionales para abstraer información del conjunto de datos y otras capas que por medio de un proceso de submuestreo reducen el tamaño de las características obtenidas. La segunda fase comúnmente es una red neuronal densa que se encarga de clasificar la información en las clases deseadas. La cantidad de capas y la cantidad de neuronas de la RN dependen y varían según el modelo de la aplicación a realizar [30]. La Fig. 2.12 ejemplifica las partes recién descritas de las CNN.

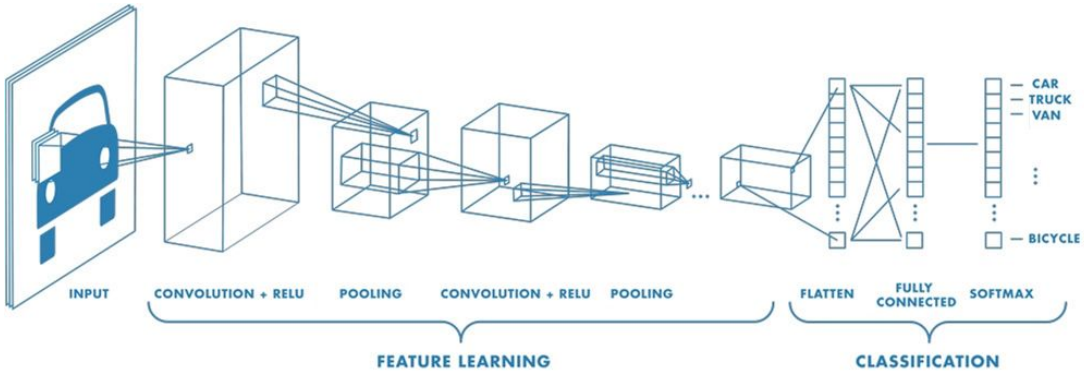


Figura 2.12: Topología de una Red Neuronal Convolucional. Fuente: [31]

Capas Convolucionales

La operación de convolución es una pieza imprescindible de las Redes Neuronales Convolucionales. Por medio de ella los modelos son capaces de abstraer las características fundamentales del conjunto de datos al aplicar una convolución matricial entre la imagen de entrada y una matriz convolucional (conocida como kernel o filtro). La imagen de la Fig. 2.13 ejemplifica el resultado de la aplicación de esta operación, que es conocido como mapa de activación o mapa de características. [32][33]

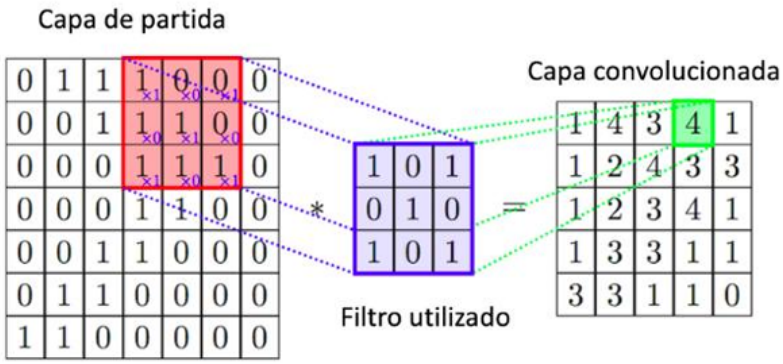


Figura 2.13: Convolución entre una imagen 7x7 y un kernel 3x3. Fuente: [34]

El contenido de un kernel varia según la característica que se desea filtrar de la imagen. Por ejemplo, existen algunos que permiten obtener los bordes horizontales y verticales de los objetos presentes en las imágenes, tal y como se aprecia en la Fig. 2.14. Esta aplicación abstrae la información para luego procesarla, por lo que en la etapa de entrenamiento de las CNN los pesos que se buscan ajustar son el contenido de las matrices de los kernels, ya que en este tipo de aplicaciones se desconoce de antemano el tipo de filtros a utilizar. [35]

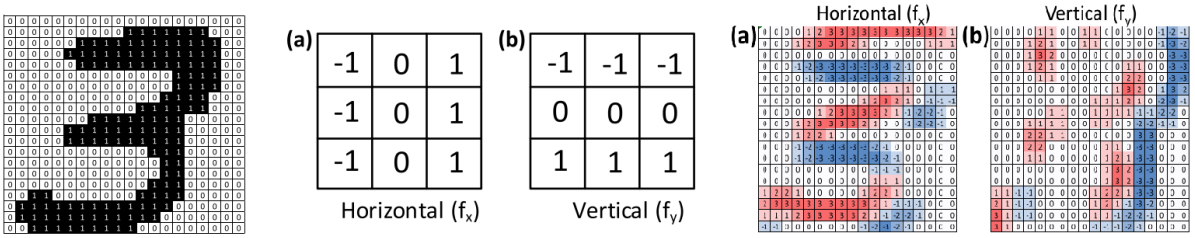


Figura 2.14: Resultado de aplicar kernels para obtener los bordes. Fuente: [35]

Además, es común aplicar una función de activación sobre el mapa de características resultante para modificar su contenido. Esto se realiza con el fin de limitar los valores de salida dentro de un rango definido o de darle la capacidad de abstraer características no lineales. En diversos casos se utiliza la función ReLU (Ec. 2.4) para eliminar los valores negativos de la matriz. [36][37]

Una vez realizado esta etapa se tiene una imagen convolucional resultante de menor tamaño. En el ejemplo de la Fig. 2.13 se aprecia como se reduce una imagen de tamaño 7x7 a una de 5x5. Esto puede llegar a ser un problema si se aplica este proceso reiteradas veces debido a la potencial pérdida de información, por lo que existe una técnica conocida como zero-padding que puede llegar a solucionarlo. Se basa en mantener el tamaño de la imagen al agregar un borde de ceros alrededor de la imagen resultante, lo que permite mantener la información en futuras convoluciones sin afectar el resultado de las mismas. La Fig. 2.15 es un ejemplo de su aplicación. [38]

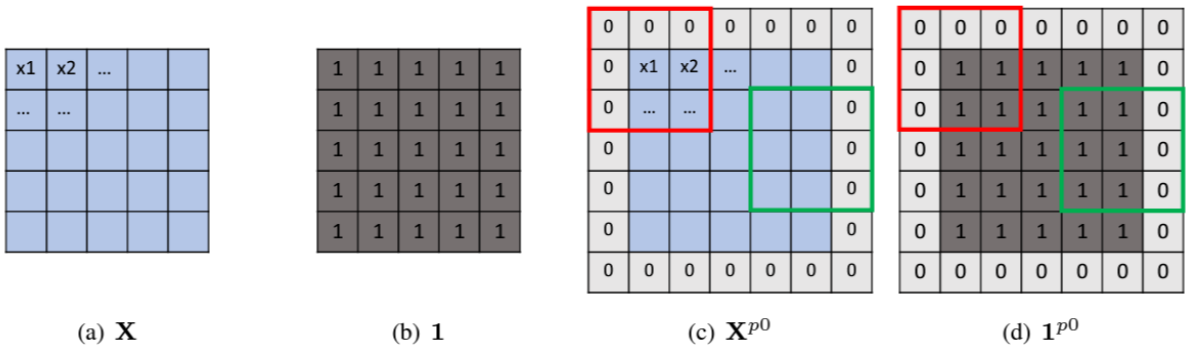


Figura 2.15: Ejemplo de la aplicación del zero-padding. Fuente: [39]

Otro factor relevante a analizar al trabajar con las CNN es el stride (también conocido como paso). Este hiperparámetro controla la cantidad de píxeles que se desplaza el kernel sobre la imagen después de cada operación. Un stride alto ayuda a reducir el tiempo de computación necesario al realizar la convolución, pero sacrifica la resolución de la imagen. La Fig. 2.16 muestra una convolución de una imagen de tamaño 7x7 con un kernel 3x3 al aplicar un paso equivalente a 2. [40]

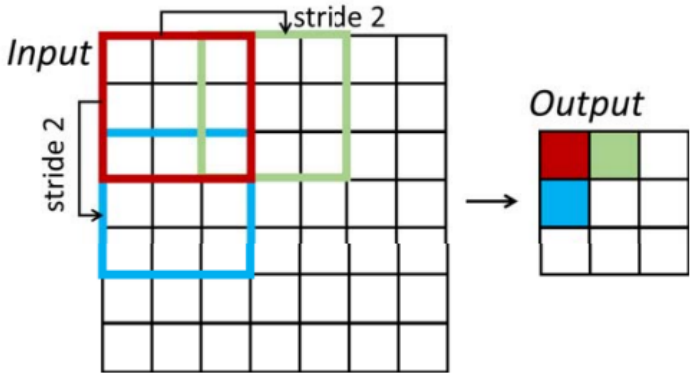


Figura 2.16: Convolución con un stride de 2. Fuente: [40]

Capas de Submuestreo

La arquitectura convencional de las Redes Neuronales Convolucionales luego de cada capa convolucional utiliza una capa de submuestreo y comúnmente se aplican en cascada de par en par. Esto con el fin de reducir la dimensión de los mapas de características resultantes y por ende, disminuir el coste computacional requerido para su funcionamiento. Esta técnica ha permitido que conjuntos de datos de gran tamaño puedan ser utilizados a la hora de entrenar modelos. [41]

Las capas de submuestreo más utilizadas son la *max pooling* y la *average pooling*, que calculan el valor máximo o la media de un segmento del mapa de activación y descartan el resto de valores. La Fig. 2.17 muestra la aplicación de ambas técnicas. [42]

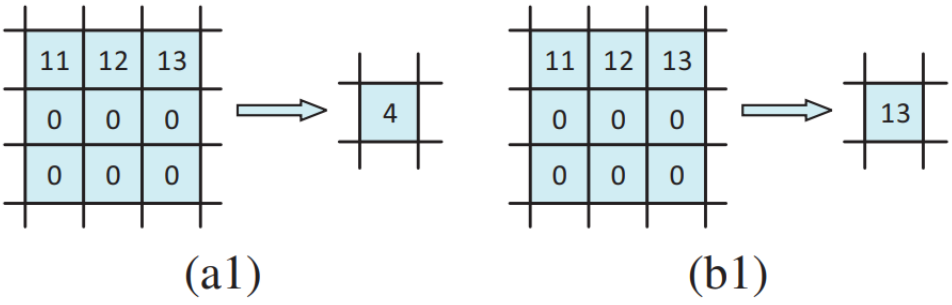


Figura 2.17: Aplicación de la capa de submuestreo max pooling (a1) y average pooling (b1). Fuente: [42]

Capas Densas

Luego de que se encuentran las características fundamentales del conjunto de datos por medio de las capas convolucionales y se reduce el tamaño de las mismas a través de

las capas de submuestreo, se utiliza un conjunto de capas densamente conectadas (redes neuronales) para reconocer los patrones presentes en los mapas de activación resultantes de las etapas anteriores y así poder clasificar las imágenes de entrada según el ajuste de los pesos de las capas neuronales.

2.2.5. Métricas

IoU

La intersección sobre la unión (IoU, por sus siglas en inglés) es una métrica utilizada para evaluar el acierto presente a la hora de predecir un cuadro delimitador en un sistema de detección. Conforme el valor de IoU se acerca a 1, significa que la predicción es más exacta. La Fig. 2.18 muestra como se evalúa el cuadro delimitador original respecto al predicho, para obtener la intersección sobre la unión. [43][44]

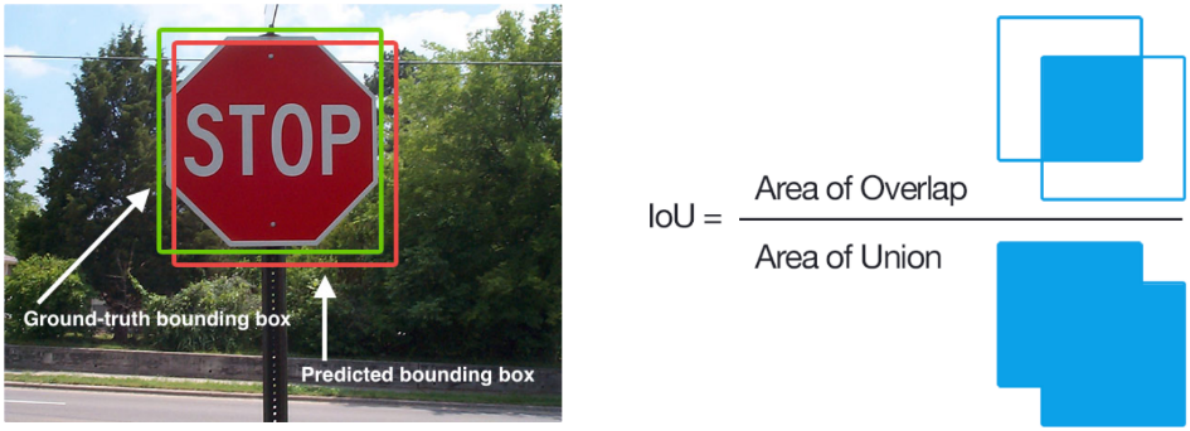


Figura 2.18: Ejemplo de la aplicación de la métrica IoU. Fuente: [43]

Matriz de confusión

La matriz de confusión es una manera estándar por la que comúnmente se muestra el error presente en un clasificador, tal y como se aprecia en la Fig. 2.19, que otorga la relación presente entre la categoría predicha y la actual. [45]

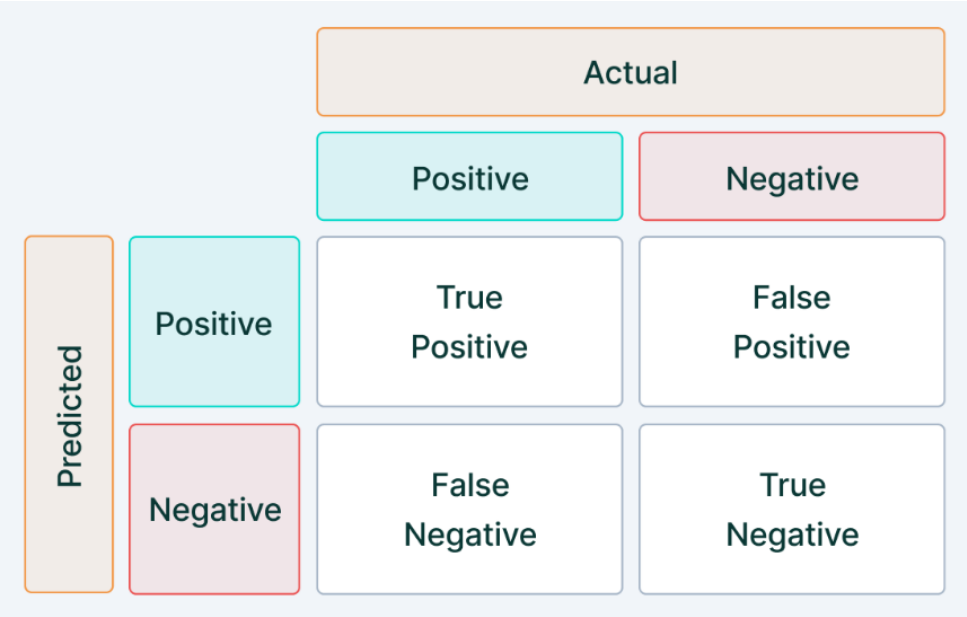


Figura 2.19: Matriz de confusión. Fuente: [43]

En los casos que una clase es positiva y se predice como positiva se tiene un verdadero positivo (TP), mientras que si se predice como negativa se tiene un falso negativo (FN), por otro lado, si la clase actual es negativa y se predice como positiva, se tiene un falso positivo (FP) y si se predice como negativa se tiene un verdadero negativo (TN). Para un problema de n clases la matriz posee un tamaño n x n. A modo de ejemplo se adjunta en la Fig. 2.20 una matriz de confusión de 10 clases. [45]

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	51	0	1	0	5	3	1	1	1	4
2	1	56	1	0	7	0	0	0	0	1
3	0	0	61	0	4	3	0	0	0	3
4	0	0	0	65	3	0	1	0	2	2
5	0	0	0	0	101	0	2	0	0	5
6	1	0	0	2	1	51	4	0	0	24
7	1	0	0	0	0	0	148	0	0	0
8	6	0	0	1	4	0	1	53	1	2
9	0	0	0	3	1	1	0	0	75	2
10	0	0	0	0	4	1	1	0	1	160

Figura 2.20: Ejemplo de una matriz de confusión de 10 clases. Fuente: [45]

Exactitud

Por medio de la exactitud se evalúa el total de aciertos del modelo respecto al número de predicciones realizadas, se utiliza la Ec. 2.8 para definirla. [46]

$$Exactitud = \frac{TP + FN}{TP + FP + TN + FN} \tag{2.8}$$

Precisión

La precisión se utiliza para medir el acierto del total de predicciones positivas, es decir, sigue lo presente en la Ec. 2.9. [46]

$$Precision = \frac{TP}{TP + FP} \tag{2.9}$$

Sensibilidad

La sensibilidad indica cual es la proporción de los positivos reales que están bien clasificados. Para ello cumple con lo mostrado en la Ec. 2.10. [46]

$$Sensibilidad = \frac{TP}{TP + FN} \tag{2.10}$$

mAP

La precisión media promedio (mAP, por sus siglas en inglés) es una manera de evaluar los modelos de detección de objetos en la que se toma en consideración tanto las predicciones positivas como las negativas. Para calcular su valor se sigue el flujo presente en la Fig. 2.21. [47]

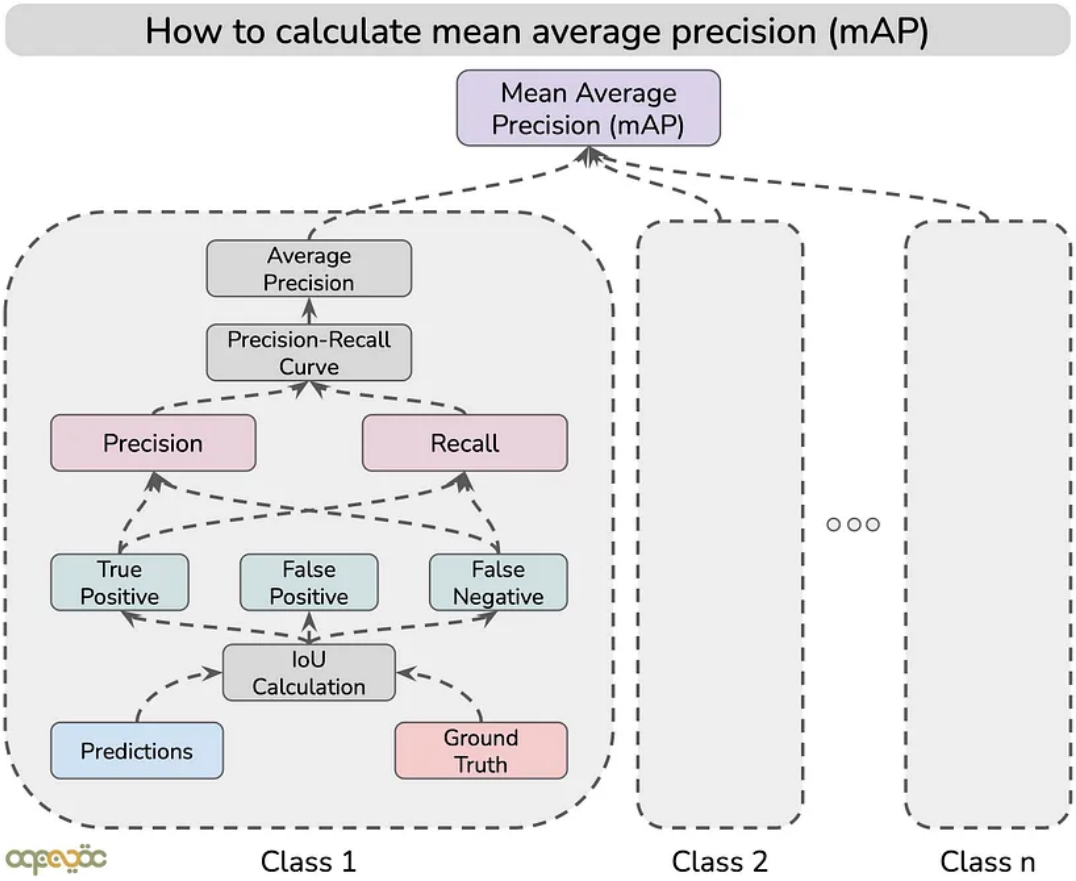


Figura 2.21: Pasos a seguir para calcular la métrica mAP. Fuente: [48]

En dicho flujo se aprecia que, en primera instancia, se comparan los cuadros delimitadores predichos respecto a los verdaderos y se calcula la IoU. Luego, según el valor del umbral de la IoU se genera la matriz de confusión de las predicciones y con base en los valores de TP, FP y FN se calcula la precisión y la sensibilidad por medio de las Ecs. 2.9 y 2.10. Posteriormente, se calcula la curva de la precisión en función de la sensibilidad. Para ello se varía el umbral de confianza utilizado por el modelo al predecir algún objeto. Mientras menor sea el valor de este umbral mayor será la cantidad de predicciones realizadas, pero al coste de que no necesariamente sean correctas. Por lo tanto, utilizando diversos umbrales de confianza se obtienen varios puntos de precisión-sensibilidad. [48]

Una vez obtenida la curva, se determina la precisión media al calcular el área bajo la curva utilizando cada uno de los puntos. A modo de ejemplo se detalla en la Fig. 2.22 la ecuación utilizada para calcular el área bajo la curva de la gráfica. [48]

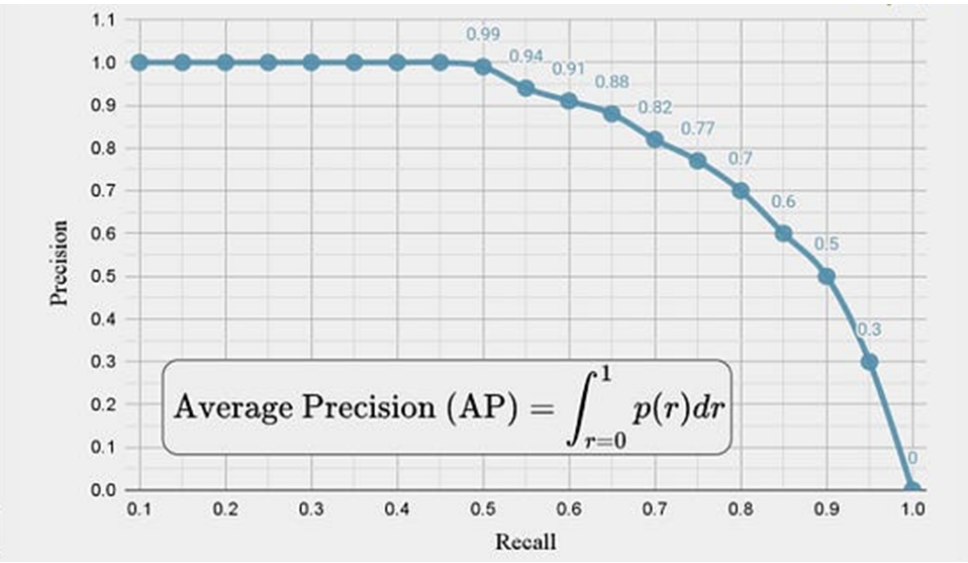


Figura 2.22: Cálculo del área bajo la curva de la precisión en función de la sensibilidad. Fuente: [48]

Una vez obtenido la precisión media para una clase, se calculan estos valores para todas las clases del conjunto de datos con el que se entrena el modelo y por medio de la Ec. 2.11 se obtiene la métrica mAP. [48]

$$mAP = \frac{1}{k} \sum_{i}^{k} AP_i \tag{2.11}$$

2.3. Métodos de detección automática de objetos

2.3.1. You Only Look Once (YOLOv6 v3.0)

YOLO es un modelo de detección automática conformado por una única red convolucional (en la Fig. 2.23 se aprecia su tipología) que predice múltiples cuadros delimitadores para los objetos de una imagen y brinda la probabilidad de pertenezcan a un clase [49].

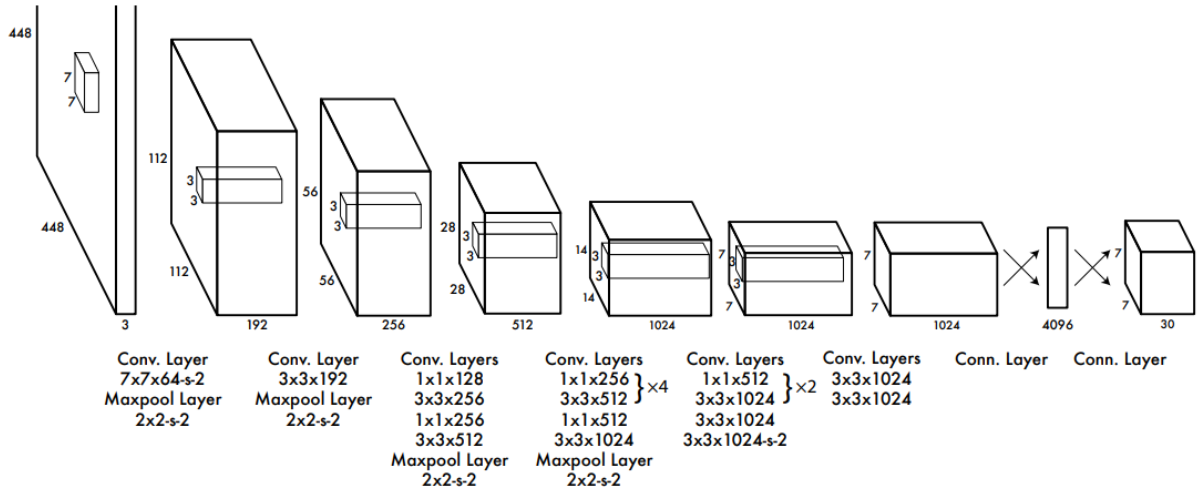


Figura 2.23: Arquitectura original del modelo YOLO. Fuente: [50]

Para ello, divide la imagen de entrada en una cuadricula de celdas de tamaño SxS y para cada celda predice la posibilidad de que exista un objeto de alguna clase dentro de ella. Luego, cada celda otorga N cuadros delimitadores de posibles objetos y les otorga una confianza en su clasificación. Donde, los cuadros delimitadores vienen dados por el centro del objeto (x, y) y el ancho de los objetos (w, h), mientras que la confianza se define por medio de la intersección sobre la unión (IoU por sus siglas en inglés) de la caja predicha respecto a la real. [50][51]

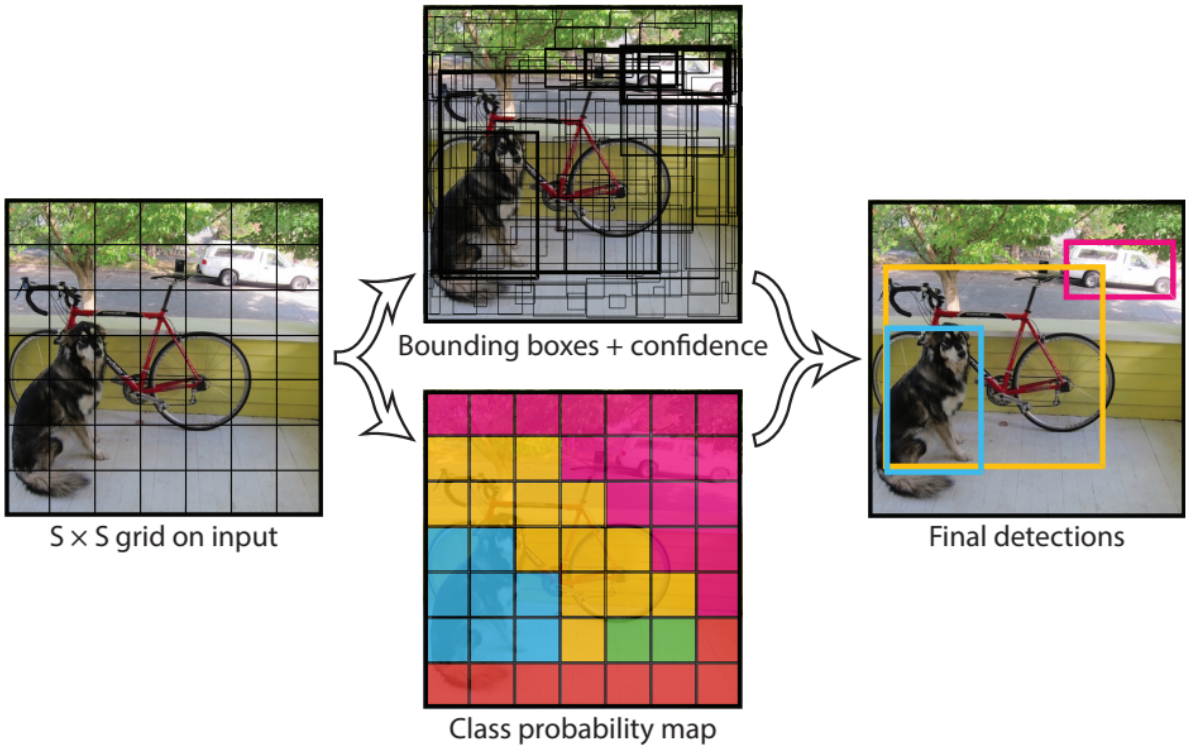


Figura 2.24: Ejemplo del funcionamiento de YOLO. Fuente: [50]

Al final del proceso se obtienen SxSxN posibles objetos, por lo que, se debe aplicar un filtrado para mostrar solamente los reales. Para ello, se define un umbral de la IoU para que el algoritmo sea capaz de filtrar todas aquellas detecciones que no posean un acierto tan alto y además, utiliza un filtro de supresión no máxima (NMS por sus siglas en inglés) que ayuda a preservar en aquellos objetos que tengan múltiples cuadros predichos solo el que posea mayor puntaje de clasificación, es decir, clasifica un objeto detectado según la clase que posea mayor probabilidad. [50]

2.3.2. Single Shot Detection (SSD)

Al igual que YOLO, está conformado por una sola red convolucional con la diferencia que utiliza los mapas de activación para la detección de los objetos. Para lograr esto, utilizan pequeños filtros convolucionales (también conocidos como kernels) en los mapas de activación de las últimas etapas del modelo, con ello buscan detectar los cuadros delimitadores y predecir la categoría a la que pertenecen. [52]

La ventaja de aplicar distintos filtros en diferentes capas del modelo es que pueden detectar objetos de distintos tamaños. Esto resulta beneficioso en el caso que se utilicen imágenes de baja resolución, ya que, permite tener una alta exactitud en la detección aún bajo esas condiciones [52]. La Fig. 2.25 ejemplifica la detección de objetos en dos distintos mapas de activación.

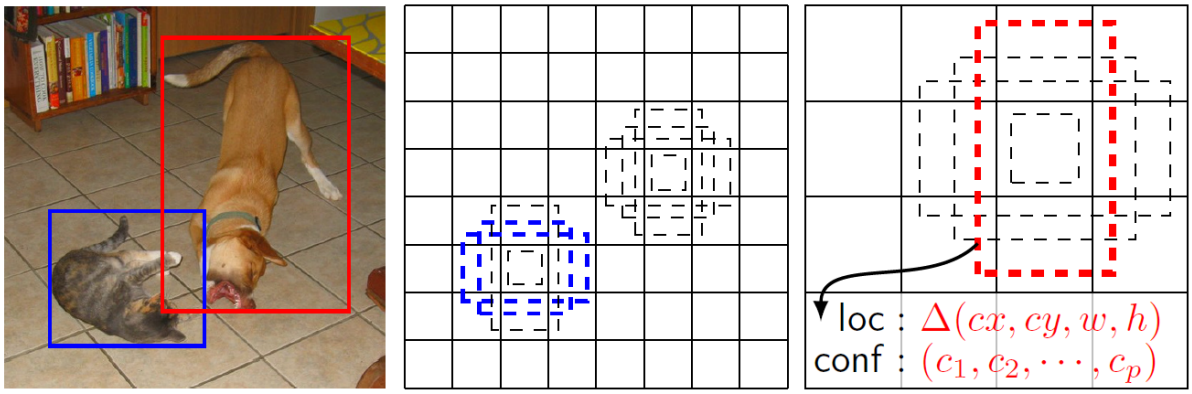


Figura 2.25: Detección del modelo SSD en distintos mapas de activación. Fuente: [52]

Por último, tal y como se aprecia en la Fig. 2.26, la topología obtiene clasificaciones de diversos mapas de activación y las filtra por medio de un filtro de supresión no máxima. Por lo tanto, para cada objeto identificado se obtiene un único cuadro delimitador con su clasificación correspondiente.

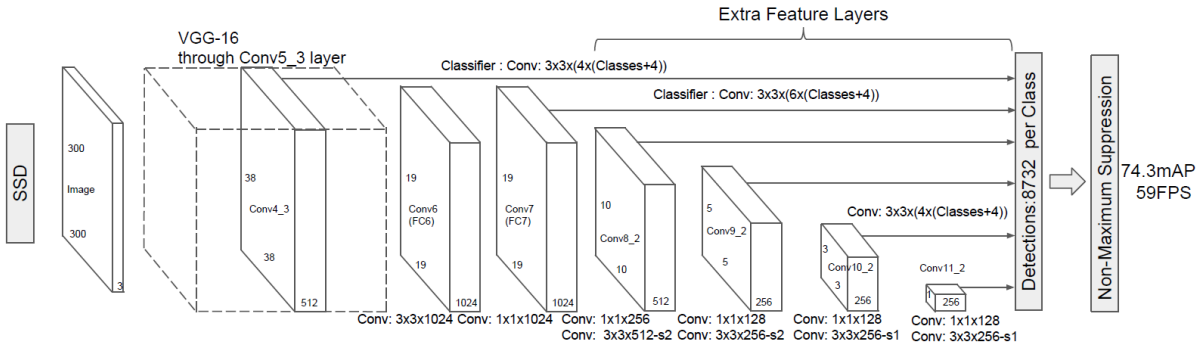


Figura 2.26: Arquitectura del modelo SSD. Fuente: [52]

2.3.3. Faster Regional-Convolutional Neuronal Network (Faster R-CNN)

Faster R-CNN es un modelo de detección de objetos que utiliza dos redes neuronales para su funcionamiento: una que indica potenciales lugares donde se pueden encontrar y otra que detecta y clasifica objetos con base en los potenciales lugares propuestos [53]. Se basa en el modelo Fast R-CNN con la diferencia que mejora la manera en que la red neuronal RPN (Region Proposal Network) propone las regiones de eventuales objetos [54].

En este caso la red RPN es una red convolucional densa que recibe la imagen de entrada y genera rectángulos de regiones propuestas con una puntuación. Para hacer esto, pasan una ventana deslizante de nxn por todo el mapa de activación de la última capa convolucional y obtiene un mapa de características. La imagen de la Fig. 2.27 representa este proceso. [55]

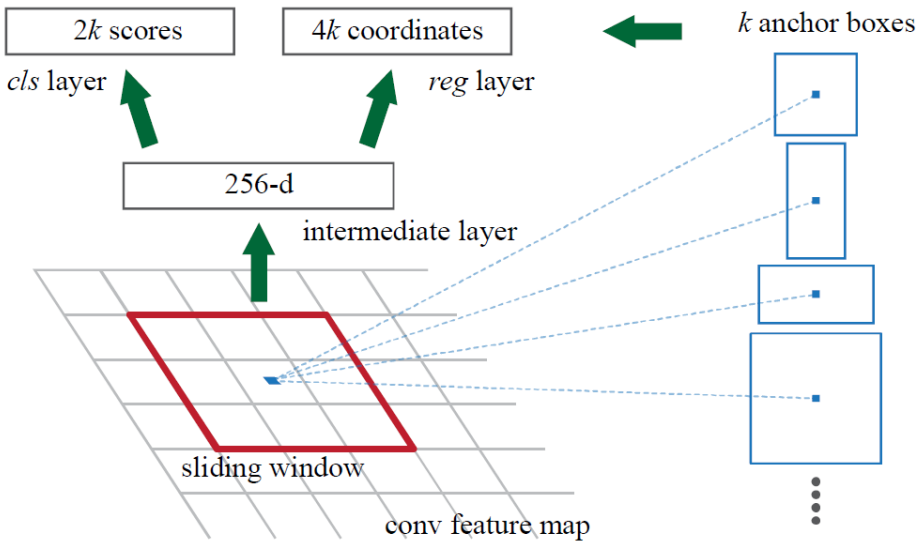


Figura 2.27: Ejemplo del resultado de la ventana deslizante en la RPN. Fuente: [55]

Luego tal y como lo muestra la Fig. 2.28 el mapa de características se pasan hacia dos redes convolucionales densas: una de regresión que predice las coordenadas y dimensiones de los cuadros delimitadores de objetos y otra de clasificación que predice qué tan probable es que una caja propuesta contenga alguna de las clases de objetos que se buscan. [55] Y con ello, se detecta los objetos y su clasificación en las imágenes.

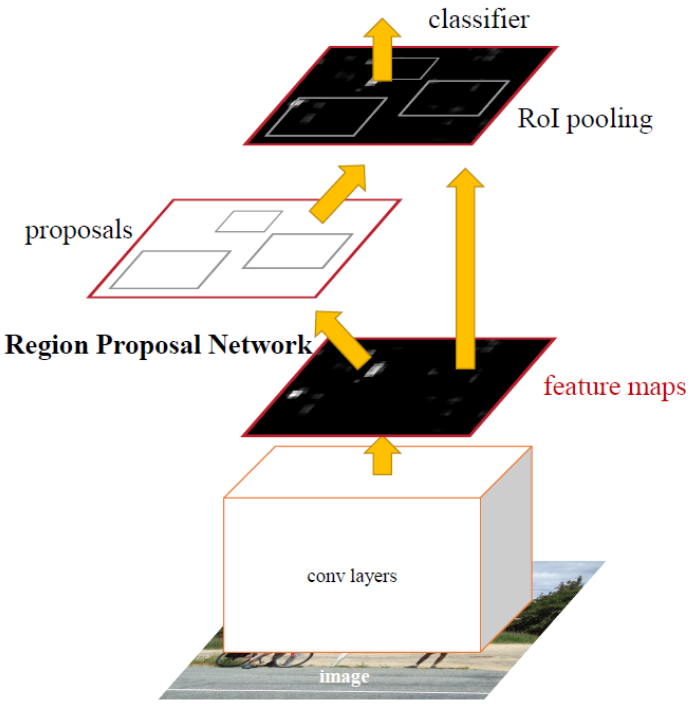


Figura 2.28: Arquitectura de Faster R-CNN. Fuente: [55]

2.3.4. Mask Regional-Convolutional Neuronal Network (Mask R-CNN)

Este modelo es una mejora del modelo Faster R-CNN, donde no solo se realiza la detección del cuadro delimitador de los objetos y su respectiva clasificación, sino que también utiliza una máscara binaria que segmenta los objetos. [56]

En este caso Mask R-CNN utiliza la misma red neuronal que indica posibles regiones donde se puedan encontrar objetos (RPN), pero, para para el caso de la red neuronal que identifica y clasifica los objetos posee una leve variación, en donde obtiene también una máscara binaria para cada región de interés (ROI) indicada por la RPN [56]. La Fig. 2.29 muestra a detalle este proceso.

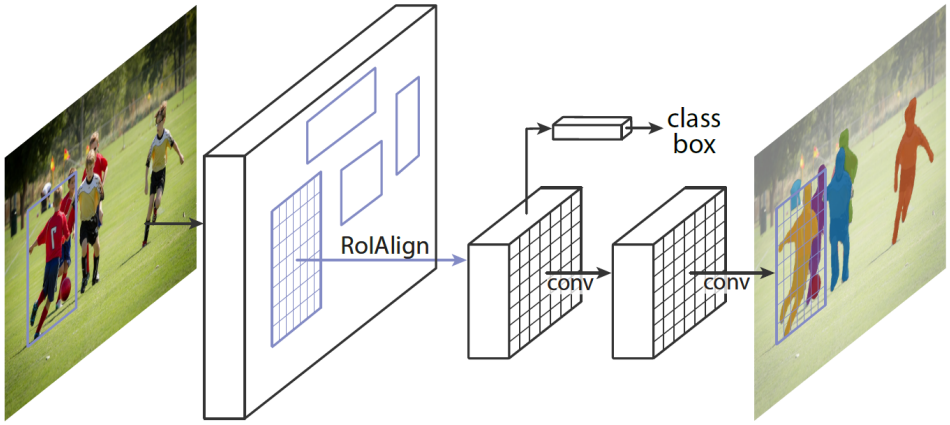


Figura 2.29: Proceso de segmentación de Mask R-CNN. Fuente: [56]

Esta máscara de tamaño mxm es la encarga de realizar la segmentación y por lo tanto, ayuda a diferenciar objetos en ambientes más poblados con mayor facilidad tal y como se aprecia en la Fig. 2.30.

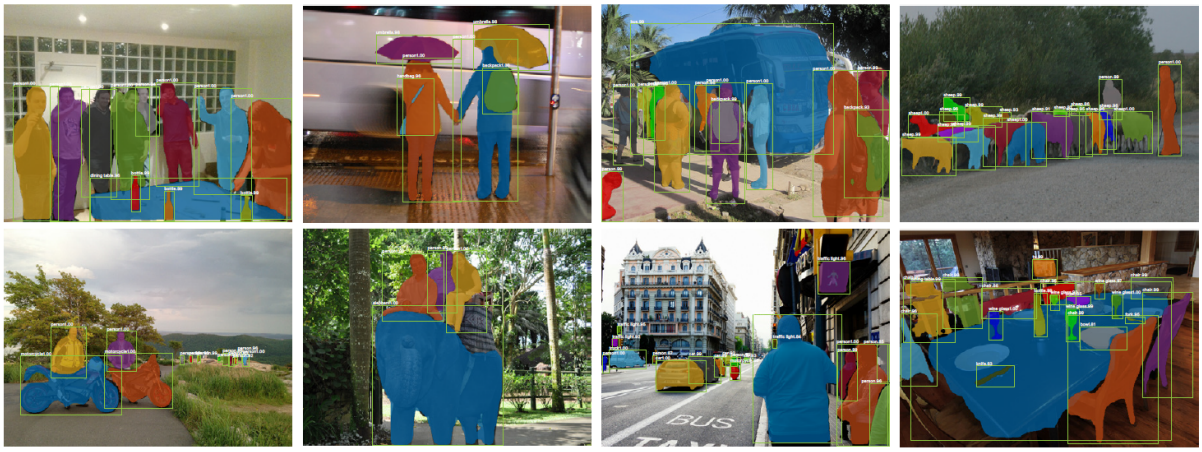


Figura 2.30: Ejemplos de la segmentación de Mask R-CNN. Fuente: [56]

2.3.5. Next generation Receptive Field (RF-Next)

Este modelo corresponde a un algoritmo genético que busca mejorar otros modelos de detección de objetos como YOLO y R-CNN por medio del estudio de los campos receptivos de las redes neuronales. Los campos receptivos son aquellas variables que tienen inferencia en los resultados del entrenamiento de una CNN. En este caso se analizan aspectos como la taza de dilatación, el tamaño del kernel, el tamaño del pooling, el paso de la convolución y el número de capas internas. [57]

Se presentan dos componentes para desarrollar este enfoque: en primera instancia una

búsqueda global de los campos receptivos por medio del algoritmo genético y luego, una búsqueda interna local que busca refinar la búsqueda global realizada. [57]

La búsqueda global busca encontrar las posibles combinaciones de campos receptivos. Debido a que existen gran cantidad de posibles combinaciones y con el fin de garantizar diversidad en los resultados, utilizan un cruce aleatorio y una mutación aleatoria en la construcción de nuevas combinaciones. [57] La Fig. 2.31 muestra a detalle este punto.

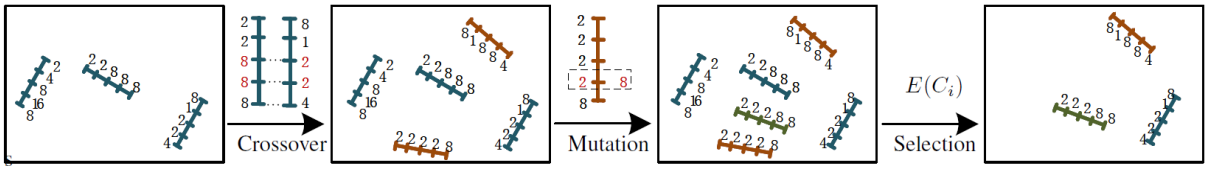


Figura 2.31: Ilustración del proceso de creación de nuevas combinaciones de campos receptivos en CNN. Fuente: [57]

Una vez obtenidos los candidatos de la búsqueda interna por medio del entrenamiento del algoritmo genético, se realiza una búsqueda interna con el fin de adecuar los campos receptivos de los candidatos a la tarea requerida y de esta manera se mejora la eficiencia de modelos de CNN ya existentes. [57]

2.4. Métodos de entrenamiento maestro-estudiante

2.4.1. Podado

El podado es una técnica de reducción de parámetros que trata de disminuir el coste computacional necesario al utilizar redes neuronales convolucionales sin disminuir su exactitud. Para lograr este objetivo [58] plantea una estrategia de tres etapas tal y como se muestra en la Fig. 2.32.

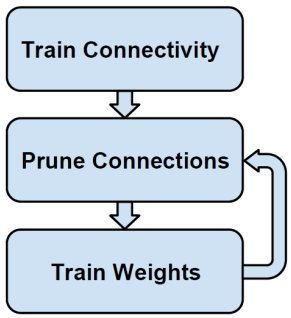


Figura 2.32: Fases de la técnica de podado: [58]

En primera instancia, como todo modelo de Inteligencia Artificial, se debe entrenar la red neuronal, luego, se plantea desconectar todas las conexiones que sean menores a cierto umbral y por último, se debe reentrenar la red neuronal para ajustar los pesos de las conexiones resultantes. Esto como resultado proporciona una red con la misma capacidad de funcionamiento, pero con una considerable menor cantidad de parámetros, conexiones y pesos, debido a que se eliminan todas las conexiones consideradas como redundantes. Además, este proceso disminuye la probabilidad de sobreentrenar la red. [58] La Fig. 2.33 muestra un ejemplo de la red neuronal previo a la aplicación de la técnica y luego de.

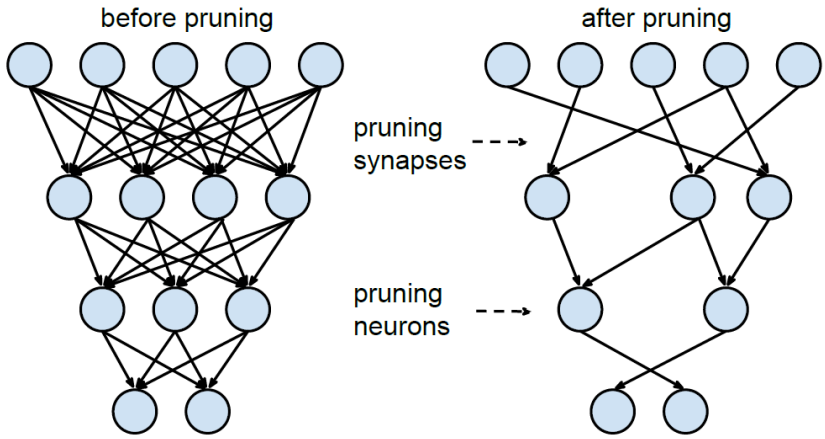


Figura 2.33: Resultado de la aplicación de la técnica de podado. Fuente: [58]

2.4.2. Cuantización

Esta técnica pretende disminuir el coste computacional requerido por una red neuronal por medio de una reducción de los bits que representan los valores de los pesos de las neuronas. [59]

Pretende disminuir el tiempo de inferencia de los cálculos al representar los valores flotantes de 32 bits por 8 bits (o menos). Además, justifica esta reducción en que la mayoría de dispositivos están diseñados para procesar datos de 8 bits de manera más rápida. Para ello, se plantea dos enfoques: una función de escala que pretende ajustar en un número de bits los valores de los pesos y los valores de activación de la red neuronal y otro de redondeo encargada de representar dichos valores en los bits deseados. [59]

Lo beneficioso de esta técnica es que sin necesidad de entrenar nuevamente la red es posible mejorar su rendimiento sin disminuir considerablemente su precisión.

2.4.3. Compresión

DeepSZ es una técnica de compresión de datos con pérdida controlada por errores que busca reducir la cantidad de parámetros de una red neuronal y a la vez preservar su precisión. Para lograrlo, [60] plantea las cuatro etapas presentes de la Fig. 2.34.

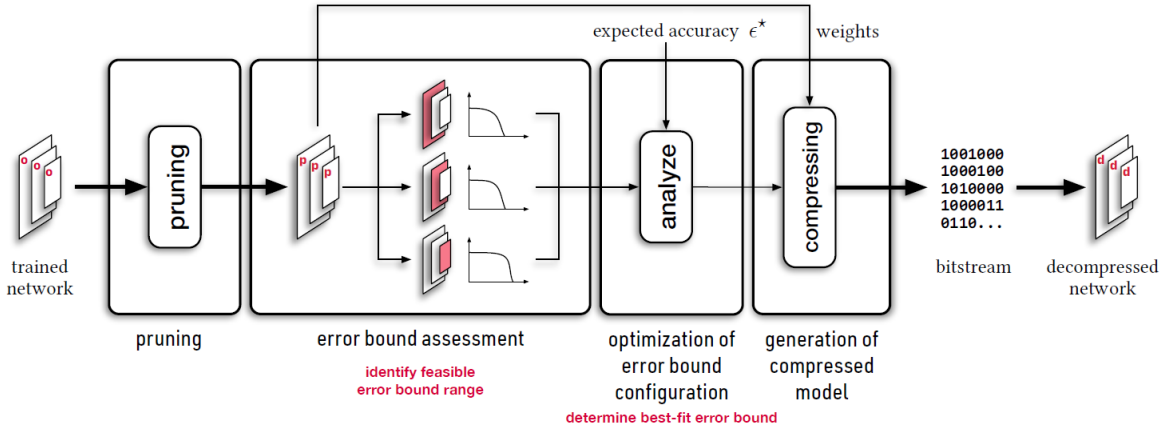


Figura 2.34: Etapas de la técnica de compresión de redes neuronales DeepSZ. Fuente: [60]

En primera instancia se plantea podar la red con el fin de eliminar toda aquella información redundante. Luego, se realiza una evaluación de los límites de error (EBA por sus siglas en inglés) que calcula el error al evaluar la precisión de la red comprimida respecto a la original. La compresión se realiza por medio de una cuantificación no uniforme, en donde se le otorga más bits a los parámetros con mayor inferencia y menos bits a los de menor relevancia. [60]

Posteriormente se optimiza la configuración de los límites de error para cada capa de la red neuronal. Para ello, se entrena el algoritmo para que encuentre una configuración que cumpla con la precisión y la tasa de compresión requerida. Por último se genera el modelo resultante con los pesos comprimidos. [60]

2.4.4. Destilación de conocimiento

En aplicaciones como la detección de objetos, el entrenamiento de una red neuronal convolucional necesita un gran conjunto de datos, que en muchas ocasiones contiene datos redundantes, lo que puede llevar a que un modelo incremente su tamaño aprendiendo la tendencia en lugar de la información. [61]

Por lo tanto, [61] propone *destilar* el conocimiento aprendido por una red neuronal maestra comúnmente robusta a otra red neuronal estudiante más eficiente, con el fin de que la

estudiante aprenda sobre la información general y omita aquel conocimiento redundante o específico. Para ello, plantea utilizar *objetivos suaves* en lugar de *objetivos duros*. Los objetivos suaves son el resultado de aplicar la Ec. 2.12 (softmax) en la salida de la red maestra con el fin de obtener una distribución de probabilidad suavizada de las salidas. Es decir, se desea distribuir la probabilidad de la salida del modelo para que no esté concentrada en un rango de valores muy específicos y que de esta forma exista una mayor variabilidad en la probabilidad, lo que ayudaría a la red estudiante a generalizar el conocimiento en lugar de ajustarse a datos puntuales.

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \tag{2.12}$$

La probabilidad de cada clase q_i se modifica por medio de la variación de la Temperatura T, que es un parámetro utilizado para suavizar la distribución de probabilidad de las salidas de la red grande; [61] explica que mientras mayor sea la temperatura más suave va a ser la distribución y comenta que una vez suavizada la información de la salida de la red maestra se entrena a la red estudiante y al final del proceso se compara la salida de la red madre con la de la red hija con el fin de medir la precisión del aprendizaje.

2.5. Estado del arte

Debido a que durante los último años existe la tendencia de reducir el tamaño de las redes neuronales es que se han desarrollado aplicaciones que facilitan la reducción de las mismas. En este apartado se pretenden mencionar algunas de las aplicaciones con el fin de evidenciar el estado del arte en esta materia.

2.5.1. TensorFlow Model Optimization Toolkit

TensorFlow Model Optimization Toolkit es una biblioteca desarrolla para optimizar modelos de Inteligencia Artificial de Tensorflow. Para ello, ofrece APIs que facilitan la implementación de alguna técnica de reducción sin necesidad de tener un conocimiento profundo en el área. [62]

A través de esta biblioteca se puede reducir el costo de inferencia de y el tiempo de latencia, desarrollar modelos donde se puede restringir fácilmente el consumo de energía, el uso del almacenamiento, entre otras características. Además, permite optimizar el modelo con base en los nuevos aceleradores de propósito especial. [62]

Implementa técnicas como la cuantización y el podado, en donde, en algunos casos, es posible reducir el tamaño del modelo sin necesidad de tener datos de entrenamiento. [62]

2.5.2. TensorFlow Lite

TensorFlow Lite es un conjunto de herramientas que pretenden colaborar con la implementación de modelos de AA en dispositivos de bajo coste (como los son sistemas IoT y los celulares). Para ello, facilitan bibliotecas que permiten elegir un modelo TensorFlow Lite ya existente, convertir un modelo complejo en un modelo TensorFlow Lite, implementar el modelo dentro de un dispositivo móvil y optimizar el funcionamiento del modelo y del hardware de empotramiento. [62]

Además, llevan a cabo tareas complicadas como los es la aceleración de hardware y se encuentra disponible en las plataformas iOS, Android, Linux y plataformas especializadas de algunos microcontroladores. [62]

Capítulo 3

Metodología

3.1. Descripción de la metodología utilizada

Con el fin de estructurar el diseño realizado en este documento, se tomo como base la metodología expuesta por Ulrich y Eppinger en su libro *Diseño y desarrollo de productos* [63]. Esta metodología consta de las siete etapas que se muestran en la Fig. 3.1.

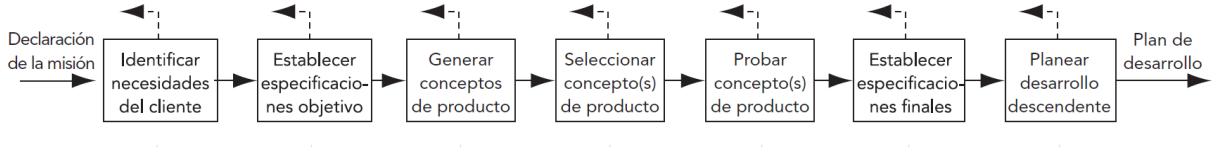


Figura 3.1: Fases de la metodología de Ulrich y Eppinger. Fuente: [63]

Sin embargo, debido a la naturaleza propia de este trabajo, se cambiaron algunas aspectos para garantizar que el proyecto de investigación pueda ser desarrollado. Por lo tanto, la modificación consta de las siguientes partes: identificación de las necesidades del cliente, establecimiento de las especificaciones objetivo, generación de conceptos, selección de conceptos, implementación de los conceptos y validación de la solución. Estos cambios pueden apreciarse en la Fig. 3.2. Cabe mencionar, que como otros procesos de diseño ingenieril, el desarrollo de este proyecto es iterativo y en cualquier instante se admite la posibilidad de retornar a una etapa previa de la metodología.

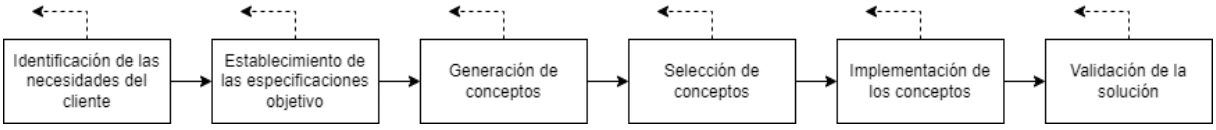


Figura 3.2: Adaptación de la metodología de Ulrich y Eppinger.

3.2. Relación entre los objetivos específicos y las etapas de la metodología

Con el fin de contextualizar al lector con la importancia de tomar en consideración las etapas de la metodología planteada, se desarrolla la relación propuesta en la Fig. 3.3. De ella se denota la relación directa (pero no absoluta) de los objetivos planteado y las fases de la metodología.

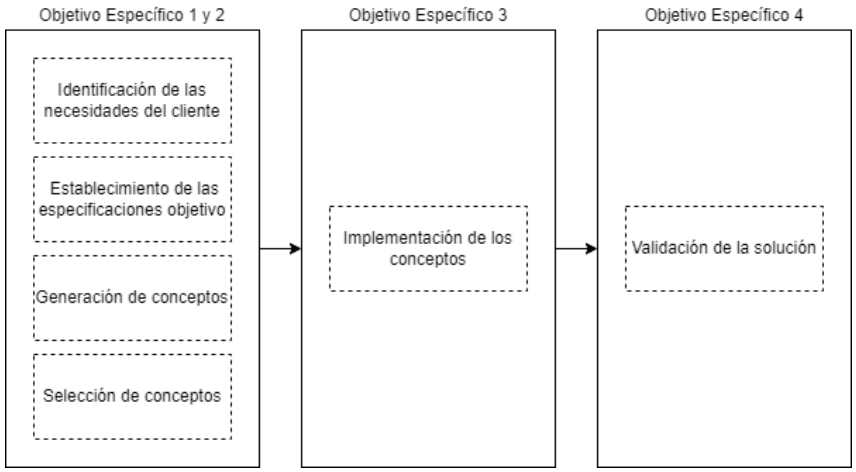


Figura 3.3: Relación de la metodología de con los objetivos específicos.

En primera instancia se aprecia que: al determinar el algoritmo de detección automático de personas para ser usado como red maestra y al definir el algoritmo de entrenamiento maestro-estudiante, se debe primeramente identificar las necesidades que surjan de las conversaciones con el cliente durante todas las reuniones. Luego, se deben establecer métricas que sirvan como medida de cumplimiento de las necesidades y de esta manera, posterior a una búsqueda exhaustiva de fuentes de información, poder seleccionar los conceptos más adecuados para cada objetivo.

Por otra parte, el tercer objetivo específico, que plantea desarrollar un programa computacional que permita implementar el algoritmo de entrenamiento maestro-estudiante para la detección automática de personas, se relaciona directamente con la fase de implementación de los conceptos, es decir, del resultado de la selección de la red maestra y el método de entrenamiento maestro-estudiante se debe desarrollar computacionalmente un programa que logre cumplir con las necesidades identificadas del cliente.

Por último se debe validar el funcionamiento y rendimiento del detector automático de personas con base en las características del eventual sistema embebido a utilizar. Lo cual tiene una relación totalmente directa con lo que se plantea en la última etapa de la metodología.

3.3. Identificación de las necesidades

Parte fundamental de todo diseño de un producto consiste en entender los requerimientos o necesidades del cliente, esto comúnmente se realiza por medio de conversaciones o entrevistas con el fin de poder abstraer la mayor cantidad de información posible. En diversas ocasiones estas necesidades no son del todo claras para los mismos clientes por lo que se debe pasar por una etapa de identificación de las mismas. En el caso de este trabajo, se tuvieron reuniones y entrevistas con el cliente con el fin de determinar los atributos deseados para la solución del problema. A modo de ejemplo, en la Tabla 3.1 se muestra un resumen de la última entrevista realizada al cliente.

Tabla 3.1: Resumen de la entrevista estudiante-cliente

N°	Pregunta	Respuesta
1	¿En sus palabras cuál es el fin del proyecto?	Lograr que una red más grande, le enseñe (o destile conocimiento) a una red más pequeña sin perder radicalmente su rendimiento. Con el fin de poder meter la chiquita a otro lado
2	¿Qué cosas necesitan para decir que como producto ya está terminado el proyecto?	De un modelo inicial (referencia), que no tiene que ser creado desde cero (puede ser modelos ya preentrenados con COCO), hacer un ajuste al tipo de datos que tengo, tener cuidado de que no haya una contaminación cruzada de los datos (que haya una separación entre entrenamiento y testeo de los datos) y que la nueva red que se proponga, que se demuestre una mejora en términos de espacio en memoria que la red necesita y los tiempo de ejecución, pero sin tener un deterioro muy considerable de su rendimiento (mAP)
3	¿Cuál es propósito de tener un conjunto de datos con la vista superior?	La idea es atacar de menor manera la privacidad de las personas con este tipo de fotos, ya que cuesta más verles las caras a las personas, lo cual, sería lo que eventualmente se haría en el laboratorio. La idea es ser lo menos intrusivos posible con estos métodos de vigilancia.
4	¿Desea que se use una IA maestra con el mejor mAP o consideraría utilizar otra que sirva de base para personas que eventualmente agarren el trabajo?	No hay interés de que sea lo más fácil de entender para otras personas. Se quiere que se base la decisión en una motivación técnica. Eso sí, se quiere que sea detección de objetos, pero tipo cajas (boxes).
5	¿Por qué la destilación de conocimiento es el tipo de entrenamiento maestro-estudiante que considera mejor?	Es un tipo de método que se ha venido utilizando bastante, no es el único, existe: dropout, punning, la idea de la destilación es que la información que se adquirió al entrenar la maestra no se pierda y no se deba empezar desde cero. La idea es que ninguna de las dos necesite empezar desde cero. Lea el paper de Hinton
6	¿Cuál es el rango de presupuesto para el sistema embebido?	La tarjeta más cara que tienen ahora ronda los 150.000 colones.
7	¿Qué otro aspecto se debe considerar para validar el funcionamiento del sistema?	Se podría utilizar el consumo de potencia, pero este proyecto no está enfocado en pasar el sistema a un embebido. Con solo memoria y tiempo de inferencia y el mAP es más que suficiente.
8	¿Cuál es el producto que desean tener al final?	Un reporte técnico que explique la red maestra que se escoge y las razones de su decisión, la técnica utilizada para el entrenamiento maestro-estudiante, la metodología utilizada y la red final. Además, todo el código relacionado. La idea es que otro estudiante agarre todo y lo pueda usar el siguiente semestre.

A raíz de estas conversaciones se identificaron las necesidades presentes en la Tabla 3.2. Cabe resaltar que [63] recomienda realizar una jerarquización de las mismas, pero se considera que esta etapa no es necesaria, ya que, las necesidades identificadas poseen características muy diferentes entre si, por lo que se optó por procesarlas directamente de la conversación.

Tabla 3.2: Necesidades identificadas del proceso de entrevista y de reuniones previas con el cliente

N°	Necesidad	Importancia
1	El SD detecta automáticamente a personas presentes en una imagen.	5
2	El SD entrena una red neuronal estudiante a partir de una red neuronal madre.	5
3	El SD disminuye la memoria necesaria para la detección automática de personas sin disminuir radicalmente su rendimiento.	5
4	El SD disminuye el tiempo de ejecución necesario para la detección automática de personas sin disminuir radicalmente su rendimiento.	5
5	El SD debe ser entrenado con el set de datos proporcionado por SIPLAB	5
6	El SD utiliza la detección de objetos por cajas para detectar automáticamente a las personas.	5
7	El SD utiliza un modelo predefinido como base para la IA maestra.	4
8	El SD permite ser eventualmente utilizado por un sistema embebido que esté dentro del presupuesto del cliente.	5
9	En esta etapa el SD no se implementa en un sistema embebido.	5
10	El SD es presentado por medio de código computacional.	5
11	El SD utiliza la validación cruzada para su entrenamiento.	4

*SD: sistema a diseñar

Además, se conversó con el cliente para que calificara la importancia de cada una de ellas, tomando como base la calificación presente en la Fig. 3.4. De este proceso se resaltan los siguientes puntos:

- El detector automático debe preservar la métrica mAP mientras que reduzca el tiempo de inferencia y la memoria consumida al correrse.
- La solución debe de realizar la detección por medio de cajas. Esta distinción es importante ya que existen otras técnicas que detectan objetos.
- El detector debe ser capaz de detectar personas desde una vista superior dado que el conjunto de datos que debe ser utilizado posee imágenes con esta connotación.
- El producto final debe cumplir con las características del eventual sistema embebido por utilizar y las características de este hardware están limitadas por el presupuesto del cliente.
- El cliente recomienda utilizar un modelo preentrenado como base para la red neuronal maestra, no obstante, no lo considera como algo imprescindible.
- Se recomienda que la validación de los resultados se haga por medio de algún método de validación cruzada, pero, no se considera totalmente necesario.

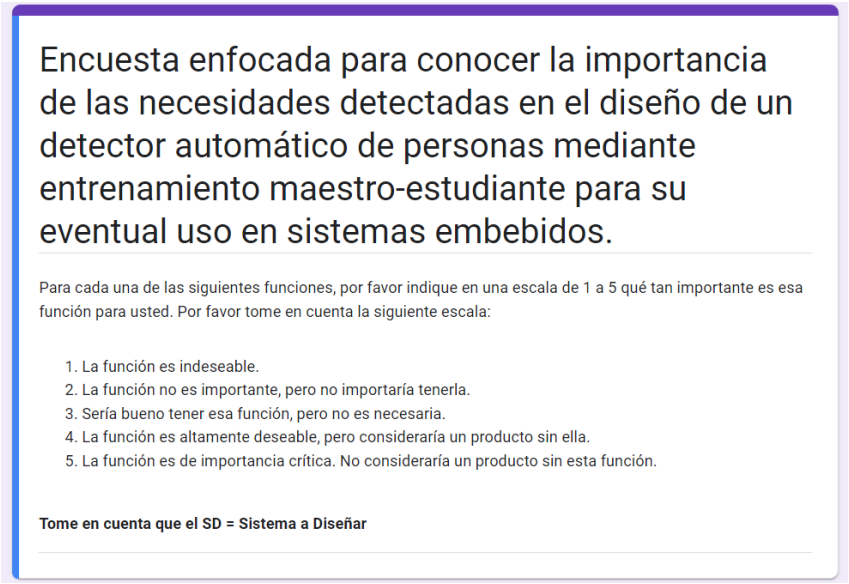


Figura 3.4: Encabezado de la encuesta proporcionada al cliente.

3.4. Establecimiento de las especificaciones

Una vez identificadas las necesidades del cliente se requiere una manera objetiva para poder cuantificar en qué momento se dan sus cumplimientos, para ello, se deben establecer las especificaciones. Como comenta [63] las especificaciones no indican cómo desarrollar una solución que tome en cuenta las necesidades de los clientes, pero sirven como base para poder saber qué pasos tomar para cumplirlas.

Por lo tanto, en la Tabla 3.3 se muestran las métricas con las que se considera que se puede medir el cumplimiento de cada una de las necesidades.

Tabla 3.3: Métricas de las necesidades identificadas

N°	Núm de necesidad	Métrica	Importancia
1	1	Capacidad de detectar personas	5
2	2	Existencia de entrenamiento maestro-estudiante	5
3	3, 8	Memoria del programa	5
4	3, 4	mAP	5
5	4	Tiempo de inferencia	5
6	5	Entrenamiento con el set de datos del SIPLAB	5
7	6	Es un detector de objetos por cajas	5
8	7	Es un modelo predefinido	4
9	8	Precio del sistema embebido	5
10	10	Es un código computacional	5
11	11	Utiliza validación cruzada	4

Una vez definidas las métricas, se establecen valores ideales y marginales para cuantificar

el nivel de cumplimiento de cada necesidad. El proceso de obtención de estos valores se da por medio de consultas bibliográficas y consideraciones del cliente.

En el caso del mAP (métrica 4), el cliente expresó el deseo de que luego del entrenamiento maestro-estudiante, su valor no debe reducirse más allá de un valor porcentual respecto a su valor inicial. De manera similar, la memoria del programa y el tiempo de inferencia (métrica 3 y 5) debe reducirse como mínimo un valor porcentual respecto a su valor de inicio luego del entrenamiento maestro-estudiante. Por otro lado el precio del eventual sistema embebido está limitado por el presupuesto del laboratorio de investigación, por lo que también fue proporcionado por el cliente. Además, es importante denotar que este es un proceso iterativo y que los valores finales presentes en la Tabla 3.4 para estas características son el resultado de la última conversación con el cliente.

Tabla 3.4: Valores marginales e ideales de las métricas

N°	Núm de necesidad	Métrica	Imp	Unidades	Valor Marginal	Valor Ideal
1	1	Capacidad de detectar personas	5	Binaria	Sí	Sí
2	2	Existencia de entrenamiento maestro-estudiante	5	Binaria	Sí	Sí
3	3, 8	Memoria del programa	5	GB	90 % del valor inicial >x	90 % del valor inicial >x
				Cantidad de parámetros	90 % del valor inicial >x	90 % del valor inicial >x
4	3, 4	mAP	5	%	x >90 % del valor inicial	x >90 % del valor inicial
5	4	Tiempo de inferencia	5	s	90 % del valor inicial >x	90 % del valor inicial >x
6	5	Entrenamiento con el set de datos del SIPLAB	5	Binaria	Sí	Sí
7	6	Es un detector de objetos por cajas	5	Binaria	Sí	Sí
8	7	Es un modelo predefinido	4	Binaria	No	Sí
9	8	Precio del sistema embebido	5	Colones	200 000> x	150 000> x
10	10	Es un código computacional	5	Binaria	Sí	Sí
11	11	Utiliza validación cruzada	4	Binaria	No	Sí

3.5. Generación de conceptos

En esta sección se detallan todos los conceptos analizados para la solución planteada. [63] la divide en los siguientes cinco apartados: aclarar el problema, búsqueda externa de conceptos, búsqueda interna de conceptos, exploración sistemática de los conceptos y reflexión sobre el proceso. Sin embargo, se adecuaron estos pasos de la metodología, por lo que, se desarrollaron las siguientes dos etapas: una descomposición funcional del problema (enfoque de la solución) y una búsqueda interna y externa de cada subproblema.

3.5.1. Descomposición funcional del problema

Con el fin de reducir la complejidad del desarrollo de la solución, se subdividió el problema en subsecciones. Este proceso fue denominado como enfoque de la solución y consta de seis partes tal y como se aprecia en la Fig. 3.5. Sus partes se describen a continuación:

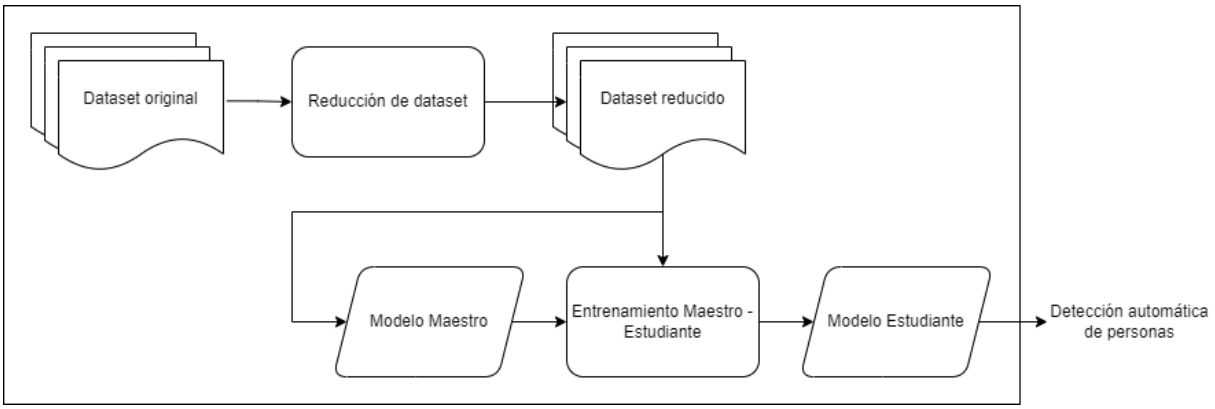


Figura 3.5: Enfoque de la solución.

Conjunto de datos original

Esta etapa compete al conjunto de datos proporcionado por el laboratorio SIPLAB, el cual posee videos capturados durante tres días con tres distintas cámaras siguiendo la distribución presente en la Fig. 3.6. La resolución original de los videos es de 1920x1080 píxeles a 30 fps, pero el conjunto de datos proporcionado tiene una resolución de 960x540 píxeles a 20 fps.

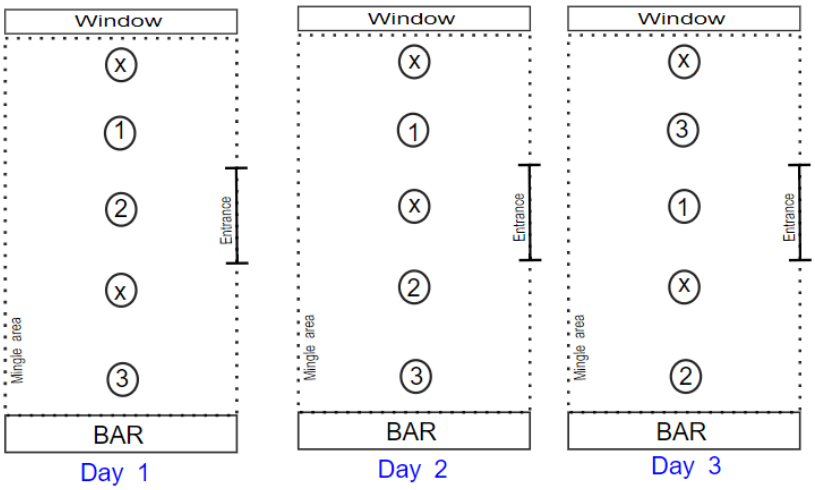


Figura 3.6: Distribución de las cámaras durante los tres días de captura. Fuente: [64]

Además, las anotaciones brindadas fueron realizadas para cada frame de los videos, en

donde se colocaron las posiciones dimensionales del cuadrado que encierra a cada una de las personas presentes en él.

Reducción del conjunto de datos

Tomando en consideración la similitud que pueden tener los datos proporcionados por SIPLAB, esta sección se basa en la teoría descrita en el Capítulo 2 respecto al sobreentrenamiento de los algoritmos de Inteligencia Artificial. Por lo tanto, se planteó una reducción del conjunto de datos con el fin de facilitar la generalización de las redes neuronales convolucionales entrenadas.

Conjunto de datos reducido

Esta etapa es el resultado de la reducción del conjunto de datos luego de filtrar todas aquellas imágenes que se consideren como inadecuadas para entrenar la red neuronal. Con ello se pretende evitar el sobreentrenamiento de la solución y reducir el coste computacional requerido para llevarlo a cabo.

Modelo Maestro

Este proyecto pretende usar como referencia una red neuronal convolucional funcional tomada de algún detector de objetos previamente desarrollado. Por lo tanto, el resultado de esta etapa se deriva de un estudio bibliográfico que permita utilizar la red más adecuada para las necesidades que plantea el cliente. Además el modelo seleccionado debe adecuarse al conjunto de datos proporcionado por SIPLAB por medio de un entrenamiento que ajuste los pesos de la red.

Técnica de entrenamiento Maestro-Estudiante

De manera similar al modelo maestro, la técnica de entrenamiento maestro-estudiante se deriva de una investigación bibliográfica respecto a los algoritmos existentes para llevarlo a cabo; se plantea seleccionar una que cumpla con las necesidades expresadas por el cliente. Esto se realiza con el fin de poder adecuar el modelo maestro a un modelo con menor coste computacional requerido para su funcionamiento y con parámetros de rendimiento similares.

Modelo Estudiante

Es el modelo resultante del entrenamiento maestro-estudiante. Con este modelo se pretende dejar las bases para que posteriores investigadores realicen la implementación de la

detección automática de personas en un sistema embebido, por lo tanto, con este modelo se deben de realizar las pruebas que cumplan con las necesidades requeridas por el cliente y que a su vez den validez de que el modelo puede ser implementado en un futuro en un sistema embebido.

3.5.2. Búsqueda interna y externa de conceptos

Una vez desarrollada la descomposición funcional del problema se debe realizar una búsqueda de posibles soluciones con base en la experiencia del diseñador y en referencias bibliográficas para todos aquellos subproblemas donde su solución no es tan directa. En el caso de este trabajo, esta fase compete a la determinación del modelo maestro y a la definición del entrenamiento maestro-estudiante. En la Tabla 3.5 se muestran todos los conceptos de la solución resultantes luego de la búsqueda interna y externa; en el Capítulo 2 se detallan las bases teóricas del funcionamiento de los modelos y las técnicas.

Tabla 3.5: Resultado de la búsqueda interna y externa de conceptos

N°	**Subproblema**	
	Modelo Maestro	**Técnica de Entrenamiento Maestro-Estudiante**
1	YOLO	Podado
2	SSD	Cuantización
3	Faster R-CNN	Compresión
4	Mask R-CNN	Destilación de conocimiento
5	RF-Next	-

3.6. Selección de conceptos

Una vez establecidos todos los posibles conceptos, se debe realizar un análisis cualitativo de los posibles prospectos para elegir la mejor solución para cada una de las subtareas. [63] muestra un proceso de dos etapas para esta sección, compuesto por una filtración y luego una selección, sin embargo, se realizó una modificación a esta fase por lo que se propone únicamente una etapa de selección de donde se evaluará para cada métrica a cada uno de los conceptos con un valor de uno a cinco, siendo uno un valor muy por debajo de la media, dos un valor debajo de la media, tres un valor cercano a la media, cuatro un valor encima de la media y cinco un valor muy por encima de la media. El resultado de esta etapa se encuentra en el Capítulo 4 de este documento.

3.6.1. Modelo Maestro

En el caso del modelo maestro, para la selección del concepto ganador se deben en tomar en consideración características como la capacidad de detección de objetos, su valor de mAP, el tiempo de inferencia, entre otros aspectos. Por lo tanto, en la Tabla 3.6 se muestran cuáles de las métricas identificadas aplican para el subproblema.

Tabla 3.6: Métricas tomadas en consideración para la selección del concepto del modelo maestro

N°	Métrica
1	Capacidad de detectar persona
2	mAP
3	Tiempo de inferencia
4	Entrenamiento con el conjunto de datos de SIPLAB
5	Es un detector de objetos por cajas
6	Es un modelo predefinido

3.6.2. Técnica de Entrenamiento Maestro-Estudiante

Por otro lado, la técnica de entrenamiento maestro-estudiante debe cumplir con las métricas presentes en la Fig. 3.7.

Tabla 3.7: Métricas tomadas en consideración para la selección del concepto del la técnica de entrenamiento maestro-estudiante

N°	Métrica
1	Existencia de un entrenamiento maestro-estudiante
2	mAP
3	Tiempo de inferencia
4	Memoria del programa

3.7. Implementación de los conceptos

Posterior a la definición de los conceptos de los subproblemas, se debe llevar a cabo su implementación. El Capítulo 4 muestra a detalle los pasos llevados a cabo para el desarrollo de la solución. En este caso el producto consiste de un programa computacional

que cumple con las métricas descritas en esta sección y se sustenta en la teoría detallada en el Capítulo 2.

3.8. Validación de la solución

Por último, se debe velar que la solución implementada cumpla con los requerimiento identificados, por lo tanto, se deben definir pruebas para validar el funcionamiento de la solución. Para ello, se definen las pruebas mostradas en la Tabla 3.8 con el fin de obtener el valor de mAP de los modelos. En esta tabla se aprecia que se toma al conjunto de datos de entrenamiento y al de validación como factores de influencia que pueden afectar el resultado de la muestra. Esto resulta importante debido a que se desea que los resultados tengan un error asociado.

Tabla 3.8: Pruebas planteadas para la obtención del métrica mAP

Descripción: Obtención del mAP		
Se realizarán pruebas con el cd de validación para obtener el valor mAP del modelo. Para ello se utilizarán las imágenes del conjunto de datos y se calculará la métrica a raíz de los resultados. Este proceso se realizará en seis ocasiones contemplando las distintas maneras de obtener la métrica al hacer la validación cruzada.		
Factores de influencia		**Variables de muestra**
Cd de entrenamiento	**Cd de validación**	**mAP**
Día 1	Día 2	
Día 1	Día 3	
Día 2	Día 1	
Día 2	Día 3	
Día 3	Día 1	
Día 3	Día 2	

cd: conjunto de datos

De manera similar se tiene la Tabla 3.9, en donde los factores de influencia también son los conjuntos de datos, pero, en este caso las muestras por realizar corresponden al tiempo de inferencia y la memoria utilizada del modelo.

Tabla 3.9: Pruebas planteadas para la obtención del tiempo de influencia y uso del memoria del modelo

Descripción: Obtención del tiempo de inferencia y la memoria del modelo estudiante					
Se realizarán pruebas con el conjunto de datos de validación para calcular el tiempo que tarda el modelo en hacer inferencias y el consumo computacional durante estos instantes. Esto se realizará para cada una de las imágenes del cd y con ello se promediarán los resultados obtenidos para tener un valor con un respectivo error asociado. Además, el proceso se realizará en seis ocasiones, contemplando las distintas maneras de obtener las métricas al hacer la validación cruzada. En el caso de la memoria consumida por el programa se medirá el uso de CPU y GPU, y la cantidad de parámetros del modelo.					
Factores de influencia		**Variables de muestra**			
Cd de entrenamiento	**Cd de validación**	**Tiempo de inferencia**	**Memoria del programa**		
			CPU	**GPU**	**Parámetros del modelo**
Día 1	Día 2				
Día 1	Día 3				
Día 2	Día 1				
Día 2	Día 3				
Día 3	Día 1				
Día 3	Día 2				

*cd: conjunto de datos

Luego, por medio de las variables obtenidas a raíz de las pruebas planteadas, se debe cerciorar que el entrenamiento maestro-estudiante cumple a cabalidad con las métricas definidas, así que, en la Tabla 3.10 se plantean las pruebas por realizar para poder validar el sistema diseñado.

Tabla 3.10: Pruebas planteadas para la comparación del modelo maestro y el modelo estudiante

Descripción: Comparación porcentual del modelo maestro y el modelo estudiante				
Se realizará una comparación porcentual de las métricas mAP, tiempo de inferencia y memoria del programa del modelo maestro seleccionado respecto al modelo estudiante resultante, con el fin de verificar que el entrenamiento maestro-estudiante realizado cumple con los valores objetivo impuestos por el cliente. Además, se realizará una validación cruzada con el fin de darle una fiabilidad probabilística a los resultados.				
Factores de influencia		**Variables de muestra**		
Cd de entrenamiento	**Cd de validación**	**mAP**	**Tiempo de inferencia**	**Memoria del programa**
Día 1	Día 2			
Día 1	Día 3			
Día 2	Día 1			
Día 2	Día 3			
Día 3	Día 1			
Día 3	Día 2			

cd: conjunto de datos

Por último, es necesario que el modelo resultante posea las capacidades de implementarse en un futuro en un sistema embebido. Para ello, se debe tomar en consideración el presupuesto del laboratorio para la compra del hardware y hacer una comparación en términos de uso del CPU, GPU y tamaño de la memoria del sistema embebido y los valores requeridos en estos parámetros para que el modelo final pueda ser implementado.

Capítulo 4

Propuesta de Diseño

Esta sección consiste en la descripción del diseño propuesto para un detector automático de personas mediante un entrenamiento maestro-estudiante para su eventual uso en sistemas embebidos. Contiene las siguientes partes: reducción del conjunto de datos, elección del modelo maestro, entrenamiento del modelo maestro, elección de la técnica de entrenamiento maestro-estudiante e implementación del entrenamiento maestro-estudiante.

4.1. Reducción del conjunto de datos

Tal y como se comenta en el Capítulo 2 algunos modelos se sobreentrenan en lugar de aprender las generalidades del conjunto de datos con el que se entrenan, por lo que presentan inconvenientes a la hora de exponerse a otros datos. En el caso de este trabajo, se tiene el enfoque de utilizar el modelo resultante para eventualmente implementarlo en un sistema embebido para hacer una detección automática de personas. Por lo tanto, resulta vital validar que el producto tenga la capacidad de detectar personas que nunca ha visto en otras instancias.

Tomando en consideración lo expuesto, se analiza el conjunto de datos proporcionado por el laboratorio de investigación SIPLAB para buscar que los datos con los que se debe entrenar el modelo tengan una variabilidad tal que no se sobreentrenen los modelos. Como el conjunto de datos corresponde a nueve videos de 30 minutos a una frecuencia de 20fps, se realiza un análisis de las imágenes de cada uno con el fin de determinar si existe una variación relevante entre los frames de los videos debido a que se tienen dudas de qué tanto puede variar un frame consecutivo a otro considerando la tasa de captura.

4.1.1. Obtención del porcentaje de similitud de los videos

Para ello, en primera instancia se crea un código computacional capaz de generar imágenes a partir de videos, tomando en consideración aspectos como: cada cuántos frames se desea guardar una imagen (distancia entre frames), el día y la cámara del video a convertir, el alto y el ancho con el que se desea guardar cada imagen. El Algoritmo 1 muestra en pseudocódigo la función desarrollada para este propósito, conocido como *video2frames*. Una vez generado este código, es posible convertir en imágenes los videos de las tres cámaras capturados durante los tres días para posteriormente analizarlos.

Algoritmo 1: video2frames

```
Input: distancia entre frames, día, cámara, nueva altura, nuevo ancho
Output: frames, dirección

// Genera el directorio donde se van a guardar los frames
dirección = día + cámara + distancia entre frames
makedir(dirección)

// Recorre todos los frames del video
while cada frame do
    // Verifica que se cumpla el salto entre frames deseado
    if número frame actual % distancia entre frames == 0 then
        // Cambia el tamaño del frame
        resize(frame, nueva altura, nuevo ancho)

        // Guarda el frame en la carpeta creada
        saveImg(frame, dirección)
    end
end
```

Como segundo paso, se genera el Algoritmo 2, capaz de cargar las imágenes de una carpeta, calcular la similitud entre todos los frames consecutivos y guardar los resultados. Esta función toma en consideración que las cámaras estaban fijas a la hora de capturar los videos, por lo que calcula la diferencia presente entre los frames consecutivos del conjunto de datos para cuantificar su variabilidad. Sabiendo que la cámara no se mueve, si existe algún cambio entre dos imágenes al calcular la diferencia entre ellas se tiene como resultado en los píxeles donde existe la variación un valor distinto de cero, pero, si no existe cambio entre ellas se tiene como resultado una matriz con ceros. Por lo tanto, contando la cantidad de píxeles distintos de cero es posible establecer cuál es el porcentaje de variación entre esos frames. Sin embargo, la diferencia puede deberse al ruido presente

en las cámaras, por lo que, se añade una etapa de binarización respecto a un umbral con el fin de eliminarlo.

Algoritmo 2: frames2diff

Input: día, cámara, distancia entre frames, umbral
Output: csvfile

```
// Genera el archivo donde se van a guardar los datos
csvfile = dia + cámara + distancia entre frames + umbral
makefile(csvfile)

// Carga las imágenes de la carpeta
dirección = día + cámara + distancia entre frames + umbral
dataset = loadfile(dirección)

// Recorre todos las imágenes cargadas
while cada frame do

    // Carga la primera imagen
    img1 = dataset(número frame actual)

    // Carga la imagen consecutiva
    img2 = dataset(número frame actual + 1)

    // Convierte las imágenes de color BGR a escala de grises
    img gray1 = bgr2gray(img1)
    img gray2 = bgr2gray(img2)

    // Calcula la diferencia absoluta entre las imágenes
    img diff = absdiff(img gray1, img gray2)

    // Binariza la diferencia absoluta entre las imágenes
    img binaria = bin(img diff, umbral)

    // Cuenta la cantidad de píxeles que cambiaron (!= 0)
    cant pixeles0 = countZero(img binaria)

    // Calcula el porcentaje de similitud entre las imágenes
    % variación imgs = cant pixeles0 / size(img1) *100

    // Guarda el porcentaje de variación en el archivo .csv
    saveValue( % variación imgs, csvfile)
end
```

Por último, una vez que se determina el porcentaje de similitud entre todos los frames de un video, se pueden utilizar estos datos para promediar la variación presente en un conjunto de datos. El Algoritmo 3 muestra un pseudocódigo que calcula la variación promedio y la desviación estándar del video de una cámara grabado durante un respectivo día y gráfica los resultados en función de la distancia entre imágenes.

Algoritmo 3: plot diff

Input: día, cámara, umbral, lista de distancia entre frames
Output: mean list, std list, img gráfica

```
// Recorre la lista de distancia entre frames
for df in lista de distancia entre frames do

    // Carga los datos del .csv según la distancia entre frames
    dirección = día + cámara + df + umbral
    data = loadfile(dirección)

    // Calcula el promedio
    promedio = mean(data)

    // Calcula la desviación estándar
    desv = std(data)

    // Guarda los valores en una lista
    mean list = append(promedio)
    std list = append(desv)

end

// Grafica en una imagen los promedios
img = plot(mean list, lista de distancia entre frames)

// Guarda la imagen
saveImg(img)
```

4.1.2. Análisis de la variabilidad del conjunto de datos

En este proceso se desea analizar cuál es el porcentaje del conjunto de datos que se debe utilizar para entrenar los modelos. Específicamente, se busca determinar cada cuántos frames se debe guardar una imagen al convertir los videos en imágenes. Inicialmente, si se busca utilizar todos los frames, se tendría que la distancia entre frames es uno, por lo

que existirían 324.000 imágenes en total según la Ec. 4.1.

$$\begin{aligned} total &= 9\ videos \cdot 30min \cdot 60s \cdot 20fps \\ total &= 324\ 000\ frames \\ total &= 36\ 000\ frames/video \end{aligned} \tag{4.1}$$

Así que, con todas las imágenes del conjunto de datos, al utilizar los Algoritmos 1, 2 y 3 se obtienen para los nueve videos los porcentajes de similitud presentes en la Tabla. 4.1.

Tabla 4.1: Porcentaje de similitud del conjunto de datos

Día	1	1	1	2	2	2	3	3	3
Cámara	1	2	3	1	2	3	1	2	3
Similitud (%)	61,1882	56,0373	76,3588	66,0064	67,2594	55,4666	63,1435	60,6777	66,3123
Error (%)	8,8167	9,0986	8,5863	7,4586	8,3119	9,7784	8,8618	8,7066	7,6141

Sin embargo, estos valores pueden estar influenciados por el ruido presente en los videos, por lo que, se diseña un código capaz de cuantificar el histograma promedio de un video para evaluar el impacto en los resultados. El Algoritmo 4 muestra dicha función.

Algoritmo 4: avgHist

Input: día, cámara, distancia entre frames, umbral
Output: img Hist

```
// Carga las imágenes de la carpeta
dirección = día + cámara + distancia entre frames + umbral
dataset = loadfile(dirección)

// Recorre todos las imágenes cargadas
while cada frame do
    // Calcula el histograma de la diferencia entre imágenes
    hist = imgHist(absdiff(img frame actual, img frame siguiente))

    // Suma los valores de cada histograma
    avgHist += hist
end

// Calcula y guarda el histograma promedio
avgHist = avgHist / len(dataset)
saveImg(avgHist)
```

Con base en este algoritmo se analiza para cada video su histograma promedio. En el caso del video de la primera cámara tomado durante el primer día, se tiene el histograma presente en la Fig. 4.1, donde se ve que existe un alto porcentaje de píxeles con un valor entre cero y cinco, lo que significa que la variación entre los frames consecutivos no es tan significativa.

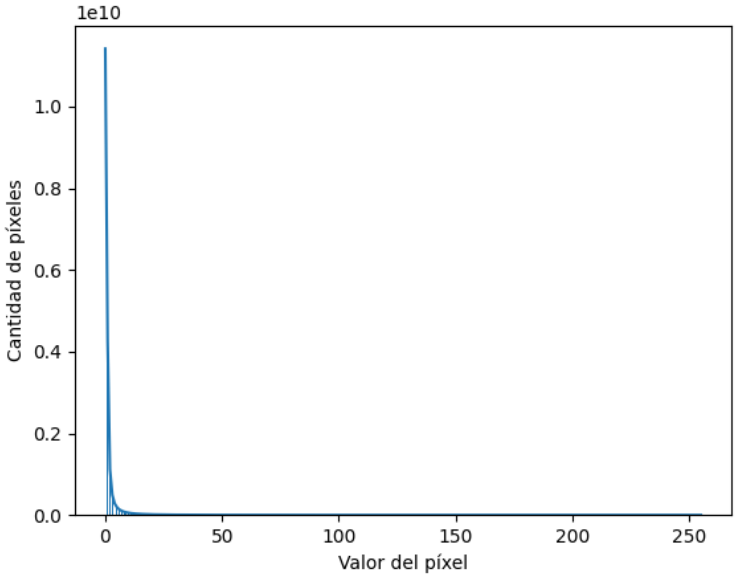


Figura 4.1: Histograma promedio del video grabado por la primera cámara durante el primer día.

Esto tiene sentido según los fps del video, no obstante, puede estar influenciado por alteraciones entre una imagen y otra debido al ruido, por lo cual, se pretende binarizar la imagen de la diferencia por medio de un umbral distinto de cero. Por lo que, para que un píxel sea reconocido como una variación debe tener un valor mayor a este umbral.

Tomando en consideración el histograma de la Fig. 4.1 y el de los otros ocho videos (presentes en el Apéndice A), se decidió utilizar dos umbrales: el primero equivalente a uno y el segundo a cinco. Estos valores pretenden eliminar las variaciones pequeñas sin influenciar mucho en los cambios reales de las imágenes. En las Tablas 4.2 y 4.3 se muestran los porcentajes de similitud del conjunto de datos para cada umbral.

Tabla 4.2: Porcentaje de similitud del conjunto de datos al usar un umbral de binarización de uno

Día	1	1	1	2	2	2	3	3	3
Cámara	1	2	3	1	2	3	1	2	3
Similitud (%)	83,9937	81,8511	91,7475	85,6164	84,0485	79,2232	83,5632	82,1222	85,1823
Error (%)	5,1450	5,3468	4,7625	4,2945	5,3568	5,9821	5,2827	5,1358	4,0754

Tabla 4.3: Porcentaje de similitud del conjunto de datos al usar un umbral de binarización de cinco

Día	1	1	1	2	2	2	3	3	3
Cámara	1	2	3	1	2	3	1	2	3
Similitud (%)	95,1582	94,6403	97,6007	95,4037	93,7567	93,1300	94,1245	93,7522	94,7666
Error (%)	1,9785	2,1070	1,7595	1,9038	2,8701	2,5618	2,5303	2,3313	1,9613

A modo de resumen se ofrece la Tabla comparativa 4.4, de donde se aprecia que con las leves variaciones del umbral de binarización, el porcentaje de similitud de cada video incrementa, por lo tanto, se puede definir que, si se ignora el ruido de los videos, el conjunto de datos presenta una baja variabilidad. Así que, con el fin de evitar el sobreentrenamiento de los modelos, se propone disminuirlo.

Tabla 4.4: Tabla comparativa de la similitud del conjunto de datos respecto al umbral de binarización utilizado

Umbral	**Similitud Promedio (%)**	**Incertidumbre (%)**
0	63,6056	± 2,8694
1	84,1498	± 1,6908
5	94,7037	± 0,7500

4.1.3. Análisis de la reducción del conjunto de datos

Para disminuir el conjunto de datos se analizan variaciones de la distancia entre frames y cómo afectan la variabilidad del conjunto de datos. Por lo tanto, se propone cambiar el tamaño del conjunto de datos conforme a una de las distancias expuestas en la Tabla 4.5.

Tabla 4.5: Reducción del conjunto de datos según la distancia entre frames

Distancia entre frames	FPS	Tiempo (s)	Tamaño del conjunto de datos	Reducción (%)
1	20,000	0,05	324 000	0,000
2	10,000	0,10	162 000	50,000
5	4,000	0,25	64 800	80,000
10	2,000	0,50	32 400	90,000
20	1,000	1,00	16 200	95,000
40	0,500	2,00	8 100	97,500
80	0,250	4,00	4 050	98,750
160	0,125	8,00	2 025	99,375

En cada uno de estos casos, es necesario determinar el porcentaje de similitud de cada uno de los videos, para ello se utilizan el Algoritmo 1 para convertir el conjunto de datos original en imágenes, luego el Algoritmo 2 para calcular el porcentaje de similitud y el Algoritmo 3 para graficar los resultados. La Fig. 4.2 muestra la curva del vídeo capturado por la primera cámara durante el primer día respecto a los tres umbrales de binarización utilizados anteriormente (diríjase al Apéndice B para visualizar las curvas de los otros 8 videos).

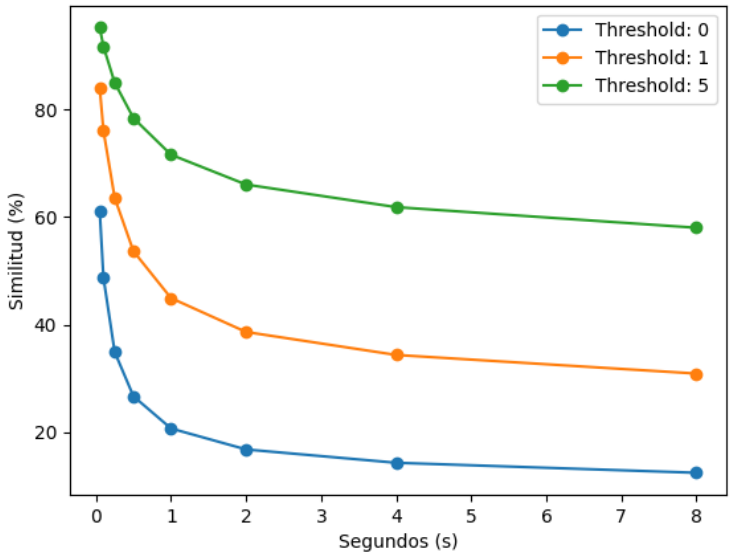


Figura 4.2: Porcentaje de similitud del conjunto de datos respecto al tiempo entre imágenes para el video capturado por la primera cámara durante el primer día.

En esta figura se aprecia como conforme se esperan más segundos para tomar otra imagen, el porcentaje de similitud entre ellas disminuye, lo cual tiene sentido dado el contexto del conjunto de datos. Además, se ve que conforme el umbral incrementa, suben los valores

del porcentaje de similitud. En este caso se toma como referencia la curva del umbral equivalente a uno, debido a que permite filtrar el ruido del conjunto de datos a la vez que influye lo menor posible en los resultados. Para dicho valor se observa que con la tasa de captura original (20fps) se tienen valores del porcentaje de similitud cercanos al 85 %, mientras que alrededor de los 0.5s la diferencia comienza a ser cercana al 50 %, lo que significa que ya una gran porcentaje de la imagen presenta una variación en su contenido, por lo cual, se analizan los resultados en la Tabla 4.6 a partir de los 0.5s (2fps).

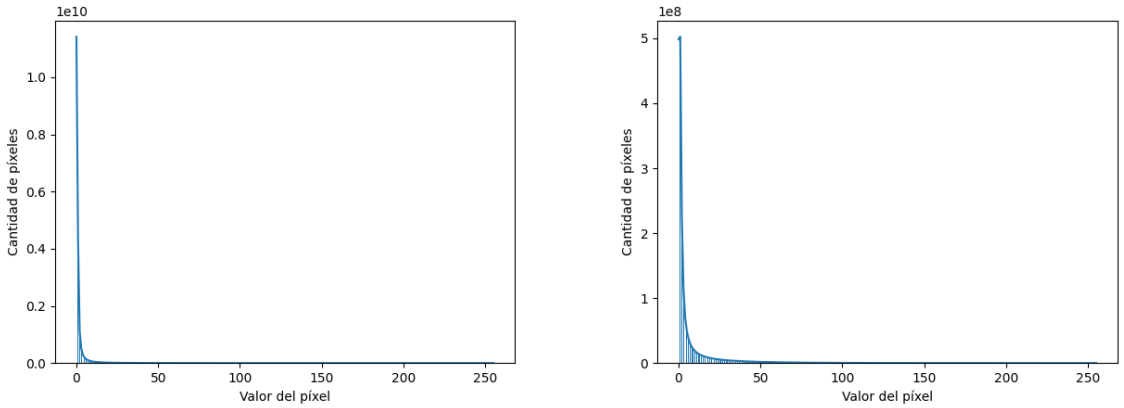
Tabla 4.6: Porcentaje de similitud según la distancia entre frames para el umbral de binarización de uno

Distancia entre frames	**FPS**	**Tiempo (s)**	**Tamaño del conjunto de datos**	**Similitud (%)**	**Error (%)**
10	2,000	0,5	32 400	56,7962	2,1924
20	1,000	1,0	16 200	48,2973	2,4019
40	0,500	2,0	8 100	41,5082	2,5100
80	0,250	4,0	4 050	36,4903	2,5669
160	0,125	8,0	2 025	32,2243	2,6492

Con base en los resultados expuestos en la Fig. 4.6 y en lo recomendado por el cliente para la reducción del conjunto de datos, se decidió utilizar una distancia entre frames de 20 (1ps), ya que, es el punto a partir del cual se tiene una variación significativa del conjunto de datos y se sigue teniendo una cantidad importante de imágenes para los futuros entrenamientos. Es decir, se reduce el conjunto de datos de lo expuesto por la Ec. 4.1 a lo mostrado por la Ec. 4.2.

$$\begin{aligned} total &= 9\ videos \cdot 30min \cdot 60s \cdot 1fps \\ total &= 16\ 200\ frames \\ total &= 1\ 800\ frames/video \end{aligned} \tag{4.2}$$

Se obtuvo el histograma resultante luego de aplicar esta reducción y a modo de ejemplo en la Fig. 4.3 se compara respecto al histograma del conjunto de datos original.



(a) Conjunto de datos original **(b)** Conjunto de datos reducido

Figura 4.3: Histograma promedio del video grabado por la primera cámara durante el primer día.

4.1.4. Coste computacional

Es importante recalcar que al reducir el tamaño del conjunto de datos no solo se disminuye la posibilidad de sobreentrenar los modelos, sino que también, baja el coste computacional requerido para entrenar los modelos. En el caso de este trabajo, se busca ajustar una red previamente entrenada y realizar un entrenamiento maestro-estudiante, que son procesos que comúnmente requieren de un alto coste computacional. Por lo que, la reducción llevada a cabo disminuye drásticamente este factor tan relevante en el ámbito del Aprendizaje Automático y por ende, posibilita llevar a cabo la tarea de mejor manera con los recursos a disposición.

4.2. Elección del modelo maestro

Para la elección del modelo maestro, en primera instancia, se realizó una búsqueda con base en fuentes recomendadas por el cliente donde existen diversidad de modelos preentrenados, entre las cuales se recalcan: PyTorch Hub [65], TensorFlow Hub [62] y Papers With Code [66]. De estas fuentes se obtuvo información del funcionamiento del estado del arte en modelos de detección de objetos. Se tomó la decisión de utilizar únicamente modelos presentes en la página Papers With Code debido a que además de otorgar códigos de modelos preentrenados funcionales, daban un respaldo bibliográfico para cada uno de ellos.

4.2.1. Características de los candidatos

Una vez definido el *nicho* de donde se planteaba obtener los candidatos para esta sección, se analizaron modelos que generan cuadros delimitadores a la hora de realizar la detección de objetos y que fueran validados por algún conjunto de datos común (o similar) para poder realizar una comparación entre ellos. En este caso se definieron cinco posibles candidatos, que fueron explicados a profundidad en el Capítulo 2. En el caso de estos modelos se analizó el tiempo de inferencia, el valor mAP resultante de la validación del modelo, el conjunto de datos con el que fue validado y las épocas de entrenamiento con el fin de comparar características del rendimiento de cada uno de los conceptos. La Tabla 4.7 muestra a detalle el resultado de este análisis.

Tabla 4.7: Características de los modelos a evaluar para la red maestra. Fuentes: [51][52][55][56][57]

Detector de objetos	**YOLOv6 v3.0**		**SSD**		**Faster RCNN**	**Mask R-CNN**	**RF-Next**
Conjunto de datos	Coco 2017 val		Coco 2015 val		Coco 2015 val	Coco 2016 val	Coco 2017 val
Modelos	YOLOv6-S	YOLOv6-L	SSD-300	SSD512	Faster RCNN	Mask R-CNN	RF-ConvNeXt
mAP	44,3	51,8	23,2	26,8	48,4	37,1	45,9
Tiempo de inferencia	2,9ms	10,3ms	-	-	200ms	200ms	-
Épocas de entrenamiento	300		240k	240k	320k	160k	-

De la Tabla 4.7 se pueden ver los resultados obtenidos por los desarrolladores luego de validar los modelos por medio del conjunto de datos de validación COCO (Common Object in Context). En este punto, cabe resaltar que a pesar de que los años del conjunto de datos son distintos, es válido comparar los resultados entre ellos debido a que no varían en grandes rasgos. Para colaborar con esta argumentación, se adjunta la Tabla 4.8 con el fin de recalcar sus similitudes. [67][68]

Tabla 4.8: Comparación entre los distintos conjunto de datos COCO. Fuentes: [67]

Característica	**COCO 2017**	**COCO 2016**	**COCO 2015**
Imágenes	330 000	330 000	328 000
Objetos anotados	2 500 000	2 500 000	2 500 000
Clases	80	80	80
Particiones	Train, Val, Test	Train, Val, Test	Train, Val, Test
Tamaño máximo imágenes	16 megapíxeles	16 megapíxeles	16 megapíxeles

4.2.2. Evaluación de los candidatos

Luego de determinar las características del rendimiento de cada modelo, se decidió realizar la selección del modelo maestro. Para ello, se evalúa en la Tabla 4.9 los criterios de

selección que el cliente proporcionó (provenientes de las métricas y especificaciones) con sus respectivos valores de importancia según las características mostradas en la Tabla 4.7. El resultado de esta evaluación pone como ganador al modelo YOLOv6.

Tabla 4.9: Evaluación y selección de los conceptos de la Red Maestra

IA Maestra			Conceptos									
Criterios de selección	Importancia	Peso %	A		B		C		D		E	
			YoloV6		Faster R-CNN		Mask R-CNN		RF-ConvNeXt-T Cascade R-CNN		SSD	
			Calificación	Evaluación ponderada	Calificación	Evaluación ponderada	Calificación	Evaluación ponderada	Calificación	Evaluación ponderada	Calificación	Evaluación ponderada
Capacidad de detectar personas	5,00	0,17	3	0,5172	3	0,5172	3	0,5172	3	0,5172	3	0,5172
mAP	5,00	0,17	5	0,8621	5	0,8621	3	0,5172	5	0,8621	1	0,1724
Tiempo de inferencia	5,00	0,17	5	0,8621	3	0,5172	3	0,5172	2	0,3448	2	0,3448
Utiliza el conjunto de datos del SIPLAB	5,00	0,17	3	0,5172	3	0,5172	3	0,5172	3	0,5172	3	0,5172
Detecta de objetos por cajas	5,00	0,17	3	0,5172	3	0,5172	3	0,5172	3	0,5172	3	0,5172
Es un modelo predefinido	4,00	0,14	3	0,4138	3	0,4138	3	0,4138	3	0,4138	3	0,4138
Puntos totales	29,00	100 %		3,69		3,34		3,00		3,17		2,48
Lugar			**1**		**2**		**4**		**3**		**5**	
¿Continuar?			Sí		No		No		No		No	

4.3. Ajuste del modelo maestro

Posterior a la evaluación y selección del modelo maestro, se debe proseguir con un análisis del modelo con el fin de descubrir su capacidad operativa a la hora de procesar el conjunto de datos de SIPLAB. Esto debido a que, el conjunto de datos utilizado en este proyecto tiene la connotación de capturar una vista superior de las personas debido a que las cámaras se encontraban posicionadas en el techo del edificio y a pesar que la mayoría de modelos de detección de objetos ya identifican personas, se desconoce su capacidad de reconocer esta vista.

4.3.1. Inicialización del ambiente

Previo al análisis del rendimiento del modelo YOLO respecto al conjunto de datos, se debe inicializar una serie de paquetes en el sistema computacional a utilizar. En este caso se cuenta con una computadora Acer Nitro 5 que posee un procesador Intel Core i5-9300H y una unidad de procesamiento gráfico NVIDIA GeForce GTX 1650.

En este dispositivo se instaló el sistema de gestión de paquetes Conda [69], que permite instalar los paquetes necesarios para correr un programa sin necesidad de instalarlos globalmente en el sistema. Además, se utilizó la versión 3.10.9 del lenguaje de programación Python, que es donde corre el código computacional brindado por [51] y se instalaron todos los paquetes presentes en la Tabla 4.10, que son los requeridos para garantizar su funcionamiento.

Tabla 4.10: Paquetes necesarios para correr el modelo YOLOv6

Paquete	Versión
torch	1.8.0
torchvision	0.9.0
numpy	1.24.0
opencv-python	4.1.2
PyYAML	5.3.1
scipy	1.4.1
tqdm	4.41.0
addict	2.4.0
tensorboard	2.7.0
pycocotools	2.0
onnx	1.10.0
onnx-simplifier	0.3.6
thop	-

4.3.2. Evaluación del modelo seleccionado

Con todos los requerimientos necesarios ya instalados, se procedió a evaluar el funcionamiento del modelo preentrenado con el conjunto de datos. Para ello, siguiendo la recomendación del cliente, se dividió el conjunto de datos en tres porciones: entrenamiento, validación y pruebas, y se asignó a cada porción uno de los tres días. Esto con el fin de hacer una validación cruzada y poder tener así una fiabilidad en los resultados.

Por lo tanto, se utilizó el Algoritmo 1 para convertir los videos a frames con el fin de construir el conjunto de datos reducido, pero, se tuvo que tomar en consideración que el modelo YOLOv6 solo admite imágenes con el mismo ancho y alto, por lo que, se redujo el tamaño de las imágenes del tamaño proporcionado (960x540) al estándar recomendado por el modelo: 640x640. Al realizar esta variación también debía de modificarse las anotaciones de los cuadros delimitadores para preservar su posición respecto a las imágenes originales, así que, se desarrolló el Algoritmo 5 para dicho fin.

Algoritmo 5: csv2data

Input: distancia entre frames, día, cámara, nueva altura, nuevo ancho
Output: txtfile

```
// Carga el archivo de las anotaciones
dirección = día + cámara
anotaciones = loadfile(dirección)

// Genera el archivo donde se van a guardar las anotaciones
txtfile = día + cámara + .txt
makefile(txtfile)

// Genera los radios de modificación de las anotaciones
xradio = nuevo ancho / 960
yradio = nueva altura / 540

// Recorre todos las filas del archivo de anotaciones
while cada fila do
    // Verifica que se cumpla el salto entre frames deseado
    if número fila actual % distancia entre frames == 0 then
        // Recorre todas las anotaciones de un frame
        for anotación in fila do
            // Modifica el tamaño del cuadro delimitador
            x1, y1, x2, y2 = anotación
            new x1 = x1 * xradio
            new x2 = x2 * xradio
            new y1 = y1 * yradio
            new y2 = y2 * yradio

            // Guarda las anotaciones modificadas
            new anotación = new x1, new y1, new x2, new y2
            saveValue(new anotación, txtfile)
        end
    end
end
```

Una vez realizados estos cambios, se verificó que los cuadros delimitadores del conjunto de datos aún envolviesen a las personas presentes en los frames.

Posterior a estas modificaciones, se asignaron los frames y las anotaciones de cada uno de los días del conjunto de datos a una de las porciones de entrenamiento y se puso a prueba el rendimiento del modelo de menor tamaño: YOLOv6 3.0 - S. De este proceso se obtuvieron los resultados presentes en la Tabla 4.11 al hacer la validación cruzada.

Tabla 4.11: Validación del modelo YOLOv6 3.0 - S con el conjunto de datos de SIPLAB

N°	**Train**	**Val**	**Test**	**mAP (%)**
1	Día 1	Día 2	Día 3	6,7253
2	Día 1	Día 3	Día 2	6,4025
3	Día 2	Día 1	Día 3	7,1859
4	Día 2	Día 3	Día 1	6,4025
5	Día 3	Día 1	Día 2	7,1859
6	Día 3	Día 2	Día 1	6,7253
Promedio				6,7712
Incertidumbre (±)				0,3215

En esta tabla se aprecia como en los casos que se usa el mismo conjunto de datos de validación, los resultados son iguales. Esto tiene sentido, debido a que, si se evalúa el modelo con un mismo conjunto de datos, las imágenes y anotaciones no van a variar y por ende, su valor de mAP será el mismo.

Por otro lado, al haber determinado el valor de mAP luego de la validación cruzada, se puede afirmar que el modelo presenta inconvenientes a la hora de realizar detecciones con el conjunto de datos proporcionado debido a su bajo valor, sin embargo, cabe resaltar, que estos resultados se llevaron a cabo con el modelo YOLOv6 3.0 - S, que es un modelo de tamaño reducido. Por lo que, se realizaron estas mismas pruebas en el modelo YOLOv6 3.0 - L con el fin de determinar si el bajo valor de mAP se debe a la simplificación del modelo o si al contrario, los modelos no están entrenados para identificar personas desde una vista superior. Se adjunta la Tabla 4.12 con los resultados de este proceso.

Tabla 4.12: Validación del modelo YOLOv6 3.0 - L con el conjunto de datos de SIPLAB

N°	**Train**	**Val**	**Test**	**mAP (%)**
1	Día 1	Día 2	Día 3	7,1356
2	Día 1	Día 3	Día 2	7,7347
3	Día 2	Día 1	Día 3	8,3014
4	Día 2	Día 3	Día 1	7,7347
5	Día 3	Día 1	Día 2	8,3014
6	Día 3	Día 2	Día 1	7,1356

Promedio	7,7239
Incertidumbre (±)	0,4760

A pesar de que tiene una leve mejoría en los datos de la Tabla 4.12 respecto a los de la Tabla 4.11, se determinó que el rendimiento era insuficiente para los fines del proyecto. La Fig. 4.4 demuestra que tanto el modelo S como el L presentan dificultades a la hora de realizar inferencias.

(a) Modelo YOLOv6 3.0 - S **(b)** Modelo YOLOv6 3.0 - L

Figura 4.4: Ejemplo de los resultados de la detección de personas. Fuente de la imagen base: [70]

4.3.3. Entrenamiento del modelo seleccionado

Al identificar que los modelos preentrenados presentan inconvenientes con la detección de personas desde una vista superior, se planteó seguir la recomendación inicial del cliente de entrenarlo para ajustar los pesos internos. Este proceso es conocido como ajustar el modelo. En este caso, se ajustó el modelo YOLOv6 3.0 - S y el YOLOv6 3.0 - L con el fin de determinar si es necesario para una buena detección tener un modelo de mayor tamaño o si es suficiente con un modelo reducido.

Por lo tanto, se procedió a hacer un entrenamiento de los modelos, sin embargo, se tuvo el inconveniente de que luego de 92 horas, el primer entrenamiento no había avanzado de manera significativa debido al alto coste computacional requerido para llevar a cabo la tarea. Por lo que, se decidió cambiar de dispositivo para realizar los entrenamientos. En este punto se hizo la consulta de si era posible utilizar algún sitio alojado, pero, la idea fue descartada, ya que el conjunto de datos proporcionado por SIPLAB presenta un acuerdo de confidencialidad que prohíbe su publicación en servidores o alojarlo en sitios de terceros. Finalmente, se tuvo acceso a una computadora de escritorio con un

procesador AMD Ryzen 5 5600G con Radeon y una unidad de procesamiento gráfico NVIDIA GeForce RTX 3000.

Luego de realizar el cambio del equipo y de inicializar nuevamente el ambiente de trabajo, se entrenaron los modelos tomando en consideración los valores predeterminados para entrenar el modelo, esto con el fin de reducir la variabilidad en los hiperparámetros y así reducir el tiempo requerido para el entrenamiento. Las únicas variaciones realizadas fueron: las épocas de entrenamiento, el batch size y los workers.

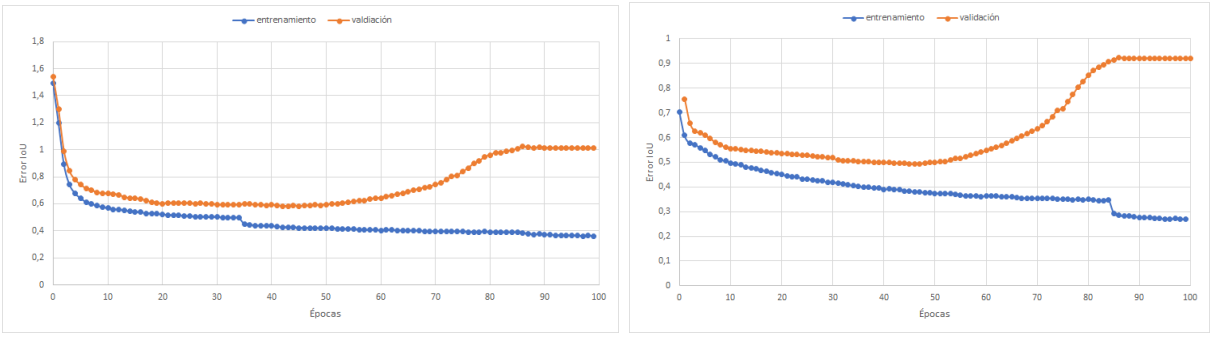
La Tabla 4.13 muestra los parámetros utilizados. El image size compete al tamaño de las imágenes del conjunto de datos de entrenamiento, que como se mencionó anteriormente para el entrenamiento debe ser una imagen cuadrada por lo que se reajustó el tamaño de las imágenes a 640x640. El batch size se refiere al tamaño de los lotes del conjunto de datos de entrenamiento con los que se entrena al modelo, por lo que se agrandó de tal manera que, el computador pudiese entrenar al modelo con el máximo valor posible sin que exista una saturación en la memoria del mismo. Las épocas corresponde a las iteraciones que se entrena el modelo, por lo que se extendió hasta un valor donde se asegure que el error de validación del modelo baje lo máximo posible. Los workers son los hilos que se preservan para correr el entrenamiento, por lo que se aumentó el valor para acelerar el proceso de entrenamiento. El dispositivo de entrenamiento corresponde a si se desea usar el GPU o el CPU durante el entrenamiento. El intervalo de evaluación es el intervalo para el cual se calcula el valor de mAP del modelo. El learning rate se refiere a la tasa de aprendizaje, que en este caso puede seleccionarse entre una tasa constante o variable. Y por último, en el optimizador se selecciona entre el descenso de gradiente estocástico (SGD), que es un tipo de optimizador que calcula el descenso del gradiente para cada lote de entrenamiento con el fin de actualizar los pesos del modelo o se elige el optimizador de momento adaptable (ADAM), que actualiza el momento del optimizador durante cada iteración.

Tabla 4.13: Valores de los hiperparámetros a la hora de hacer el fine-tuning

Hiperparámetro	**Valor**
Image size	640x640
Batch size	64
Épocas	100
Workers	12
Dispositivo	GPU
Intervalo de evaluación	3
Learning rate	Variable
Optimizador	SGD

Al realizar el entrenamiento de ambos modelos, se tuvo la curva de entrenamiento del

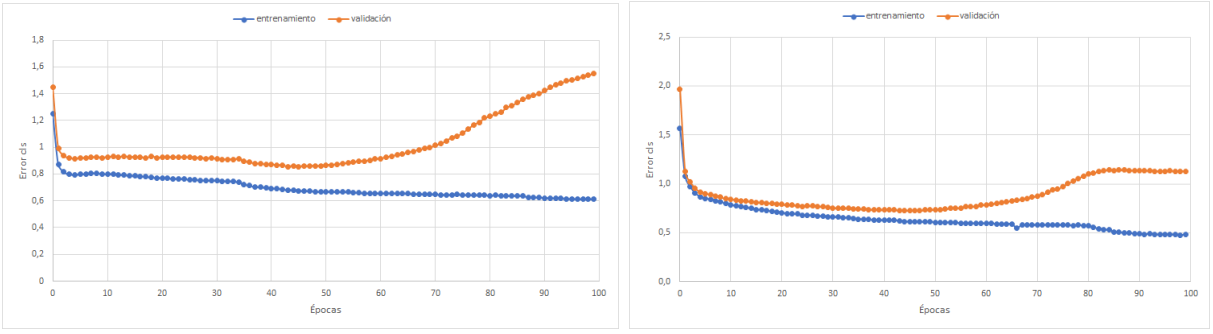
valor de mAP y la curva de los errores respecto a las épocas de entrenamiento. En el caso del error IoU se obtienen las gráficas de la Fig. 4.5, en donde se aprecia que conforme avanzan las iteraciones el error disminuye hasta que se llega a un punto de sobreentrenamiento (cercano a las 50 épocas para ambos casos) y a partir de ahí el error de validación incrementa. Cabe resaltar que este error corresponde a la disminución del error presente a la hora de calcular la intersección sobre la unión de las cuadros delimitadores encontrados por el modelo respecto a los del conjunto de datos de entrenamiento.



(a) Modelo YOLOv6 3.0 - S **(b)** Modelo YOLOv6 3.0 - L

Figura 4.5: Error *IoU* respecto a las épocas de entrenamiento

De manera similar, las gráficas presentes en la Fig. 4.6 compete al error presente a la hora de determinar a qué clase pertenece el cuadro delimitador respecto a la clase que realmente pertenece. En este caso, el conjunto de datos de entrenamiento de SIPLAB se enfoca en determinar personas, por lo que la única clase del conjunto de datos de entrenamiento se conoce como: *person*. Y al igual que con el error IoU, el cls disminuye hasta sobreentrenarse en un valor cercano a las 45-50 épocas.

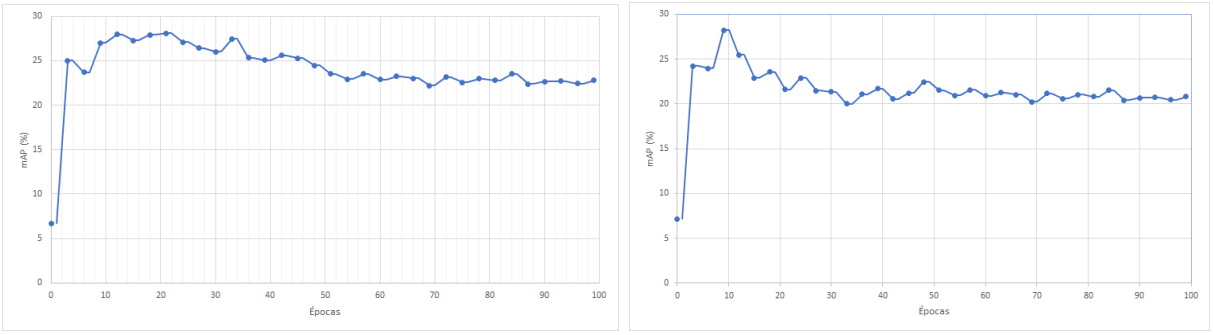


(a) Modelo YOLOv6 3.0 - S **(b)** Modelo YOLOv6 3.0 - L

Figura 4.6: Error *cls* respecto a las épocas de entrenamiento

Por último, se tiene la Fig. 4.7 con los valores de mAP resultantes de los entrenamientos. En ambos casos se aprecia como conforme transcurren las épocas el valor converge poco a poco a un determinado valor, no obstante, el valor máximo para el modelo S se da en la

época 21 y es de 28.0733 %, mientras que el modelo L presente su máximo en la iteración nuevo y es de 28.2062 %. En ambos casos los máximos se encuentran previo al punto de sobreentrenamiento, por lo que, se deduce que ambos valores son fiables.



(a) Modelo YOLOv6 3.0 - S **(b)** Modelo YOLOv6 3.0 - L

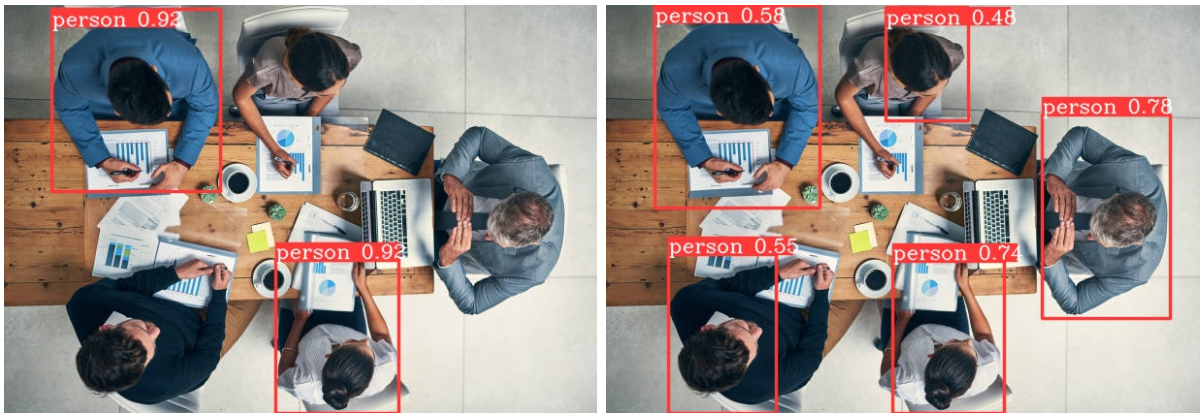
Figura 4.7: mAP respecto a las épocas de entrenamiento

Habiendo establecido que al ajustar los modelos se mejora considerablemente el rendimiento de los modelos, se tomó la decisión de continuar con el entrenamiento del modelo YOLOv6 3.0 - S, debido a que, a pesar de ser una simplificación del modelo L, presenta un mAP similar (modelo S tiene 18 530 545 parámetros y el modelo L 59 536 542). Por lo tanto, se realizó un validación cruzada de la red al entrenarla por 21 épocas y se obtuvo los valores de la Tabla 4.14 como resultado.

Tabla 4.14: Validación cruzada del modelo YOLOv6 3.0 - S

N	Train	Val	mAP (21 épocas)	mAP (modelo original)
1	Day1	Day2	27,0168 %	6,7253 %
2	Day3	Day2	26,6810 %	
3	Day1	Day3	27,0280 %	6,4025 %
4	Day2	Day3	29,4053 %	
5	Day2	Day1	32,6778 %	7,1859 %
6	Day3	Day1	24,9546 %	
Promedio			27,9606 %	6,7712 %
Error			2,4757 %	0,3215 %

Al analizar estos valores, luego de ajustar la red, se tiene como elección de red maestra el modelo YOLOv6 3.0 - S con un valor de mAP equivalente a (27.9606 ± 2.4757) %. A modo de ejemplo se adjunta la Fig. 4.8 en donde se muestra la mejora en la detección de personas.



(a) Modelo previo al ajuste **(b)** Modelo luego del ajuste

Figura 4.8: Comparación de la detección de personas luego de ajustar el modelo maestro. Fuente de la imagen base: [70]

4.4. Elección de la técnica de entrenamiento maestro-estudiante

Una vez definida y ajustada la red maestra, se tuvo que seleccionar la técnica de entrenamiento maestro-estudiante, que posibilite la reducción de la red maestra. Para ello, se explican los fundamentos teóricos de los conceptos seleccionados en el Capítulo 2 de este documento.

La selección de la técnica debe darse de tal manera que se tomen en cuenta los requisitos del cliente. En este caso, el fin de este trabajo consiste en reducir el tamaño de una red sin afectar considerablemente su rendimiento, ya que un posterior trabajo corresponderá en implementarlo dentro de un sistema embebido de bajo coste. Por lo cual, se adjunta la Tabla 4.15 en donde se presentan características de cada una de las técnicas y posibles ventajas y desventajas de la utilización de cada uno de ellas.

Tabla 4.15: Características de las técnicas de entrenamiento maestro-estudiante. Fuentes: [71][72][73][74][75]

Técnica	Descripción	Ventajas	Desventajas
Podado	Elimina las conexiones innecesarias o menos importantes del modelo	Disminuye el tamaño del modelo	Un podado agresivo tiende a bajar radicalmente la precisión de los modelos
		Puede disminuir el tiempo necesario para su procesamiento	-
Cuantización	Reduce la precisión de los parámetros del modelo	No depende del modelo, por lo que puede ser aplicado a cualquier modelo	Puede llevar a una disminución de la precisión del modelo
		Disminuye los requisitos de memoria del modelo	Puede llegar a requerir de hardware especializado para maximizar el rendimiento del modelo
		Puede ayudar a controlar el sobreentrenamiento de una red	-
Compresión	Mediante la codificación de los parámetros simplifica el tamaño del modelo	Disminuye el tamaño del modelo	La compresión extrema tiende a bajar radicalmente la precisión de los modelos
		Disminuye los requisitos de memoria del modelo	-
Destilación de conocimiento	Transfiere el conocimiento aprendido por un modelo más grande a otro con menor complejidad computacional	El modelo resultante aprende nada más la información relevante	Inevitablemente existe una disminución en la precisión del modelo
		Puede ayudar a controlar el sobreentrenamiento de una red	Requiere de gran cantidad de datos de entrenamiento

Una vez establecidas las características de cada uno de los conceptos, se realiza una comparación cuantitativa de la precisión respecto al nivel de reducción de una red al usar cada una de las técnicas, para ello, se tomaron referencias bibliográficas con el fin de tener una base comparativa entre las técnicas. Los resultados de esta fase se presentan en la Tabla 4.16.

Tabla 4.16: Factor de reducción de las técnicas. Fuentes: [72][74][75][76][77]

Técnica	Factor de reducción
Destilación de conocimiento	Se puede tener una radio de reducción de 7.5 veces con una pequeña pérdida en la precisión
Podado	Con un radio de reducción desde 2 hasta 6 el modelo tiene una pequeña pérdida en la precisión, pero conforme se pasa este umbral la precisión tiende a decaer.
Cuantización	Puede dar un radio de reducción entre 4 y 32 veces, pero para maximizar su eficiencia necesita de un hardware específico para que no decaiga la precisión.
Compresión	Puede reducir al modelo desde 4 veces hasta 35 veces, sin embargo, la precisión comienza a decaer conforme aumenta el radio de compresión

Con base en lo mostrado en las Tablas 4.15 y 4.16, se implementa la Tabla 4.17 con el fin de evaluar y seleccionar alguna técnica de entrenamiento maestro-estudiante. En este caso, se puede apreciar como al evaluar características como si es o no una técnica de entrenamiento maestro-estudiante, si posee la capacidad de preservar la precisión de la red (el valor de mAP), de reducir el tiempo de inferencia y de disminuir la memoria del programa, se tiene como concepto ganador a la destilación de conocimiento, lo cual concuerda con la sugerencia inicial dada por el cliente al entablar las primeras conversaciones del proyecto.

Tabla 4.17: Evaluación y selección de los conceptos de la Técnica de Entrenamiento Maestro-Estudiante

Técnica de entrenamiento maestro - estudiante			**Conceptos**							
Criterios de selección	**Importancia**	**Peso %**	**A**		**B**		**C**		**D**	
			Destilación de conocimiento		**Podado**		**Cuantización**		**Compresión**	
			Calificación	**Evaluación ponderada**	**Calificación**	**Evaluación ponderada**	**Calificación**	**Evaluación ponderada**	**Calificación**	**Evaluación ponderada**
Entrenamiento maestro-estudiante	5,00	0,25	3	0,75	3	0,75	3	0,75	3	0,75
mAP	5,00	0,25	5	1.25	2	0,50	3	0,75	2	0,50
Tiempo de inferencia	5,00	0,25	3	0,75	3	0,75	3	0,75	3	0,75
Memoria del programa	5,00	0,25	4	1,00	4	1,00	5	1,00	5	1,00
Puntos totales	20,00	100 %		3,75		3,00		3,50		3,25
Lugar			**1**		**4**		**2**		**3**	
¿Continuar?			Sí		No		No		No	

4.5. Implementación de la técnica de entrenamiento maestro-estudiante

Una vez establecido la destilación de conocimiento como técnica, se procedió a elegir cuál de sus variaciones desarrollar. En este caso la arquitectura base es la mostrada en la Fig. 4.9, de donde se aprecia que se transfiere el conocimiento por medio de la comparación entre la salida de la red maestra y la red estudiante.

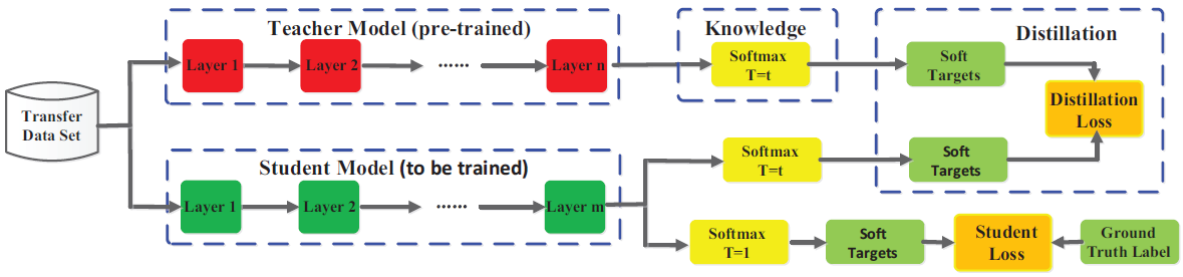


Figura 4.9: Arquitectura base de la destilación de conocimiento. Fuente: [78]

Sin embargo, en ocasiones esto puede limitar el potencial rendimiento de la red estudiante,

por lo que se optó por utilizar un enfoque donde se usan los mapas de características de las capas internas junto a la capa de salida para realizar la transferencia del conocimiento. La Fig. 4.10 ejemplifica esta variación. [78]

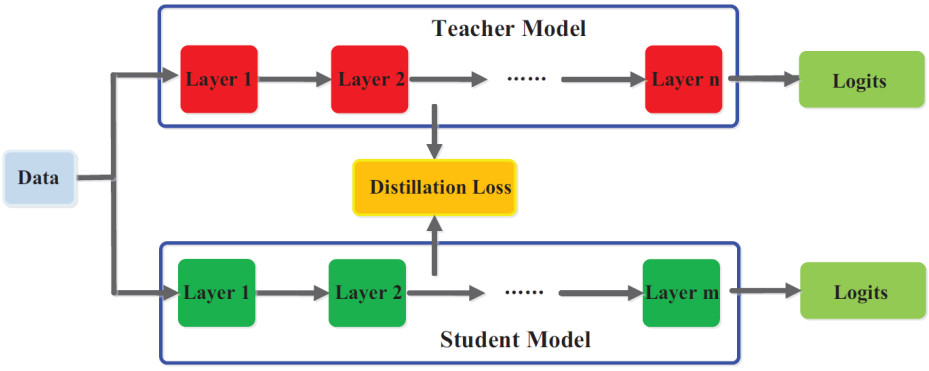


Figura 4.10: Arquitectura de destilación de conocimiento basada en los mapas de características. Fuente: [78]

[78] compara distintas arquitecturas de destilación de conocimiento y muestra que la destilación permite recudir el tamaño de modelos sin afectar su rendimiento, sin importar si son arquitecturas en línea (modelos maestros sin preentrenar), desconectadas (modelos maestros ya preentrenados), basadas en los mapas de características o basadas en relaciones entre las capas (observar la Fig. 6.1 del Anexo A para mayor detalle). Por lo tanto, se tomó en consideración esta arquitectura debido a su simplicidad en términos de la transferencia de conocimiento y su capacidad de generar altos resultados al igual que el resto de variaciones. Por lo que, a continuación se muestra el desarrollo del algoritmo implementado.

4.5.1. Carga de los datos

Con el fin de realizar el entrenamiento, se necesita de un código computacional que cargue los datos para poder ser utilizados. En este caso se utilizó el código por medio del cual la arquitectura YOLOv6 carga los datos para ajustar el modelo, que se puede ver representado por medio del Algoritmo 6.

Algoritmo 6: get data loader

Input: archivo.yaml, img size
Output: train dataset, val dataset

```
// Carga el archivo de la distribución del conjunto de datos
archivo datos = load yaml(archivo.yaml)

// Asigna las direcciones de los datos
train path = get(archivo datos, "train")
val path = get(archivo datos, "val")
number classes = get(archivo datos, "nc")
classes names = get(archivo datos, "names")

// Construye los conjuntos de datos
train dataset = build dataset(train path, img size, number classes, classes names)
val dataset = build dataset(val path, img size, number classes, classes names)
```

4.5.2. Carga de los modelos

De manera similar, los modelos se cargan por medio del código proporcionados por YOLOv6. En este caso, se utiliza el Algoritmo 7 para cargar e iniciar tanto el modelo maestro como el modelo estudiante.

Algoritmo 7: get model

Input: modelo a usar, preentrenado
Output: Modelo

```
// Carga el archivo del modelo a utilizar
dirección = modelo a usar
archivo modelo = loadfile(dirección)

// Genera la cabeza, el cuello y la columna vertebral del modelo
head = get(archivo modelo, "head")
neck = get(archivo modelo, "neck")
backbone = get(archivo modelo, "backbone")

// Construye el modelo
Modelo = build model(head, neck, backbone)

// Carga los pesos en los casos que el modelo es preentrenado
if preentrenado == True then
    pesos = torch.load(archivo modelo)
    Modelo = load state dict(pesos)
end

// Retorna el modelo
return Modelo
```

4.5.3. Optimizador

Posteriormente es necesario un modo de optimizar los modelos durante el entrenamiento. Para ello, el Algoritmo 8 muestra el pseudocódigo de los optimizadores implementados, que corresponden al optimizador ADAM y al SGD.

Algoritmo 8: set optimizer

Input: tipo optimizador, Modelo
Output: optimizador

```
// Genera una lista con los parámetros a optimizar
bias list = [ ]
weight list = [ ]

// Recorre los parámetros del modelo
for parametro in Modelo.parametros do
    if parametro == "sesgo" then
        bias list.append(parametro)
    end
    if parametro == "peso" then
        weight list.append(parametro)
    end
end

// Genera el optimizador
if tipo optimizador == "SGD" then
    optimizador = torch.optim.SGD(bias list, weight list)
end
if tipo optimizador == "ADAM" then
    optimizador = torch.optim.Adam(bias list, weight list)
end

// Retorna el optimizador
return optimizador
```

4.5.4. Tasa de aprendizaje

Una vez definido el optimizador por utilizar, se generó la tasa de aprendizaje sobre la que se basara el optimizador. En este caso se consideraron dos opciones: la primera que fuera una tasa de aprendizaje variable de manera lineal con respecto a las épocas y las segunda que fuese constante durante todo el entrenamiento. El Algoritmo 9 ejemplifica el código diseñado para esta sección.

Algoritmo 9: set lr scheduler

Input: tipo de tasa, época actual, total épocas
Output: lr

```
// Elije la tasa de aprendizaje
if tipo de tasa == "Variable" then
    lr = 1 - época actual / total épocas
end
if tipo de tasa == "Constante" then
    lr = 1
end

// Retorna el valor de la tasa de aprendizaje
return lr
```

4.5.5. Error

Una vez establecidas la manera de cargar los conjuntos de datos, una forma de iniciar los modelos y diseñado un optimizador junto a su tasa de aprendizaje, se necesita implementar el error a ser optimizado. En este caso, se poseen dos tipos de errores: el de la salida de los modelos y el de los mapas de activación. A continuación se detallan ambos.

Salidas del modelo

En este caso, se tomó en consideración la explicación dada por [61], en donde se comenta que para desarrollar el error de destilación se deben someter las partes del modelo estudiante y del modelo maestro a la función *softmax* con el fin de suavizar la distribución de la salida, además, se utilizó una variable conocida como temperatura que permite aumentar o disminuir la aplicación de la función softmax. El Algoritmo 10 detalla lo recién comentado.

Algoritmo 10: kd loss output

Input: salida modelo estudiante, salida modelo maestro
Output: error

```
// Aplica la función softmax para suavizar la distribución
```
salida estudiante suavizada = softmax(salida modelo estudiante, temperatura)
salida maestro suavizada = softmax(salida modelo maestro, temperatura)

```
// Calcula el error entre las distribuciones
```
error = cross entropy(salida estudiante suavizada, salida maestro suavizada)

```
// Retorna el error
```
return error

Mapas de activación del modelo

De manera similar, se generó un código que permitiese calcular el error presente entre los mapas de activación del modelo maestro y el modelo estudiante. El Algoritmo 11 muestra su respectivo pseudocódigo.

Algoritmo 11: kd loss maps

Input: mapas del ME, mapas del MM
Output: error

```
// Aplica la función softmax para suavizar la distribución
```
mapas estudiante suavizada = softmax(mapas del ME, temperatura)
mapas maestro suavizada = softmax(mapas del MM, temperatura)

```
// Calcula el error entre las distribuciones
```
error = cross entropy(mapas estudiante suavizada, mapas maestro suavizada)

```
// Retorna el error
```
return error

4.5.6. mAP

Una vez establecidos los errores del entrenamiento, se utilizó el código computacional del modelo YOLOv6 para obtener el valor de mAP durante cada época y de esta manera analizar el rendimiento del modelo estudiante respecto al maestro. El Algoritmo 12 detalla dicha función.

Algoritmo 12: eval mAP

Input: modelo, dataset
Output: mAP

```
// Lista de AP para cada clase
mAP list = [ ]

for i in classes(dataset) do
    // Obtiene del conjunto de datos de cada clase
    dataset clase = dataset[i]

    // Evalúa el conjunto de datos en el modelo
    inputs, t outputs = split(dataset clase)
    bbox = eval(modelo, inputs)

    // Calcula la IoU
    iou model = IoU(bbox, t outputs)

    // Genera la curva de precisión-sensibilidad
    p list, s list = [ ], [ ]
    for th in [0,1], step = 0.5 do
        confusion matrix = (iou model, th)
        p = precision(confusion matrix)
        s = recall(confusion matrix)
        p list.append(p)
        s list.append(s)
    end

    // Obtiene el valor de AP de la clase (área bajo la curva)
    AP = auc(p list, s list)
    mAP list.append(AP)

end

// Retorna el valor de mAP
return mean(mAP list)
```

4.5.7. Ciclo de entrenamiento

Finalmente, una vez establecidas todas las funciones necesarias para implementar la técnica, se desarrolló un algoritmo que fuera capaz de realizar un entrenamiento maestro-estudiante. En este caso, el código sigue el diagrama de flujo mostrado en la Fig. 4.11

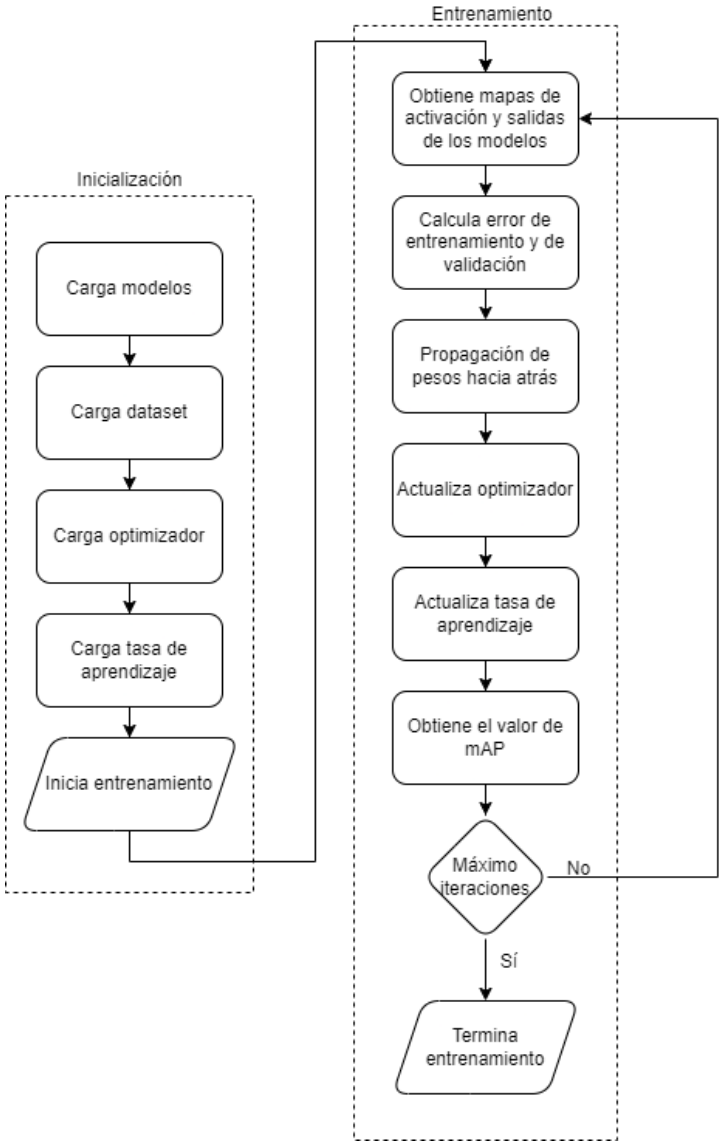


Figura 4.11: Diagrama de flujo de la técnica de destilación de conocimiento desarrollada.

En este caso, en primera instancia se cargan los modelos (maestro preentrenado y estudiante vacío) y el conjunto de datos por utilizar por medio de los Algoritmos 6 y 7, luego se carga el optimizador seleccionado junto a su tasa de aprendizaje al usar los códigos de los Algoritmos 8 y 9 para proceder con el entrenamiento del modelo estudiante. Durante el entrenamiento se realiza para cada época las inferencias de ambos modelos por medio del conjunto de datos, en esta fase se obtienen los cuadros delimitadores en sus salidas y los mapas de activación internos. Una vez obtenidos estos resultados se calcula el error de destilación por medio de los Algoritmos 10 y 11. Al obtener este valor, se obtiene la curva de validación y de entrenamiento. Luego, por medio del error de entrenamiento, se realiza una propagación hacia atrás de los pesos para ajustar los parámetros internos del modelo estudiante y se actualiza el optimizador junto a su tasa de aprendizaje para la siguiente iteración. Finalmente se obtiene el valor de mAP obtenido para cada época.

4.6. Definición de los modelos estudiantes

Con el fin de tener variantes para los modelos estudiantes, se hizo un estudio del la topología de la red neuronal, de ella se notaron dos parámetros que permitieron generar modelos estudiantes con base en el modelo maestro: profundidad de la red y ancho de la red. El primero de ellos permite variar la cantidad de repeticiones de los sub-bloques del modelo, ya sea en la columna, el cuello o la cabeza de la red. El segundo, posibilita variar el contenido de las convoluciones de cada sub-bloque, es decir, permite variar el tamaño del kernel presente. Por lo tanto, considerando la red neuronal base, se decidió generar tres variantes.

- Variante A: Se disminuyó a la mitad el ancho de la red. El resultado de este proceso dio como resultado una red estudiante con 4 644 477 parámetros.
- Variante B: Se redujo a la mitad la profundidad del modelo maestro, esto permitió tener un modelo reducido con 3 861 757 parámetros.
- Variante C: Se disminuyeron a la mitad el ancho y la profundidad de la red maestra, dando así como resultado un modelo estudiante con 1 166 909 parámetros.

En el Apéndice C se encuentran a detalle el contenido de las capas internas de cada una de las variantes.

Capítulo 5

Resultados y análisis

Esta sección se muestra el entrenamiento del modelo estudiante llevado a cabo por medio del programa desarrollado en el Capítulo 4, la validación y el análisis del modelo resultante respecto a los requisitos del cliente y un análisis económico sobre el proyecto llevado a cabo.

5.1. Entrenamiento del modelo estudiante

Tomando en consideración el flujo mostrado en la Fig. 4.11 se desarrollaron pruebas sobre el programa con el fin de encontrar el modelo estudiante óptimo. Para ello, se debe tomar en consideración los hiperparámetros del programa, los cuales se aprecian en la Fig. 5.1.

Tabla 5.1: Hiperparámetros de la destilación

Hiperparámetros
Temperatura
Épocas
Optimizador
Tasa de aprendizaje
Modelo

En este caso se tomó en consideración un entrenamiento en dos fases: pruebas preliminares y pruebas exhaustivas. La primera pretende ajustar y conocer los valores ideales de los hiperparámetros con el fin de usar los valores óptimos para la segunda fase, que consiste en llevar a cabo pruebas con todos las posibles combinaciones entre los parámetros ideales. Además, se le recuerda al lector que el conjunto de datos está divido en tres días, por lo que, existen seis posibles combinaciones para entrenar los modelos tal y como se muestra

en la Tabla 5.2. En el caso de las pruebas preliminares y las pruebas exhaustivas se utilizó la primera variante y posteriormente, en la sección de validación se utilizan las demás.

Tabla 5.2: Posibles variantes del conjunto de datos para hacer el entrenamiento

N	Train	Val	Test
1	Día 1	Día 2	Día 3
2	Día 3	Día 2	Día 1
3	Día1	Día3	Día 2
4	Día2	Día3	Día 1
5	Día2	Día1	Día 3
6	Día3	Día1	Día 2

5.1.1. Pruebas preliminares

En primera instancia, se desconocía cuál de las variaciones de los modelos estudiantes presentes en el Apéndice C era la ideal para llevar a cabo el entrenamiento, por lo que se realizaron las pruebas presentes en la Tabla 5.3 con el fin de detallar el rendimiento de las variantes.

Tabla 5.3: Pruebas preliminares sobre las variantes del modelo estudiante

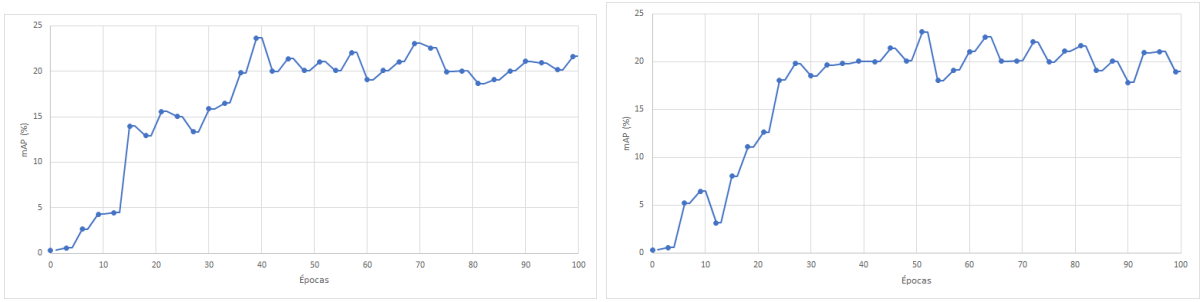
Épocas	Tasa de aprendizaje	Optimizador	Modelo	Temperatura	mAP(%)	N° Parametros
100	Variable	SGD	Variante A	10	23,0944	4 644 477
100	Variable	SGD	Variante A	20	23,6465	4 644 477
100	Variable	SGD	Variante B	10	16,6547	3 861 757
100	Variable	SGD	Variante B	20	18,6052	3 861 757
100	Variable	SGD	Variante C	10	7,1613	1 166 909
100	Variable	SGD	Variante C	20	7,6634	1 166 909

En estas pruebas se decidió utilizar una temperatura equivalente a 10 y 20 debido a que [76] presenta buenos resultados con estos valores de hiperparámetros, por lo que se consideró como un buen punto de partida. Luego de 100 iteraciones se aprecia en la Tabla 5.4 como las Variantes B y C presentan una reducción una importante caída en sus rendimientos, por lo que se tomó la decisión de utilizar únicamente la Variante A para el resto de pruebas.

Tabla 5.4: Resultado de la reducción de la precisión para cada variante de modelo estudiante luego de las pruebas preliminares

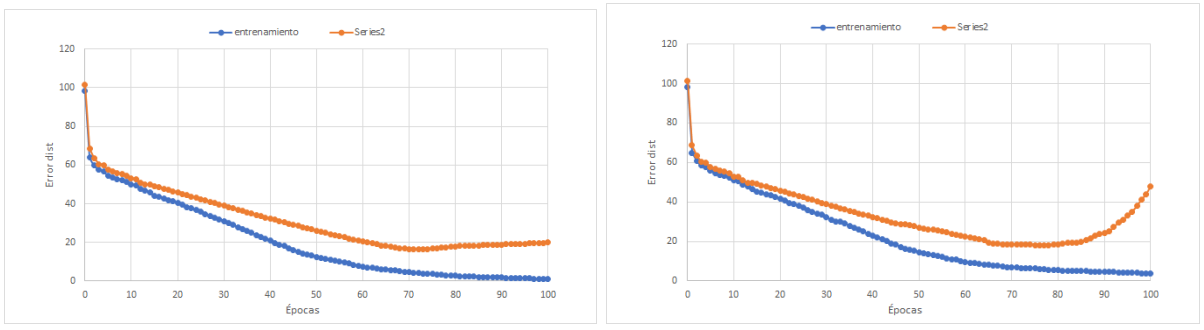
Variante	Resultados	mAP maestro (%)	mAP estudiante (%)	Reudcción (%)
A	1	27,0168	23,0944	-14,5184
	2	27,0168	23,6465	-12,4748
B	1	27,0168	16,6547	-38,3543
	2	27,0168	18,6052	-31,1347
C	1	27,0168	7,1613	-73,4932
	2	27,0168	7,6634	-71,6347

Habiendo establecido el modelo estudiante por utilizar, se procedió a hacer un estudio de la afectación de las épocas en el entrenamiento, ya que, cada entrenamiento consumió un alto coste computacional en función del tiempo (15 horas aproximadamente), por lo que, se deseaba conocer si era posible reducir el tiempo de entrenamiento sin disminuir el rendimiento de los modelos resultantes. Para ello, se tomó en consideración las curvas presentes en las Figs. 5.1 y Fig. 5.2, en donde se aprecia que los valores máximos de mAP obtenidos no se encuentran dentro del rango de sobreentrenamiento, que ronda cerca de las 75-80 épocas.



(a) Modelo previo al ajuste **(b)** Modelo luego del ajuste

Figura 5.1: Comparación de la detección de personas luego de ajustar el modelo maestro.



(a) Modelo previo al ajuste **(b)** Modelo luego del ajuste

Figura 5.2: Comparación de la detección de personas luego de ajustar el modelo maestro.

Por lo tanto, se decidió utilizar 25, 50 y 75 épocas para entrenar a la variante A y los resultados de esta fase se observan en la Tabla 5.3.

Tabla 5.5: Pruebas preliminares sobre las épcoas de entrenamiento

Temperatura	Épocas	Optimizador	Tasa de aprendizaje	mAP(%)
20	25	SGD	Variable	5,7603
20	50	SGD	Variable	22,4556
20	75	SGD	Variable	24,9318

En este caso se puede apreciar como es necesario una cantidad importante de épocas para que la destilación de conocimiento logre transferir correctamente las características al modelo estudiante, sin embargo, se aprecia que tampoco es necesario extender este valor más allá de 75 épocas.

Posteriormente, se utilizó una cantidad fija de épocas y se procedió a variar la temperatura con el fin de encontrar la variación óptima para este hiperparámetro. En la Tabla 5.6 se adjuntan los resultados de esta fase.

Tabla 5.6: Pruebas preliminares sobre la temperatura al destilar conocimiento

Temperatura	Épocas	Optimizador	Tasa de aprendizaje	mAP(%)
0	50	SGD	Variable	0,0000
10	50	SGD	Variable	24,3801
20	50	SGD	Variable	20,4556
40	50	SGD	Variable	20,1601
80	50	SGD	Variable	16,4238

Con base en los resultados recién mostrados, se denota como a menores temperaturas parece ser que el modelo estudiante presenta una mayor capacidad de aprender correctamente durante el entrenamiento.

Por último se analizaron los resultados al variar tanto la tasa de aprendizaje como el optimizador, en la Tabla 5.7 se muestra a detalle los resultados obtenidos luego de entrenar a la Variante A durante 75 épocas al hacer estas variaciones y se aprecia como al utilizar el optimizador Adam el rendimiento del modelo disminuye sin importar el tipo de tasa de aprendizaje, por lo que se descarta la idea de utilizar el optimizador Adam durante las pruebas exhaustivas.

Tabla 5.7: Pruebas preliminares sobre la tasa de aprendizaje y el optimizador

Temperatura	Épocas	Optimizador	Tasa de aprendizaje	mAP(%)
20	75	Adam	Constante	15,0651
20	75	Adam	Variable	14,0696
20	75	SGD	Constante	23,9436
20	75	SGD	Variable	22,8783

5.1.2. Pruebas exhaustivas

Posterior al ajuste de los hiperparámetros del modelo, se llevaron a cabo entrenamientos tomando en consideración que: la variante con los mejores modelos resultantes es la A, el modelo tuvo buenos resultados al entrenarse durante 50 y 75 épocas, se tuvo mejores resultados al utilizar temperaturas bajas (pero distintas de cero), se presentó inconvenientes al utilizar el optimizador Adam, con el optimizador SGD se tuvo resultados similares al variar la tasa de aprendizaje de fija a variable. Por lo tanto, con base en lo recién mencionado, se realizaron las pruebas exhaustivas mostradas en la Tabla 5.8.

Tabla 5.8: Pruebas exhaustivas llevadas a cabo

N°	Temperatura	Tasa de aprendizaje	Épocas	mAP (%)
1	5	Constante	50	9,8198
2	5	Constante	75	24,0643
3	5	Variable	50	21,2552
4	5	Variable	75	23,1654
5	10	Constante	50	20,3011
6	10	Constante	75	24,1118
7	10	Variable	50	23,2249
8	10	Variable	75	24,3801
9	15	Constante	50	19,6547
10	15	Constante	75	22,5872
11	15	Variable	50	16,8450
12	15	Variable	75	22,5855
13	20	Constante	50	16,5381
14	20	Constante	75	23,9436
15	20	Variable	50	20,4556
16	20	Variable	75	25,0761
17	25	Constante	50	15,8094
18	25	Constante	75	21,9012
19	25	Variable	50	19,2328
20	25	Variable	75	21,6997

De estos resultados se aprecia como la prueba número 16 es la que presenta el mejor rendimiento de mAP, además, al analizar las gráficas de la Fig. 5.3 se aprecia como el modelo no presente un sobreentrenamiento antes de tener su valor más alto de mAP (época 57 de la Fig. 5.4), por lo que se cerciora que los resultados obtenidos son fiables.

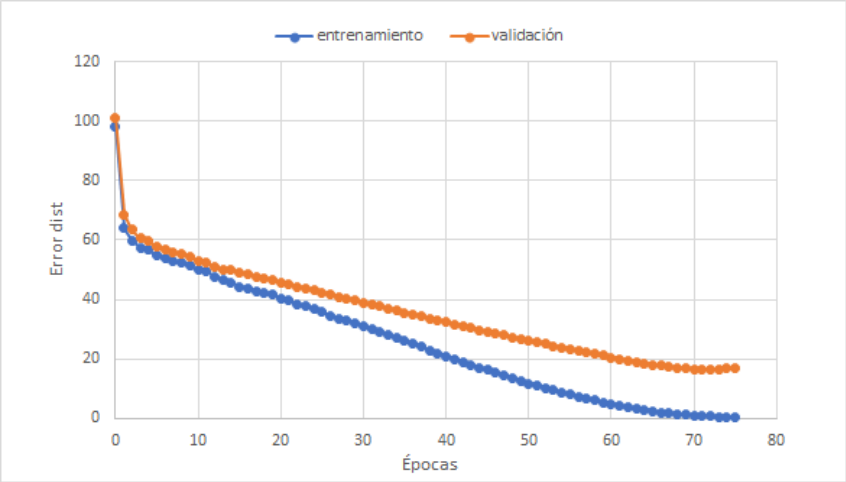


Figura 5.3: Error de la destilación respecto a las épocas de entrenamiento.

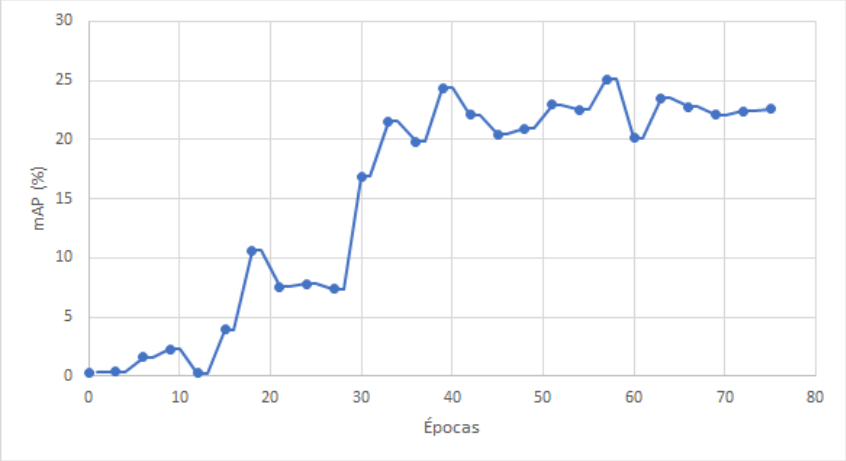


Figura 5.4: mAP del modelo estudiante respecto a las épocas de entrenamiento.

Por lo tanto, con base en los resultados obtenidos en esta sección, se procede a realizar la validación cruzada en la siguiente sección usando los siguientes hiperparámetros: Temperatura: 20, tasa de aprendizaje: variable, épocas: 75, optimizador: SGD y modelo: Variante A.

5.2. Validación del modelo estudiante

En este apartado, se pretende analizar si la destilación efectuada cumple con los requisitos del cliente de mantener la precisión de la red mientras que se reduce el tiempo de inferencia y la memoria consumida. Por lo tanto, se divide la validación de los resultados en dos apartados: el primero en la validación de la precisión y el segundo en la validación del consumo de memoria y el tiempo de inferencia.

5.2.1. mAP

En primera instancia se plantea hacer una comparación directa entre el modelo estudiante resultante y su modelo maestro, por lo que, en la Fig. 5.5 se aprecian los resultados luego de hacer una inferencia en ambos modelos.

(a) Modelo maestro **(b)** Modelo estudiante

Figura 5.5: Comparación de la detección de personas luego de entrenar el modelo estudiante. Fuente de la imagen base: [70]

A pesar de que en el ejemplo se aprecia de manera gráfica que la precisión de los resultados se mantienen y no caen abruptamente, se debe de realizar una comparación de los valores de mAP del modelo estudiante respecto al maestro. Para ello, tal y como se menciona en el Capítulo 3 de este documento, se pretende darle fiabilidad a los resultados por medio de una validación cruzada, por lo tanto, la Tabla 5.9 muestra el resultado luego de la destilación de conocimiento para cada variación.

Tabla 5.9: Resultados de la validación cruzada del entrenamiento del modelo estudiante

N	Train	Val	Test	mAP(%)
1	Día 1	Día 2	Día 3	25,0761

2	Día 3	Día 2	Día 1	24,8980
3	Día1	Día3	Día2	25,4717
4	Día2	Día3	Día 1	27,8155
5	Día2	Día1	Día 3	30,3916
6	Día3	Día1	Día2	23,6201
		Promedio		26,2122
		Error		2,2481

Los resultados de este proceso muestran que al hacer el entrenamiento maestro-estudiante por medio de la destilación de conocimiento se tuvo un valor de mAP equivalente a (26,2122±2,2481) %. Y al comparar estos resultados respecto a los del modelo maestro (Tabla 4.14), se tienen los resultados de la Tabla 5.10. Esta tabla muestra como en promedio el porcentaje de reducción de la precisión del modelo es menor al 6.2291 %, además, este valor tiene un error de 0.7506 %.

Tabla 5.10: Reducción de la precisión del modelo estudiante

N	**Train**	**Val**	**Test**	**Estudiante(%)**	**Maestro (%)**	**Reducción (%)**
1	Día 1	Día 2	Día 3	25,0761	27,0168	-7,1834
2	Día 3	Día 2	Día 1	24,8980	26,6810	-6,6825
3	Día1	Día3	Día2	25,4717	27,0280	-5,7582
4	Día2	Día3	Día 1	27,8155	29,4053	-5,4065
5	Día2	Día1	Día 3	30,3916	32,6778	-6,9961
6	Día3	Día1	Día2	23,6201	24,9546	-5,3476
		Promedio		26,2122	27,9606	-6,2291
		Error		2,2481	2,4756	0,7506

5.2.2. Tiempo de inferencia y consumo de memoria

Por otro lado, no solo se deseaba mantener la precisión del modelo, sino también disminuir su consumo de memoria y el tiempo de inferencia. Con el fin de cuantificar estos valores, se desarrolló el Algoritmo 13, que pretende poder medir los recursos computacionales usados por la computadora al hacer inferencias y el tiempo que toma en hacer cada una.

Algoritmo 13: memory2csv

Input: cant, test, dataset
Output: csvfile

```
// Inicia las listas
GPU memory list = [ ], RAM memory list = [ ], inference time list = [ ]

// Toma datos de la memoria al no realizar inferencias
start = now()
sleep(120)
end = now()
GPU memory list.append(GPU.use(end - start), -1)
RAM memory list.append(RAM.use(end - start), -1)

// Carga el conjunto de datos de prueba
test dataset = load(test)
td = test dataset[0:cant]

// Realiza la inferencia de cada imagen
for num, imagen in td do
    start = now()
    infer(modelo, imagen)
    end = now()

    // Añade los datos a las listas
    GPU memory list.append(GPU.use(end - start), num)
    RAM memory list.append(RAM.use(end - start), num)
    inference time list.append((end - start), num)
end

// Toma datos de la memoria al no realizar inferencias
start = now()
sleep(120)
end = now()
GPU memory list.append(GPU.use(end - start), -2)
RAM memory list.append(RAM.use(end - start), -2)


// Guarda los valores en un archivo .csv
csvfile.save(GPU memory list, RAM memory list, inference time list)
```

Con este código fue posible cuantificar el uso de la memoria CPU, memoria RAM y el tiempo de inferencia de los modelos. En el Apéndice D se muestran los resultados

al utilizar cada uno de los modelos resultantes de las pruebas exhaustivas desarrolladas (Tabla 5.8) con este algoritmo para hacer inferencias sobre 2000 imágenes del conjunto de datos. Estas pruebas se realizaron, en la mañana de un día, en la tarde del siguiente y en la noche de otro día. Esto con el fin de incluir alguna afectación en el rendimiento del hardware debido a cambios climáticos o factores similares. Además fueron realizadas en una computadora Acer Nitro 5 con un procesador Intel Core i5-9300H y una unidad de procesamiento gráfico NVIDIA GeForce GTX 1650.

Los resultados del modelo estudiante seleccionado (prueba exhaustiva 16) se aprecian en la Fig. D.4 del Apéndice D, pero, cabe recalcar que estos competen a los de la primera variación de la validación cruzada, así que, con el fin de darle una mayor confiabilidad probabilística a estos valores, se llevaron a cabo pruebas con este modelo para el resto de variaciones, teniendo como resultado la Tabla E.1.

Tabla 5.11: Consumo de memoria y tiempo de inferencia del modelo estudiante elegido

N		Cantidad parámetros	GPU run			CPU run		
			t inferencia (s)	Memoria GPU (GB)	RAM (GB)	t inferencia (s)	Memoria GPU (GB)	RAM (GB)
1	1,1	4 644 477	7,369144	0,567548	3,486851	3,375318	0,277073	3,486246
	1,2	4 644 477	7,369393	0,636345	3,486524	3,204071	0,292703	3,486309
	1,3	4 644 477	6,779296	0,635330	3,486508	3,142476	0,331176	3,486312
2	2,1	4 644 477	7,132132	0,599849	3,489790	3,219652	0,306916	3,479798
	2,2	4 644 477	7,261562	0,594965	3,486232	3,256157	0,296541	3,486546
	2,3	4 644 477	7,095857	0,580893	3,486543	3,219651	0,298494	3,489622
3	3,1	4 644 477	7,016516	0,601951	3,490103	3,256516	0,298102	3,489624
	3,2	4 644 477	7,126754	0,591713	3,486851	3,246876	0,292145	3,487663
	3,3	4 644 477	6,645651	0,606155	3,485163	3,276154	0,297898	3,489741
4	4,1	4 644 477	7,034665	0,586516	3,483516	3,312655	0,302999	3,487583
	4,2	4 644 477	6,849846	0,609849	3,487945	3,301695	0,298795	3,486645
	4,3	4 644 477	7,006470	0,599652	3,486979	3,286156	0,279489	3,486457
5	5,1	4 644 477	7,198550	0,609490	3,483617	3,361565	0,289849	3,487352
	5,2	4 644 477	6,948984	0,576156	3,486555	3,336322	0,296955	3,483213
	5,3	4 644 477	7,298456	0,586516	3,489585	3,289489	0,301942	3,486542
6	6,1	4 644 477	6,984984	0,599794	3,486515	3,196820	0,299655	3,486542
	6,2	4 644 477	7,366517	0,600616	3,484954	3,331437	0,289495	3,489498
	6,3	4 644 477	7,196850	0,589652	3,465168	3,264568	0,290654	3,469457

De esta tabla, se calculan los valores promedio para cada variación y se obtiene datos del consumo de memoria y el tiempo de inferencia con un respectivo error para cada variación de la validación cruzada. Dichos resultados se aprecian en la Tabla 5.12.

Tabla 5.12: Validación cruzada del consumo de memoria y tiempo de inferencia del modelo estudiante elegido

N	GPU run			CPU run		
	t inferencia (s)	Memoria GPU (GB)	RAM (GB)	t inferencia (s)	Memoria GPU (GB)	RAM (GB)
1	7,172611	0,613074	3,486628	3,240622	0,300317	3,486289
2	7,163183	0,591903	3,487521	3,231820	0,300650	3,485322
3	6,929640	0,599940	3,487372	3,259849	0,296048	3,489009
4	6,963660	0,598672	3,486147	3,300169	0,293761	3,486895
5	7,148663	0,590721	3,486585	3,329125	0,296249	3,485702
6	7,182783	0,596687	3,478879	3,264275	0,293268	3,481832
Promedio	7,093424	0,598499	3,485522	3,270977	0,296716	3,485842
Error	0,049611	0,012314	0,034379	0,013872	0,025121	0,027688

De manera similar, se obtienen las características del modelo maestro (visualizar el Apéndice E) y en la Tablas 5.13 y 5.14 se comparan los resultados respecto a los del modelo maestro.

Tabla 5.13: Comparación del tiempo de inferencia y el consumo de memoria entre el modelo maestro y el modelo estudiante al correr los modelos con el GPU

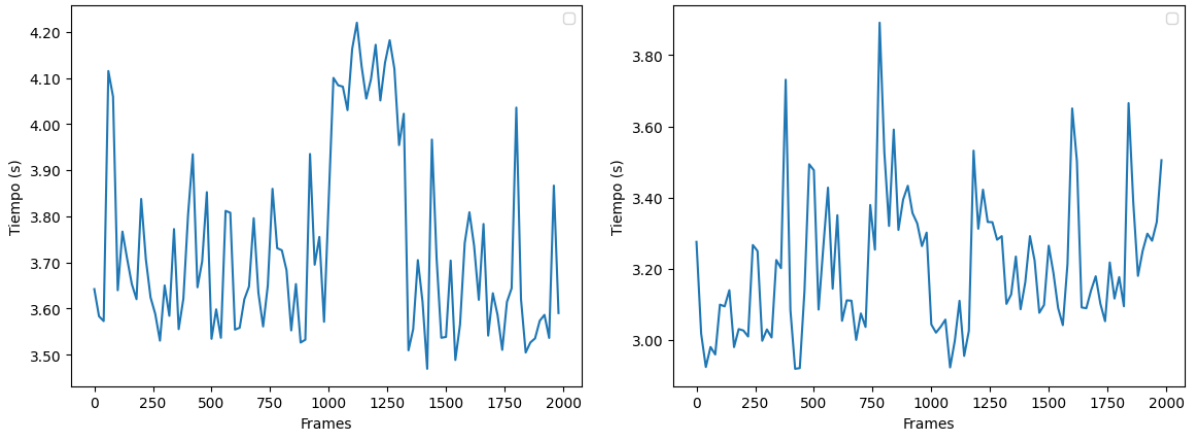
Características		Variantes de la validación cruzada						Promedio	
		1	2	3	4	5	6	Valor	Error
Estudiante	t inferencia (s)	7,172611	7,163183	6,929640	6,963660	7,148663	7,182783	7,093424	0,049611
	Memoria GPU (GB)	0,613074	0,591903	0,599940	0,598672	0,590721	0,596687	0,598499	0,012314
	RAM (GB)	3,486628	3,487521	3,487372	3,486147	3,486585	3,478879	3,485522	0,034379
Maestro	t inferencia (s)	7,582571	7,597515	7,600263	7,597481	7,592809	7,601440	7,595346	0,006328
	Memoria GPU (GB)	0,698482	0,696276	0,698973	0,704410	0,702207	0,690441	0,698465	0,004449
	RAM (GB)	3,490052	3,491738	3,488903	3,489003	3,487877	3,479564	3,487856	0,003896
Reducción (%)	t inferencia	-5,406605	-5,716754	-8,823676	-8,342517	-5,849556	-5,507598	-6,607784	1,410766
	Memoria GPU	-12,227570	-14,990295	-14,168386	-15,010862	-15,876590	-13,578849	-14,308758	1,176106
	RAM	-0,098117	-0,120749	-0,043876	-0,081874	-0,037036	-0,019683	-0,066889	0,035940

Tabla 5.14: Comparación del tiempo de inferencia y el consumo de memoria entre el modelo maestro y el modelo estudiante al correr los modelos con el CPU

Características		Variantes de la validación cruzada						Promedio	
		1	2	3	4	5	6	Valor	Error
Estudiante	t inferencia (s)	3,240622	3,231820	3,259849	3,300169	3,329125	3,264275	3,270977	0,013872
	Memoria GPU (GB)	0,300317	0,300650	0,296048	0,293761	0,296249	0,293268	0,296716	0,025121
	RAM (GB)	3,486289	3,485322	3,489009	3,486895	3,485702	3,481832	3,485842	0,027688
Maestro	t inferencia (s)	3,879896	3,868571	3,894875	3,840635	3,906813	3,790832	3,863604	0,038640
	Memoria GPU (GB)	0,337806	0,354284	0,352638	0,358552	0,354183	0,364273	0,353623	0,008058
	RAM (GB)	3,488004	3,490804	3,493638	3,487955	3,488210	3,489364	3,489663	0,002040
Reducción (%)	t inferencia	-16,476577	-16,459594	-16,304149	-14,072317	-14,786669	-13,890276	-15,331597	1,117240
	Memoria GPU	-11,097603	-15,138627	-16,047571	-18,070144	-16,357128	-19,492210	-16,033880	2,626131
	RAM	-0,049178	-0,157053	-0,132481	-0,030398	-0,071899	-0,215850	-0,109476	0,065076

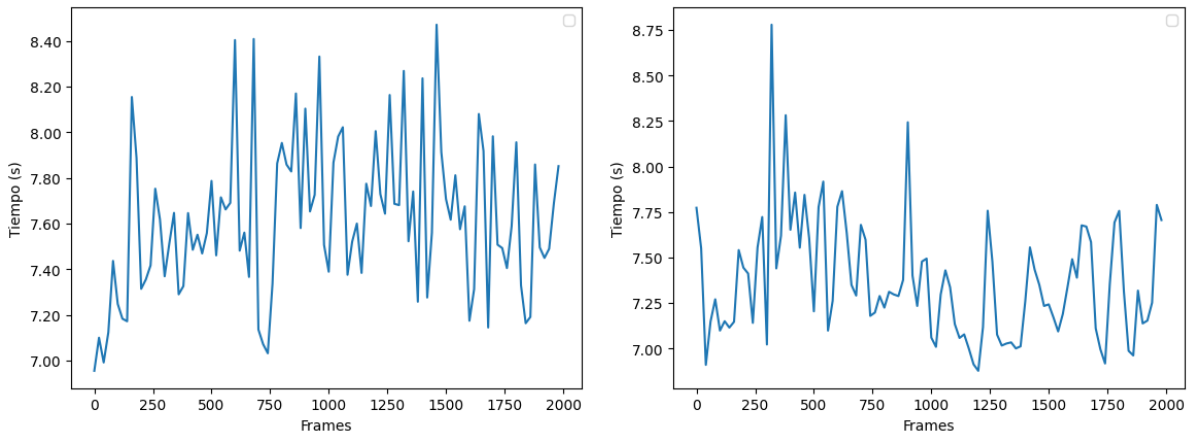
De estos resultados se puede denotar como se reduce porcentualmente el tiempo de inferencia al usar el modelo estudiante si se compara con el modelo maestro, tanto cuando se corre

con el GPU o con el CPU. De hecho, se tiene una reducción del (6,607784±1,410766) % al utilizar el GPU y un (15.331597±1.117240) % al usar el CPU. Con el fin de respaldar este resultado se adjunta las imágenes de las Figs. 5.6 y 5.7, en donde se observa la disminución cuantitativa de los resultados.



(a) Modelo maestro (media: 3,736967s) **(b)** Modelo estudiante (media: 3,204071s)

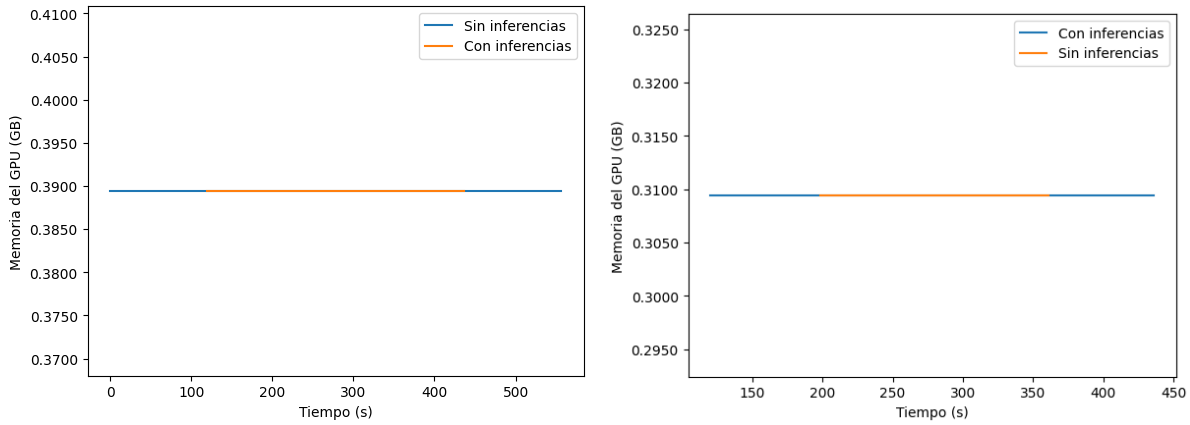
Figura 5.6: Comparación del tiempo de inferencia al correr los modelos con el CPU.



(a) Modelo maestro (media: 7,614192s) **(b)** Modelo estudiante (media: 7,369393s)

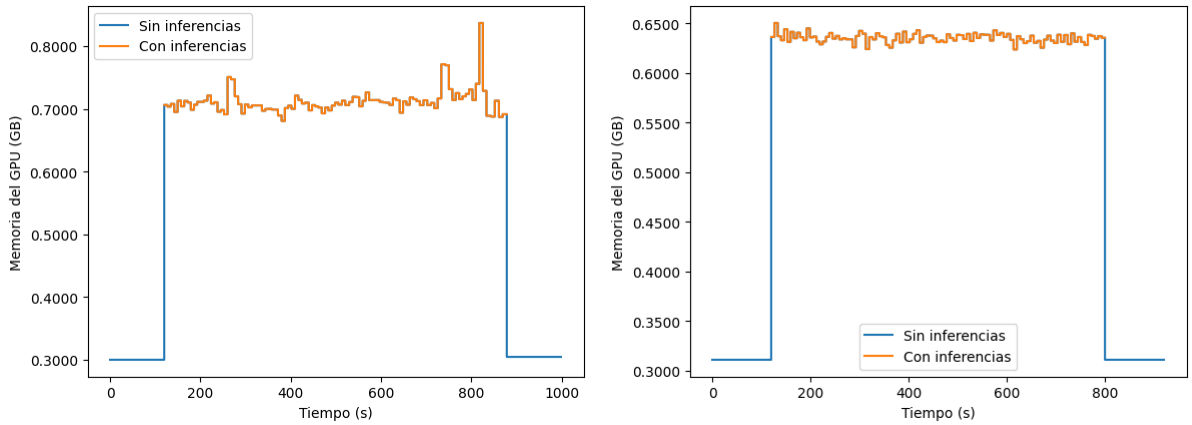
Figura 5.7: Comparación del tiempo de inferencia al correr los modelos con el GPU.

Por otra parte, para el consumo de la memoria del GPU se tiene una reducción equivalente a (14,308758±1,176106) % cuando se corre el modelo con el GPU y una reducción igual a (16,0.33880±2.626131) % cuando se lleva a cabo con el CPU. De igual manera se adjuntan las imágenes de las Figs 5.8 y 5.9 con el fin de evidenciar dicha mejora en el rendimiento del programa.



(a) Modelo maestro (media: 0,389427 GB) **(b)** Modelo estudiante (media: 0,308453 GB)

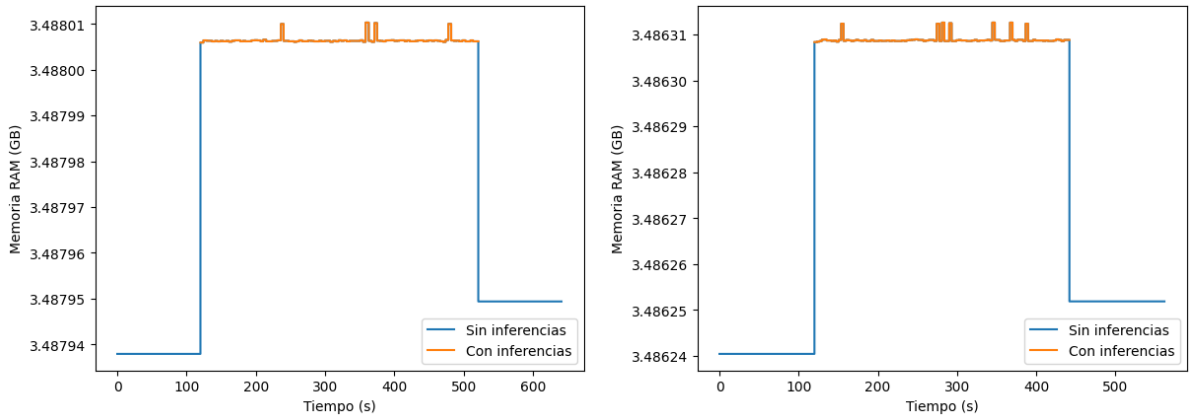
Figura 5.8: Comparación del consumo de la memoria RAM al correr los modelos con el CPU.



(a) Modelo maestro (media: 0,711363 GB) **(b)** Modelo estudiante (media: 0,635330 GB)

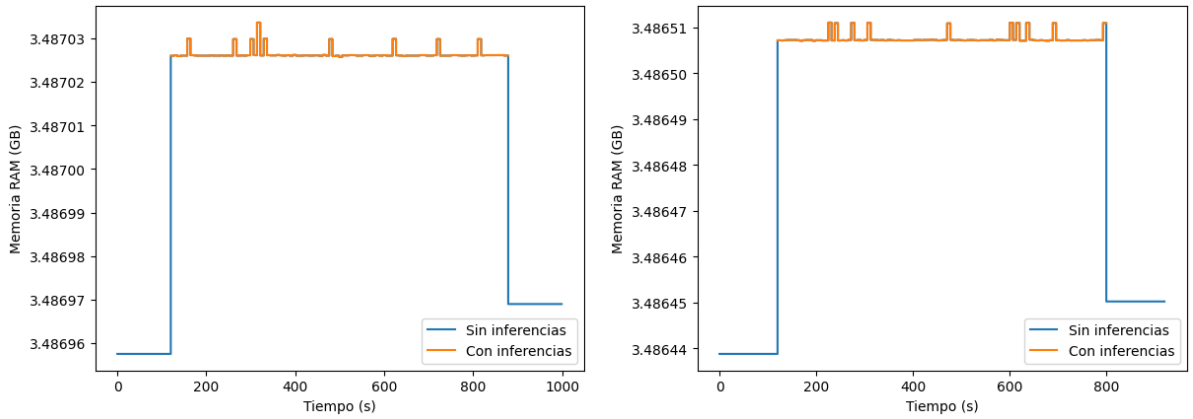
Figura 5.9: Comparación del consumo de la memoria RAM al correr los modelos con el GPU.

De igual forma, en las Tablas 5.13 y 5.14 también se evidencia la comparación entre el consumo de la memoria RAM. En este caso, los resultados indican que esta característica no disminuyó a pesar de reducir el tamaño del modelo, de hecho en el caso de la corrida con el GPU se tuvo una reducción de $(0.066889 \pm 0.035940)\,\%$ y en el de la corrida con el CPU de $(0.109476 \pm 0.065076)\,\%$, siendo estos valores casi cero. A modo de ejemplo se adjuntan las Figs. 5.10 y 5.11 en donde se aprecia la similitud en los resultados para el consumo de memoria RAM.



(a) Modelo maestro (media: 3,488006 GB) **(b)** Modelo estudiante (media: 3,486309 GB)

Figura 5.10: Comparación del consumo de memoria RAM al correr los modelos con el CPU.



(a) Modelo maestro (media: 3,487026 GB) **(b)** Modelo estudiante (media: 3,486508 GB)

Figura 5.11: Comparación del tiempo de inferencia al correr los modelos con el GPU.

Por último, se desea agregar que tal y como se aprecia en el Apéndice C, el modelo estudiante elegido presenta 4 644 477 de parámetros y como el modelo maestro posee 18 530 545 de parámetros, se tiene una reducción en los resultados mostrados anteriormente equivalente a 74.94 %.

5.3. Análisis de los resultados

5.3.1. Análisis cuantitativo

A modo de resumen, se muestra la Tabla 5.15 con los resultados finales obtenidos. De ella se puede analizar como al disminuir el tamaño de la red se tiene una reducción significativa en el tiempo de inferencia y la memoria consumida por el modelo al correrlo con el CPU.

De manera similar pero con menor alcance, se logró reducir el consumo de memoria y el tiempo de inferencia al correr el modelo por medio del GPU.

Tabla 5.15: Resumen de los resultados finales de la comparación del rendimiento del modelo estudiante respecto al modelo maestro

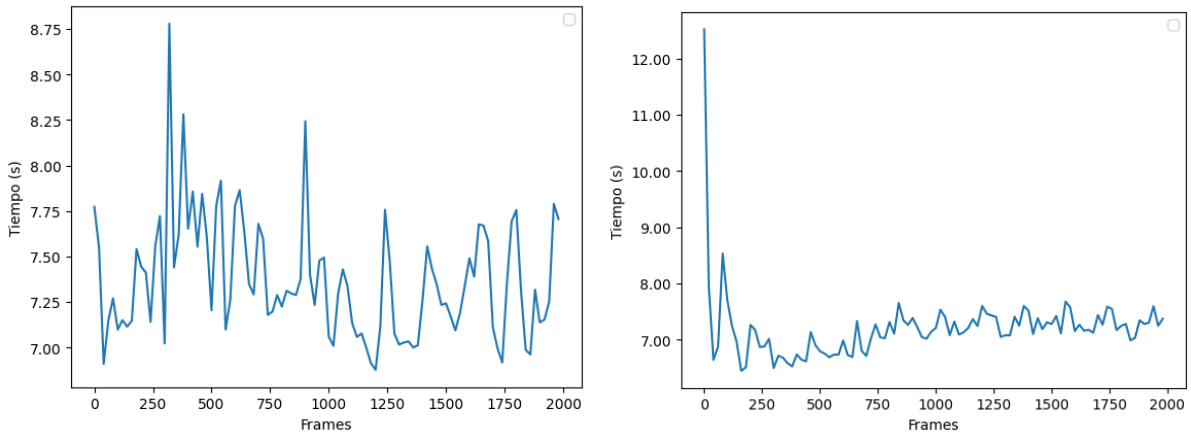
N			**1**	**2**	**3**	**4**	**5**	**6**	**Promedio**	**Error**
Conjunto de datos de entrenamiento			**Día 1**	**Día 3**	**Día 1**	**Día 2**	**Día 2**	**Día 3**		
Conjunto de datos de validación			**Día 2**	**Día 2**	**Día 3**	**Día 3**	**Día 1**	**Día 1**		
mAP			-7,1834	-6,6825	-5,7582	-5,4065	-6,9961	-5,3476	**-6,2291**	0,7506
Tiempo de inferencia		GPU run	-5,4066	-5,7168	-8,8237	-8,3425	-5,8496	-5,5076	**-6,6078**	1,4108
		CPU run	-16,4766	-16,4596	-16,3041	-14,0723	-14,7867	-13,8903	**-15,3316**	1,1172
Memoria del programa	Parámetros		-74,9361	-74,9361	-74,9361	-74,9361	-74,9361	-74,9361	**-74,9361**	0,0000
	GPU run	Memoria GPU	-12,2276	-14,9903	-14,1684	-15,0109	-15,8766	-13,5788	**-14,3088**	1,1761
		RAM	-0,0981	-0,1207	-0,0439	-0,0819	-0,0370	-0,0197	**-0,0669**	0,0359
	CPU run	Memoria GPU	-11,0976	-15,1386	-16,0476	-18,0701	-16,3571	-19,4922	**-16,0339**	2,6261
		RAM	-0,0492	-0,1571	-0,1325	-0,0304	-0,0719	-0,2159	**-0,1095**	0,0651

Estos resultados evidencian que se cumplen con los objetivos iniciales del proyecto de reducir la red sin perder sin perder drásticamente la precisión, ya que:

- En términos de la memoria: se redujo el tamaño de la red en un 74.93 %, se disminuyó el consumo de la memoria del GPU en un 14.3088 % al correr el modelo con el GPU y en un 16.0339 % al hacer inferencias en el CPU, siendo todos esto valores superiores al porcentaje de reducción de memoria planteado por el cliente (10 %).
- En términos del tiempo de inferencia: se redujo el tiempo en un 15.3316 % y 6.2237 % al correr el modelo en el CPU y GPU, respectivamente y de la misma manera el cliente estuvo satisfecho con el trabajo realizado debido a que se evidenció que fue posible reducir el tiempo de inferencia más allá del 10 % al utilizar el CPU.
- En términos de la precisión: se mantuvo una precisión no menor al 10 % respecto al modelo maestro, teniendo una reducción del 6.221 % en los resultados.

Además, se conversó con el cliente experto con el fin de comprender el comportamiento de las curvas del tiempo de inferencia de las Figs. 5.6 y 5.7, ya que, resultó curioso que el tiempo de inferencia fuera variante entre el conjunto de imágenes. En este caso, se tomó en consideración que el modelo YOLOv6 3.0 realiza una supresión no máxima antes de retornar los cuadros delimitadores resultantes, por lo que, este aspecto podría influir en

el tiempo de inferencia que requiere cada imagen. De manera similar se argumentó que el factor de la cantidad de personas presentes en la imagen también podía aumentar el tiempo de procesamiento del modelo. Por lo que se hizo un análisis del tiempo de inferencia del modelo utilizando 2000 veces la misma imagen con el fin de analizar si existía una consistencia en los resultados. En este caso, se adjuntan las imágenes de la Fig. 5.12.



(a) Inferencias con 2000 imágenes distintas **(b)** 2000 inferencias con la misma imagen

Figura 5.12: Comparación del tiempo de inferencia al realizar inferencias con 2000 imágenes del conjunto de datos respecto a una misma imagen.

Estas imágenes evidencian gráficamente que existe una mayor consistencia en los resultados al realizar la prueba con la misma imagen, de hecho al comparar la desviación estándar de ambas pruebas se tiene que para la prueba con 2000 distintas imágenes este valor es de 0.6351, mientras que en el caso de la prueba con la misma imagen, al obviar la primera inferencia, se tiene una desviación estándar de 0.2862.

Por otro lado, el resultado del consumo de la memoria RAM también es un tema que salió a relucir, por lo que, se hizo un estudio con el fin de lograr entender por qué se seguía manteniendo este parámetro a pesar de reducir drásticamente el tamaño del modelo. Al analizar el funcionamiento de la librería de Python se notó que esta calcula como consumo de la memoria RAM el de la memoria RAM física sumado al de la memoria SWAP.

Además, cuando se ejecuta el sistema operativo de una computadora este le asigna a cada programa una porción de la memoria RAM para ser utilizado y en los casos que se excede dicho consumo, la memoria SWAP, que es un tipo de memoria RAM virtual, se utiliza como extensión de la memoria RAM para poder llevar a cabo las tareas. [79][80] Por lo tanto, se decidió cuantificar los resultados del modelo maestro y del modelo estudiante para la memoria RAM física al hacer la modificación del Algoritmo 13 (para mayor detalle visualice la Tabla F.1 del Apéndice F) y con base en estos resultados, se calcularon los valores presentes en la Tabla 5.16.

Tabla 5.16: Comparación entre el consumo de memoria RAM física entre el modelo maestro y el modelo estudiante

N	Maestro		Estudiante		Reducción (%)	
	CPU run	**GPU run**	**CPU run**	**GPU run**	**CPU run**	**GPU run**
1	0,263624	0,254750	0,113534	0,124467	-56,933432	-51,141441
2	0,247166	0,265228	0,123201	0,122193	-50,154579	-53,929037
3	0,261966	0,251089	0,132201	0,124814	-49,534998	-50,290800
4	0,249533	0,252126	0,123765	0,118209	-50,401383	-53,115106
5	0,253021	0,261389	0,128518	0,131441	-49,206427	-49,714196
6	0,261612	0,263079	0,125644	0,124783	-51,972962	-52,568232
Promedio	0,256154	0,257943	0,124477	0,124318	**-51,367297**	**-51,793135**
Error	0,006504	0,005513	0,005763	0,003938	**2,638783**	**1,522835**

De estos resultados se aprecia como la memoria RAM física disminuye al aplicar la técnica de entrenamiento-estudiante, inclusive su reducción es un poco mayor al 50 % en ambos casos.

5.3.2. Análisis cualitativo

De igual manera, se conversó con el cliente experto respecto a las razones que pueden existir para que el modelo consuma más tiempo al realizar inferencias con el GPU que con el CPU. En este caso, se toma en consideración que el modelo al hacer estas pruebas está cargando constantemente una imagen de una dirección de memoria, por lo que, este proceso consume de cierto tiempo y como las GPU están destinadas a realizar en paralelo pequeñas tareas, es posible que esté consumiendo mayor tiempo en cargar dichas imágenes que la CPU.

De igual manera, cabe destacar, que el programa diseñado en el Algoritmo 13 toma las estadísticas de la memoria consumida por el GPU cuando el modelo realiza inferencias, pero, no se ejecuta directamente a través de este hardware, por lo que puede que la transferencia de los datos entre el GPU y CPU conlleve en un incremento del tiempo de procesamiento.

De hecho en esta línea argumentativa [81] menciona que antes de la ejecución del GPU, el CPU carga los datos desde la memoria principal hasta la memoria del GPU, luego se realiza la ejecución del kernel del GPU y por último se transportan los resultados a la memoria del CPU (para mayor detalle apreciar la Fig. 5.13).

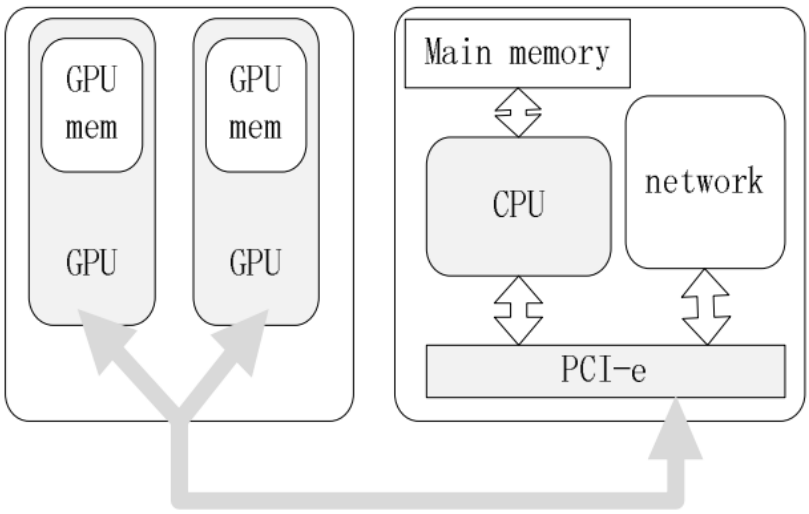


Figura 5.13: Comunicación entre la memoria del GPU y la memoria del CPU. Fuente: [81]

Además, [82] comenta que la comunicación entre un GPU y CPU comúnmente presenta algunos problemas debido a que como poseen memorias aparte, se deben de comunicar constantemente (lo cual significa que está en una constante lectura y escritura) y este proceso se encuentra limitado por el ancho de banda disponible. Por lo tanto, se toma en consideración estas características para dar un sentido al comportamiento presentado en la diferencias en los tiempos de inferencia entre el GPU y el CPU.

5.4. Análisis del potencial hardware a utilizar

Por último, con base en los resultados obtenidos y deseados por el cliente, se debe demostrar que el modelo resultante tiene el posible potencial de ser implementado en algún hardware dentro del mercado con un precio cercano (o menor) a 150 000 colones. Para ello, se toma en consideración que el modelo puede ser corrido a través de GPU o CPU, lo cual incrementa las posibles opciones de sistemas embebidos por utilizar.

En la Tabla 5.17 se presentan algunas posibles opciones que el cliente puede llegar a utilizar para llevar a cabo el empotramiento del programa tomando en consideración que:

- El consumo de la memoria RAM es de 3,49 GB (valor proveniente de la corrida del modelo con el CPU).
- El consumo de la memoria del GPU es de 0,60 GB (valor proveniente de la corrida del modelo con el GPU).

Tabla 5.17: Recomendaciones de sistemas embebidos a utilizar. Fuentes: [83][84][85][86][87][88][89]

Dispositivo	Memoria RAM (GB)	Memoria GPU (GB)	Precio ($)
Jetson Nano Developer Kit	4	Integrado en la RAM	212
Raspberry Pi 4	4	Integrado en la RAM	163
BeagleBone AI	4	-	105
Odroid-N2	4	Integrado en la RAM	182

Cualquiera de estas cuatro opciones presenta la capacidad de soportar el modelo estudiante resultante con base en el consumo de la memoria analizado. Además, se adjunta su precio en dolares debido al posible cambio del colón costarricense. Tomando en consideración que el tipo de cambio para el 25 de mayo del 2023, según [90] 212$ equivalen a 115 540 colones, lo que significa que cualquiera de las opciones se adecuan al presupuesto del cliente.

5.5. Análisis Económico

Este proyecto es un trabajo meramente de investigación, por lo que no se requirió de algún insumo físico para su desarrollo. Por lo tanto, en la Tabla 5.18 se detallan los eventuales costos requeridos para replicar el trabajo. En el caso de las horas laboradas se tomó en consideración una jornada de ocho horas durante de lunes a viernes durante 15 semanas.

Tabla 5.18: Costos asociados al proyecto

Recurso	Cantidad	Costo	Subtotal (colones)
Renumeración económica	600	3 668,29/hora	2 200 947
Computadora portatil	1	500 000/unidad	500 000
Computadora de escritorio	1	1 500 000/unidad	1 500 000
Asistencia de investigación	4	57 500/mes	230 000
	Total		4 430 947

5.5.1. Beneficio académico

Por otra parte, se tienen beneficios a raíz del desarrollo del proyecto, en este caso, se puede variar el enfoque en tres aristas: la primera en el avance investigativo desarrollado en el área de la Inteligencia Artificial, la segunda en las potenciales aplicaciones del proyecto de investigación y la tercera en el avance investigativo proporcionado hacia el laboratorio de investigación SIPLAB.

Para el primer punto, se logró desarrollar un método de entrenamiento maestro-estudiante capaz de reducir el tamaño de una red específica y por ende reducir su consumo de memoria y tiempo de inferencia, a la vez que se preservó su precisión. Es decir, se desarrolló un método potencialmente capaz de facilitar la implementación de un modelo de Inteligencia Artificial en un sistema embebido de bajo coste.

Para la segunda arista se tiene un avance significativo en la eventual implementación de estos dispositivos en espacios físicos con peligros latentes. Se brindó herramientas para eventualmente llevar a cabo el control automático de aforo en lugares con potenciales peligros. A continuación se mencionan algunos posibles ejemplos:

- Control de aforo en un elevador, en donde se puede, por medio de la detección de personas, controlar y restringir la máxima cantidad de individuos automáticamente, reduciendo de esta manera la capacidad de fallo de los elevadores permitiendo solamente el ingreso del aforo máximo.
- Control de aforo de eventos masivos en estadios, conciertos, bares, entre otros lugares, en donde por la sobreventa de localidades puede existir una sobrepoblación y resultar en un accidente fatídico.
- Identificación de personas en peligro. En este caso se le puede dar una posible aplicación al implementar un sistema de detección de personas en escuelas, colegios y universidades, y ante un eventual sismo o un incendio, poder detectar cuántas personas hacen falta de evacuar en sitios específicos.

Por último, y muy ligado de la mano con los dos puntos anteriores, se implementó una aplicación de entrenamiento maestro-estudiante en redes neuronales para que posteriormente, algún otro investigador, tenga los insumos para continuar con el trabajo ligado a la VIE del ITCR de la *Detección automática in situ de aforo en video para retorno a presencialidad debido a la pandemia de la COVID19*, en donde se ajustó una red de neuronal para que fuera capaz de detectar personas desde una vista superior. Además, la componente investigativa del proyecto le posibilita al laboratorio la publicación de un documento de investigación (conocidos como papers en inglés) que puede potencialmente atraer nuevos ingresos.

A pesar de que estos aspectos no son renumerables en este momento, se espera que en un futuro tengan un beneficio económico y social en la sociedad costarricense.

5.6. Comparación respecto al estado del arte

Al comparar los resultados obtenidos por la aplicación desarrollada contra aplicaciones del estado del arte se pueden tener alguna ventajas y desventajas evidentes. En primera instancia, la aplicación propuesta posee la desventaja de que, a pesar de ser desarrollada de la mejor manera posible, no es posible asegurar que se maximizan los recursos computacionales disponibles para llevar a cabo el entrenamiento. Caso contrario, sucede con las bibliotecas que proporcionan TensorFlow Lite y TensorFlow Model Optimization Toolkit, en donde, se poseen a grupos de ingenieros que se encargan de producir bibliotecas totalmente optimizadas para facilitar su uso.

Por otro lado, las herramientas proporcionadas por TensorFlow no permiten implementar un modelo en cualquier dispositivo electrónico, sino que, se limita su uso a los dispositivos que se encuentran afiliados con la aplicación, mientras que, en el caso de la aplicación desarrollada, la implementación del modelo puede realizarse en cualquier dispositivo electrónico, siempre y cuando cumpla con los requerimientos mínimos de memoria.

Por último, a pesar de que TensorFlow Model Optimization Toolkit y TensorFlow Lite brindan opciones que facilitan la reducción de modelos de IA, no necesariamente corresponden a las necesidades que se requieren en la industria, por lo que, una aplicación de propósito específico como la desarrollada en este documento puede presentar diversos beneficios respecto a las necesidades puntuales de los clientes.

Capítulo 6

Conclusiones

- Se logró elegir un modelo maestro capaz de ajustarse al conjunto de datos brindado por el laboratorio de investigación SIPLAB y de esta manera poder detectar personas automáticamente desde una vista superior.
- Fue posible escoger una técnica de entrenamiento maestro-estudiante con la que se redujo el tamaño del modelo maestro.
- Se desarrolló una variante de la destilación de conocimiento con la que se generó un modelo estudiante capaz de reducir el tiempo de inferencia y el consumo de memoria del modelo, mientras que se preservaba su precisión.
- Se validó el modelo estudiante resultante, luego de desarrollar el programa por medio de tres pruebas de validación: una validación cruzada para magnificar el valor de la precisión del modelo estudiante, otra validación cruzada para puntualizar el tiempo de inferencia y la memoria consumida por el modelo estudiante, y una comparación de la reducción del modelo estudiante respecto al modelo maestro.
- Se pasó de un valor de mAP de (27.9606±2.4756) % a (26.2122±2.2481) % luego de llevar a cabo el entrenamiento maestro-estudiante en una red neuronal capaz de detectar automáticamente a personas.
- Se redujo el tiempo de inferencia luego de la destilación de conocimiento de (7,595346 ± 0,006328)s a (7,093424±0,049611)s al correr el modelo con el GPU y de (3,863604 ± 0,038640)s a (3,270977±0,013872)s al correr el modelo con el CPU.
- Se disminuyó el tamaño de la red estudiante en un 74.94 % como resultado del entrenamiento maestro-estudiante realizado.
- Se bajó el consumo de memoria del GPU en un 16.0339 % al correr el modelo con el CPU y en un 14,3088 % al correr el modelo con el GPU.

6.0.1. Trabajo futuro

- Desarrollar más pruebas preliminares y exhaustivas con el fin de profundizar en mayor medida los posibles resultados del modelo estudiante.
- Investigar por medio de un estudio de hardware el tiempo consumido durante la comunicación de la memoria del GPU con la memoria del CPU.
- Empotrar el modelo final resultante en un sistema embebido de bajo coste.
- Exponer el modelo entrenado a condiciones en vivo con el fin de determinar su rendimiento antes posibles ruidos no existentes dentro del conjunto de datos brindado por SIPLAB.
- Realizar nuevas variaciones en modelos estudiantes con el fin de encontrar uno de menor tamaño que tenga resultados similares en términos de la precisión en caso de ser requerido.

Bibliografía

[1] P. Alvarado, *SIP-Lab*, 2011. dirección: http://palvarado.ietec.org/pmwiki/index.php/Main/SIP-Lab (visitado 28-02-2023).

[2] TEC. «Vicerrectoría de Investigación y Extensión,» Tecnológico de Costa Rica. (2023), dirección: https://www.tec.ac.cr/unidades/vicerrectoria-investigacion-extension (visitado 28-02-2023).

[3] S. Theodoridis y K. Koutroumbas, *Pattern Recognition*, 4th. Academic Press, 2008.

[4] J. F. V. Serrano, A. B. M. Díaz, Á. S. Calle y J. L. E. Sánchez-Marín, *Visión por Computador*. Delta Publicaciones, 2012.

[5] J. L. C. Mariño, *Tema XCP04: Introducción a la clasificación en imágenes*, Área Académica Ingeniería Mecatrónica. Sistemas de Visión. Instituto Tecnológico de Costa Rica., 2022.

[6] L. Rouhiainen, «Inteligencia artificial,» *Madrid: Alienta Editorial*, 2018.

[7] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd. John Wiley & Sons, Ltd., 2007.

[8] G. Rebala, A. Ravi y S. Churiwala, *An Introduction to Machine Learning*. CRC Press, 2019.

[9] J. Alzubi, A. Nayyar y A. Kumar, «Machine Learning from Theory to Algorithms: An Overview,» *Journal of Physics: Conference Series*, vol. 1142, n.º 1, pág. 012 012, nov. de 2018. DOI: 10.1088/1742-6596/1142/1/012012.

[10] T. Dietterich, «Overfitting and undercomputing in machine learning,» *ACM computing surveys (CSUR)*, vol. 27, n.º 3, págs. 326-327, 1995.

[11] MathWorks. «Overfitting in Machine Learning: What It Is and How to Prevent It,» The MathWorks, Inc. (2021), dirección: https://www.mathworks.com/discovery/overfitting.html (visitado 21-03-2023).

[12] S. K. Noon, M. Amjad, M. A. Qureshi y A. Mannan, «Overfitting Mitigation Analysis in Deep Learning Models for Plant Leaf Disease Recognition,» en *2020 IEEE 23rd International Multitopic Conference (INMIC)*, 2020, págs. 1-5. DOI: 10.1109/INMIC50486.2020.9318044.

[13] X. Ying, «An overview of overfitting and its solutions,» en *Journal of physics: Conference series*, IOP Publishing, vol. 1168, 2019, pág. 022 022.

[14] H. Jabbar y R. Z. Khan, «Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study),» *Computer Science, Communication and Instrumentation Devices*, vol. 70, págs. 163-172, 2015.

[15] Z. Reitermanova et al., «Data splitting,» en *WDS*, Matfyzpress Prague, vol. 10, 2010, págs. 31-36.

[16] Neptune. «Cross-Validation in Machine Learning: How to Do It Right.» (oct. de 2020), dirección: https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right (visitado 06-03-2023).

[17] Alteryx Community. «Holdouts and Cross-Validation: Why the Data Used to Evaluate your Model Matters.» (sep. de 2020), dirección: https://community.alteryx.com/t5/Data-Science/Holdouts-and-Cross-Validation-Why-the-Data-Used-to-Evaluate-your/ba-p/448982?lightbox-message-images-448982=71548iB33D10C7233B394A (visitado 06-03-2023).

[18] S. Paraskar, A. S. Pathan, R. Parteki, R. A. Jichkar, L. Thakare y T. Deotale, «A Review on Deep Learning Methods and CT Scan Approaches for Covid-19 Detection,» en *2022 10th International Conference on Emerging Trends in Engineering and Technology - Signal and Information Processing (ICETET-SIP-22)*, 2022, págs. 1-6. DOI: 10.1109/ICETET-SIP-2254415.2022.9791687.

[19] S. Singh. «Cousins of Artificial Intelligence,» Medium. (2020), dirección: https://towardsdatascience.com/cousins-of-artificial-intelligence-dda4edc27b55 (visitado 21-03-2023).

[20] L. A. C. Salazar, D. J. M. Aldana y J. A. C. Montes, «Implementación de redes neuronales y lógica difusa para la clasificación de patrones obtenidos por un Sónar,» en *2013 II International Congress of Engineering Mechatronics and Automation (CIIMA)*, 2013, págs. 1-6. DOI: 10.1109/CIIMA.2013.6682787.

[21] D. A. Sousa, *Neurociencia educativa: Mente, cerebro y educación*. Narcea Ediciones, 2014, vol. 131.

[22] J. Crespo y R. Loaiza, *Tema 2: Redes Neuronales Artificiales (I)*, Cartago, Costa Rica, 2021.

[23] J. Noguera, N. Portillo y L. Hernandez, «Redes Neuronales, Bioinspiración para el Desarrollo de la Ingeniería,» *Ingeniare*, n.º 17, págs. 117-131, jul. de 2014. DOI: 10.18041/1909-2458/ingeniare.17.584. dirección: https://revistas.unilibre.edu.co/index.php/ingeniare/article/view/584.

[24] H. Li, J. Li, X. Guan, B. Liang, Y. Lai y X. Luo, «Research on Overfitting of Deep Learning,» en *2019 15th International Conference on Computational Intelligence and Security (CIS)*, 2019, págs. 78-81. DOI: 10.1109/CIS.2019.00025.

[25] V. M. Lupea, «Multi-Valued Neuron with a periodic activation function - as part of a multi-layered Neural Network,» en *2013 IEEE 11th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2013, págs. 121-124. DOI: 10.1109/SAMI.2013.6480958.

[26] M. Abdolshah, R. M. Yusuff, M. Y. B. Ismail y T. S. Hong, «A New Technique to Measure Process Capability with Taguchi Loss Functions,» en *2009 International Conference on Information Management and Engineering*, 2009, págs. 186-190. DOI: 10.1109/ICIME.2009.123.

[27] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information science and statistics), 1st ed. 2006. Corr. 2nd printing. Springer, 2006, ISBN: 978-0387310732; 0387310738.

[28] K. Kida, S. Ubukata, A. Notsu y K. Honda, «Comparison of Gradient Descent Methods in Online Fuzzy Co-clustering,» en *2019 International Conference on Fuzzy Theory and Its Applications (iFUZZY)*, 2019, págs. 9-14. DOI: 10.1109/iFUZZY46984.2019.9066194.

[29] X. Zhai y F. Qiao, «A Deep Learning Model with Adaptive Learning Rate for Fault Diagnosis,» en *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*, 2020, págs. 668-673. DOI: 10.1109/DDCLS49620.2020.9275094.

[30] D. S. Comas, A. Amalfitano, G. J. Meschino y V. L. Ballarin, «Interpretación y visualización de características en texturas mediante Redes Neuronales Convolucionales,» en *2022 IEEE Biennial Congress of Argentina (ARGENCON)*, 2022, págs. 1-8. DOI: 10.1109/ARGENCON55245.2022.9939876.

[31] MathWorks. «Convolution,» The MathWorks, Inc. (2023), dirección: https://la.mathworks.com/discovery/convolution.html (visitado 22-04-2023).

[32] H. Wei, Z. Wang y G. Hua, «Dynamically Mixed Group Convolution to Lighten Convolution Operation,» en *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 2021, págs. 203-206. DOI: 10.1109/ICAIBD51990.2021.9459076.

[33] G. Lu, W. Zhang y Z. Wang, «Optimizing GPU Memory Transactions for Convolution Operations,» en *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, págs. 399-403. DOI: 10.1109/CLUSTER49012.2020.00050.

[34] Á. Artola Moreno, «Clasificación de imágenes usando redes neuronales convolucionales en Python,» 2019.

[35] L. Gao, P.-Y. Chen y S. Yu, «Demonstration of Convolution Kernel Operation on Resistive Cross-Point Array,» *IEEE Electron Device Letters*, vol. 37, n.º 7, págs. 870-873, 2016. DOI: 10.1109/LED.2016.2573140.

[36] M. Kaloev y G. Krastev, «Comparative Analysis of Activation Functions Used in the Hidden Layers of Deep Neural Networks,» en *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2021, págs. 1-5. DOI: 10.1109/HORA52670.2021.9461312.

[37] K. Nasiri y K. Ghiasi-Shirazi, «PowerLinear Activation Functions with application to the first layer of CNNs,» en *2021 11th International Conference on Computer Engineering and Knowledge (ICCKE)*, 2021, págs. 531-536. DOI: 10.1109/ICCKE54056.2021.9721450.

[38] A.-D. Nguyen, S. Choi, W. Kim, S. Ahn, J. Kim y S. Lee, «Distribution Padding in Convolutional Neural Networks,» en *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, págs. 4275-4279. DOI: 10.1109/ICIP.2019.8803537.

[39] G. Liu, K. J. Shih, T.-C. Wang et al., *Partial Convolution based Padding*, 2018. arXiv: 1811.11718 [cs.CV].

[40] D. Filippas, C. Nicopoulos y G. Dimitrakopoulos, «LeapConv: An Energy-Efficient Streaming Convolution Engine with Reconfigurable Stride,» en *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, págs. 200-205. DOI: 10.1109/ISVLSI54635.2022.00047.

[41] N. Devi y B. Borah, «Cascaded pooling for Convolutional Neural Networks,» en *2018 Fourteenth International Conference on Information Processing (ICINPRO)*, 2018, págs. 1-5. DOI: 10.1109/ICINPRO43533.2018.9096860.

[42] Z. Yu, S. Dai e Y. Xing, «Adaptive Salience Preserving Pooling for Deep Convolutional Neural Networks,» en *2019 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, 2019, págs. 513-518. DOI: 10.1109/ICMEW.2019.00094.

[43] D. Shah. «Mean average precision (MAP) explained: Everything you need to know,» V7. (2022), dirección: https://www.v7labs.com/blog/mean-average-precision#:~:text=let's%5C%20dive%5C%20in!-,What%5C%20is%5C%20Mean%5C%20Average%5C%20Precision%5C%20(mAP)%5C%3F,values%5C%20from%5C%200%5C%20to%5C%201 (visitado 07-05-2023).

[44] M. M. Dasari y R. K. S. S. Gorthi, «IOU – Siamtrack: IOU Guided Siamese Network For Visual Object Tracking,» en *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, págs. 2061-2065. DOI: 10.1109/ICIP40778.2020.9191188.

[45] Y. Xiong, «Building text hierarchical structure by using confusion matrix,» en *2012 5th International Conference on BioMedical Engineering and Informatics*, 2012, págs. 1250-1254. DOI: 10.1109/BMEI.2012.6513202.

[46] R. Agarwal. «The 5 classification evaluation metrics every data scientist must know,» Medium. (2022), dirección: https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226 (visitado 07-05-2023).

[47] I. Kim y C.-H. Lee, «Optimization of average precision with Maximal Figure-of-Merit Learning,» en *2011 IEEE International Workshop on Machine Learning for Signal Processing*, 2011, págs. 1-6. DOI: 10.1109/MLSP.2011.6064638.

[48] A. Anwar. «What is average precision in object detection amp; localization algorithms and how to calculate it?» Medium. (2022), dirección: https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b#:~:text=Average%5C%20precision%5C%20is%5C%20the%5C%20area,is%5C%20between%5C%200%5C%20to%5C%201 (visitado 07-05-2023).

[49] V. Rojas, A. Venegas-Murillo y E. Monge-Román, *YOLO: you only look once: un enfoque para detección de objetos en tiempo real*, Revista Tecnología en Marcha, Universidad Latina de Costa Rica, 2018. dirección: https://revistas.ulatina.ac.cr/index.php/tecnologiavital/article/download/250/260/547.

[50] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, 2016. arXiv: 1506.02640 [cs.CV].

[51] C. Li, L. Li, Y. Geng et al., *YOLOv6 v3.0: A Full-Scale Reloading*, 2023. arXiv: 2301.05586 [cs.CV].

[52] W. Liu, D. Anguelov, D. Erhan et al., «SSD: Single Shot MultiBox Detector,» en *Computer Vision – ECCV 2016*, Springer International Publishing, 2016, págs. 21-37. DOI: 10.1007/978-3-319-46448-0_2.

[53] H. Devanathan. «The Basics of Object Detection: YOLO, SSD, R-CNN,» Medium. (2018), dirección: https://towardsdatascience.com/the-basics-of-object-detection-yolo-ssd-r-cnn-6def60f51c0b (visitado 02-03-2023).

[54] J. Du, «Understanding of Object Detection Based on CNN Family and YOLO,» *Journal of Physics: Conference Series*, vol. 1004, n.º 1, pág. 012 029, abr. de 2018. DOI: 10.1088/1742-6596/1004/1/012029. dirección: https://dx.doi.org/10.1088/1742-6596/1004/1/012029.

[55] S. Ren, K. He, R. Girshick y J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2016. arXiv: 1506.01497 [cs.CV].

[56] K. He, G. Gkioxari, P. Dollár y R. Girshick, *Mask R-CNN*, 2018. arXiv: 1703.06870 [cs.CV].

[57] S. Gao, Z.-Y. Li, Q. Han, M.-M. Cheng y L. Wang, *RF-Next: Efficient Receptive Field Search for Convolutional Neural Networks*, 2022. arXiv: 2206.06637 [cs.CV].

[58] S. Han, J. Pool, J. Tran y W. Dally, «Learning both weights and connections for efficient neural networks,» *Advances in Neural Information Processing Systems*, vol. 28, págs. 1135-1143, 2015.

[59] R. Krishnamoorthi, *Quantizing deep convolutional networks for efficient inference: A whitepaper*, 2018. arXiv: 1806.08342 [cs.LG].

[60] S. Jin, S. Di, X. Liang, J. Tian, D. Tao y F. Cappello, «DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression,» *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, n.º 2, págs. 715-727, 2021.

[61] G. Hinton, O. Vinyals y J. Dean, «Distilling the knowledge in a neural network,» en *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[62] TensorFlow-Hub, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. dirección: https://www.tensorflow.org/ (visitado 07-05-2023).

[63] K. T. Ulrich y S. D. Eppinger, *Diseño y desarrollo de productos*, 3rd ed. México: McGraw-Hill, 2005.

[64] L. Cabrera-Quiros, A. Demetriou, E. Gedik, L. v. d. Meij y H. Hung, «The MatchNMingle dataset: a novel multi-sensor resource for the analysis of social interactions and group dynamics in-the-wild during free-standing conversations and speed dates,» *Transactions on Affective Computing*, 2018.

[65] Pytorch. «PyTorch Hub,» The Linux Foundation. (2023), dirección: https://pytorch.org/hub/ (visitado 07-05-2023).

[66] Papers with code. «Papers with code - the latest in machine learning,» Meta AI. (2023), dirección: https://paperswithcode.com/ (visitado 07-05-2023).

[67] COCO. «Common objects in context,» COCO Consortium. (2021), dirección: https://cocodataset.org/ (visitado 07-05-2023).

[68] T.-Y. Lin, M. Maire, S. Belongie et al., *Microsoft COCO: Common Objects in Context*, 2015. arXiv: 1405.0312 [cs.CV].

[69] «Conda,» Anaconda, Inc. (2017), dirección: https://docs.conda.io/en/latest/ (visitado 08-05-2023).

[70] iStock, *Vista superior de personas*, iStockphoto LP, 2023. dirección: https://www.istockphoto.com/es/fotograf%5C%C3%5C%ADas-de-stock (visitado 08-06-2023).

[71] B. Rokh, A. Azarpeyvand y A. Khanteymoori, *A Comprehensive Survey on Model Quantization for Deep Neural Networks*, 2022. arXiv: 2205.07877 [cs.LG].

[72] D. Blalock, J. J. G. Ortiz, J. Frankle y J. Guttag, *What is the State of Neural Network Pruning?* 2020. arXiv: 2003.03033 [cs.LG].

[73] A. Polino, R. Pascanu y D. Alistarh, *Model compression via distillation and quantization*, 2018. arXiv: 1802.05668 [cs.NE].

[74] S. Han, H. Mao y W. J. Dally, *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*, 2016. arXiv: 1510.00149 [cs.CV].

[75] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney y K. Keutzer, *A Survey of Quantization Methods for Efficient Neural Network Inference*, 2021. arXiv: 2103.13630 [cs.CV].

[76] F. Ruffy y K. Chahal, *The State of Knowledge Distillation for Classification*, 2019. arXiv: 1912.10850 [cs.LG].

[77] T. Liang, J. Glossner, L. Wang, S. Shi y X. Zhang, *Pruning and Quantization for Deep Neural Network Acceleration: A Survey*, 2021. arXiv: 2101.09671 [cs.CV].

[78] J. Gou, B. Yu, S. J. Maybank y D. Tao, «Knowledge Distillation: A Survey,» *International Journal of Computer Vision*, vol. 129, n.º 6, págs. 1789-1819, mar. de 2021. DOI: 10.1007/s11263-021-01453-z. dirección: https://doi.org/10.1007%5C%2Fs11263-021-01453-z.

[79] S. Desireddy y D. R. Pathireddy, «Optimize In-kernel swap memory by avoiding duplicate swap out pages,» en *2016 International Conference on Microelectronics, Computing and Communications (MicroCom)*, 2016, págs. 1-4. DOI: 10.1109/MicroCom.2016.7522551.

[80] H. Bahn y J. Kim, «Reducing the Overhead of Virtual Memory Swapping by Considering Application Characteristics and Memory Situations,» en *2022 12th International Conference on Information Technology in Medicine and Education (ITME)*, 2022, págs. 434-438. DOI: 10.1109/ITME56794.2022.00099.

[81] Y. Wu, J. Song, F. Lu y F. Yin, «Communication and Memory Access Latency Characteristics of CPU/GPU Heterogeneous Cluster,» en *2012 Fourth International Conference on Computational and Information Sciences*, 2012, págs. 958-961. DOI: 10.1109/ICCIS.2012.104.

[82] R. Mokhtari y M. Stumm, «BigKernel – High Performance CPU-GPU Communication Pipelining for Big Data-Style Applications,» en *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, págs. 819-828. DOI: 10.1109/IPDPS.2014.89.

[83] NVIDIA, *Jetson Nano Developer Kit*, NVIDIA Corporation, 2023. dirección: https://store.nvidia.com/en-us/jetson/store/?page=1&limit=9&locale=en-us (visitado 24-05-2023).

[84] Amazon, *NVIDIA Jetson Nano Kit de desarrollador (945-13450-0000-100)*, Amazon.com, Inc Corporation, 2023. dirección: https://www.amazon.com/dp/B084DSDDLT?ref_=cm_sw_r_cp_ud_dp_370XPVTRPQDKEZK8E83P (visitado 24-05-2023).

[85] Amazon, *Raspberry Pi 4 computadoras modelo B de 8 GB de una sola placa de computadora adecuada para construir mini PC, robot inteligente, consola de juegos, estación de trabajo, centro de medios, etc.* Amazon.com, Inc Corporation, 2023. dirección: https://www.amazon.com/Raspberry-computadoras-computadora-construir-inteligente/dp/B0899VXM8F/ref=sr_1_3?adgrpid=83052647738&hvadid=585362632932&hvdev=c&hvlocphy=1003683&hvnetw=g&hvqmt=b&hvrand=14760661424824936021&hvtargid=kwd-972466948035&hydadcr=22333_13333537&keywords=rasberry+pi+4+computer&sr=8-3 (visitado 24-05-2023).

[86] Amazon, *Beaglebone Black Rev C (4G) Single Board Computer Junta de Desarrollo*, Amazon.com, Inc Corporation, 2023. dirección: https://www.amazon.com/-/es/Beaglebone-Black-Single-Computer-Desarrollo/dp/B00LC1924G/ref=sr_1_1?__mk_es_US=%5C%C3%5C%85M%5C%C3%5C%85%5C%C5%5C%BD%5C%C3%5C%95%5C%C3%5C%91&keywords=BeagleBone+IA&sr=8-1 (visitado 24-05-2023).

[87] *Raspberry Pi 4. Your tiny, dual-display, desktop computer*, Raspberry Pi, Inc Corporation, 2023. dirección: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/?variant=raspberry-pi-4-model-b-4gb (visitado 24-05-2023).

[88] C. Long, *BeagleBone AI Fast Track to Embedded Artificial Intelligence*, BeagleBoard.org Foundation, 2023. dirección: https://beagleboard.org/AI (visitado 24-05-2023).

[89] HardKernel, *ODROID-N2+ with 4GByte RAM*, Hardkernel co., Ltd., 2019. dirección: https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram-2/ (visitado 24-05-2023).

[90] BCCR, *Tipo de cambio de referencias y tasas*, Banco Central de Costa Rica co., Ltd., 2023. dirección: https://www.bccr.fi.cr/SitePages/Inicio.aspx (visitado 24-05-2023).

Anexo A: Comparación de técnicas de destilación

Tabla 6.1: Comparación entre técnicas de destilación de conocimiento al usar el conjunto de datos CIFAR100. Fuente: [78]

Offline Distillation							
Methods	Knowledge	Teacher (baseline)		Student (baseline)		Accuracies	
FSP	RelK	ResNet32	(64.06)	ResNet14	(58.65)	63.33	(4.68↑)
FT	FeaK	ResNet110	(73.09)	ResNet56	(71.96)	74.48	(2.52↑)
RKD	RelK, FeaK	ResNet50	(77.76)	VGG11	(71.26)	74.66	(3.40↑)
IRG	RelK	ResNet20	(78.40)	ResNet20-x0.5	(72.51)	74.64	(2.13↑)
KR	FeaK	ResNet32	(64.06)	ResNet14	(58.65)	63.95	(5.30↑)
LKD	RelK	ResNet110	(75.76)	ResNet20	(69.47)	72.63	(3.16↑)
LKD	RelK	WRN-40-2	(75.61)	WRN-16-2	(73.10)	75.44	(2.34↑)
SSKD	RelK, ResK	VGG13	(75.38)	MobileNetV2	(65.79)	71.53	(5.74↑)
SSKD	RelK, ResK	ResNet50	(79.10)	MobileNetV2	(65.79)	72.57	(6.78↑)
FN	FeaK	ResNet110	(82.01)	ResNet56	(81.73)	82.23	(0.50↑)
AdaIN	FeaK	WRN-40-4	(78.31)	WRN-16-4	(75.68)	78.25	(2.57↑)
AdaIN	FeaK	ResNet34	(77.26)	MobileNetV2	(68.36)	70.66	(2.30↑)
PAD-L2	FeaK	ResNet18	(75.86)	MobileNetV2	(68.16)	74.06	(5.90↑)
MGD	FeaK	WRN-28-4	(78.91)	WRN-28-2	(75.12)	78.82	(3.70↑)
JointRD	FeaK	ResNet18	(77.92)	plain-CNN 18	(77.44)	78.24	(0.80↑)
CTKD	RelK, FeaK	ResNet110	(72.65)	ResNet20	(68.33)	70.75	(2.42↑)
CTKD	RelK, FeaK	WRN-40-2	(75.42)	WRN-16-2	(72.27)	74.70	(2.43↑)
SemCKD	FeaK	ResNet-32x4	(79.42)	VGG13	(74.82)	79.43	(4.61↑)
SemCKD	FeaK	WRN-40-2	(75.61)	MobileNetV2	(65.43)	69.61	(4.18↑)
SemCKD	FeaK	VGG13	(74.64)	ShuffleNetV2	(72.60)	76.39	(3.79↑)
RKD	FeaK	ResNet34	(73.05)	ResNet18	(68.06)	72.82	(4.76↑)
Online Distillation							
Methods	Knowledge	Teacher (baseline)		Student (baseline)		Accuracies	
DML	ResK	WRN-28-10	(78.69)	MobileNet	(73.65)	80.28, 77.39	(3.74↑)
DML	ResK	MobileNet	(73.65)	ResNet32	(68.99)	76.13, 71.10	(8.11↑)
DCM	ResK	WRN-28-10	(81.28)	ResNet110	(73.45)	82.18, 77.01	(3.56↑)
DCM	ResK	WRN-28-10	(81.28)	MobileNet	(73.70)	83.17, 78.57	(4.87↑)
KDCL	ResK	WRN-16-2	(72.20)	ResNet32	(69.90)	75.50, 74.30	(4.40↑)
ACNs	ResK	ResNet14	(66.88)	ResNet14	(66.88)	68.40	(1.52↑)
ACNs	ResK	VGG11	(67.38)	VGG11	(67.38)	70.11	(2.73↑)
ACNs	ResK	AlexNet	(39.45)	AlexNet	(39.45	46.27	(6.8↑)2
Self-Distillation							
Methods	Knowledge	Teacher (baseline)		Student (baseline)		Accuracies	
SD	ResK	—		ResNet32	(68.39)	71.29	(2.90↑)
Tf-KD	ResK	—		ResNet18	(75.87)	77.10	(1.23↑)
Tf-KD	ResK	—		ShuffleNetV2	(70.34)	72.23	(1.89↑)
Tf-KD	ResK	—		ResNeXt29	(81.03)	82.08	(1.05↑)
CS-KD	ResK	—		ResNet18	(75.29)	78.01	(2.72↑)

ResK: basadas en las salidas

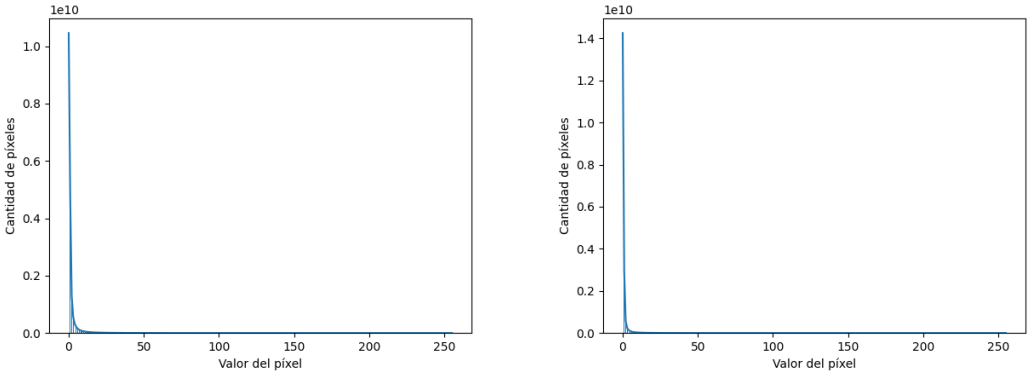
ResK: basadas en los mapas de características

Relk: basadas en relaciones entre capas

Apéndice A

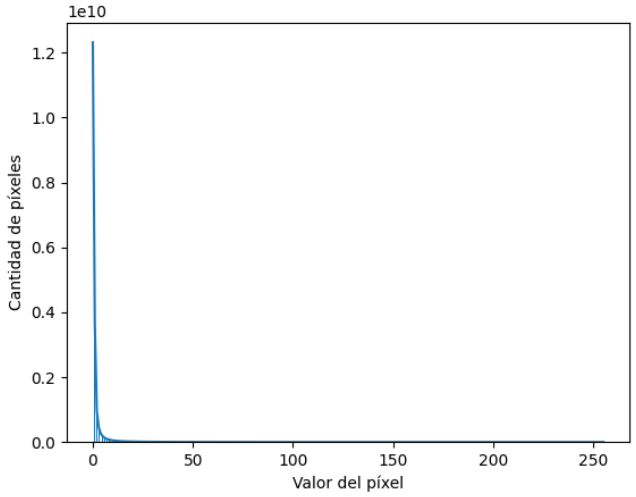
Histogramas de los videos del conjunto de datos sin reducir

A continuación, se muestran las figuras obtenidos al calcular los histogramas promedio de los videos del conjunto de datos.

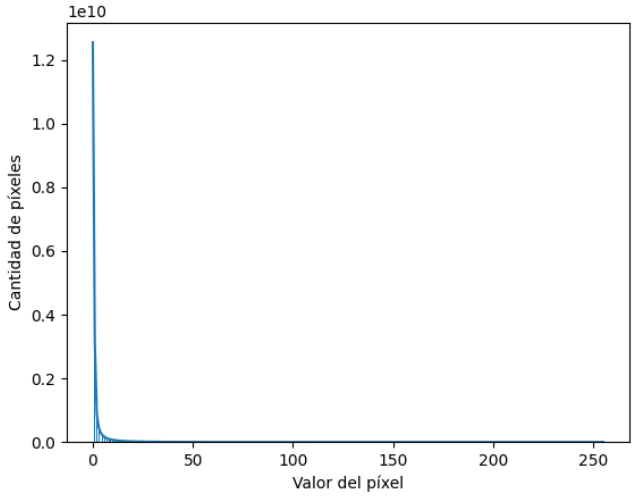


(a) Segunda cámara **(b)** Tercera cámara

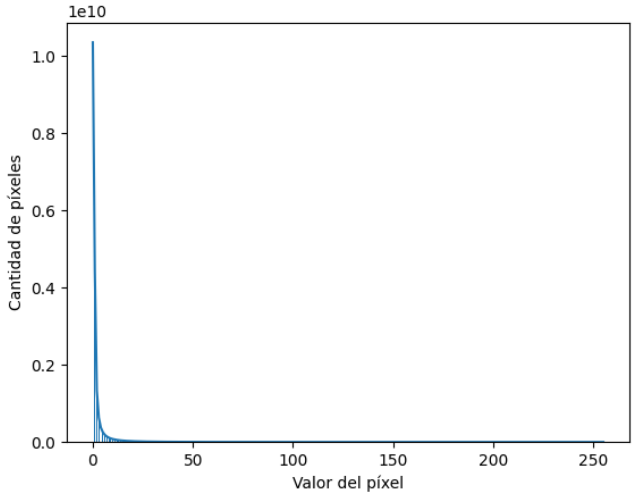
Figura A.1: Histogramas promedio de los videos grabados durante el primer día.



(a) Primera cámara

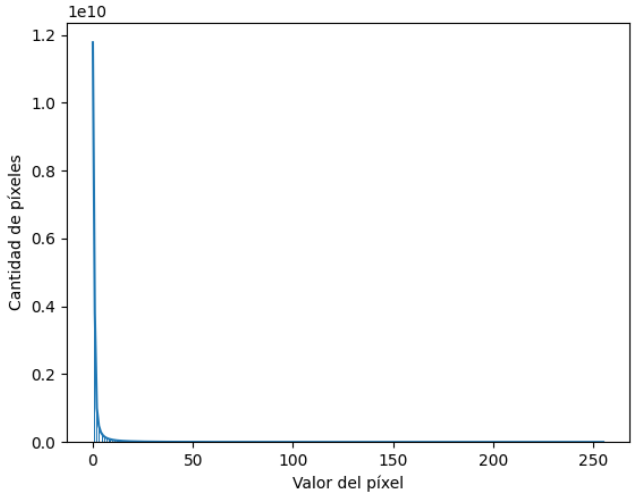


(b) Segunda cámara

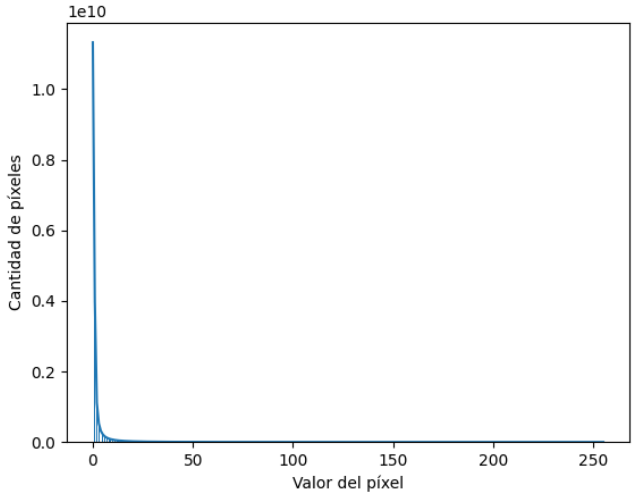


(c) Tercera cámara

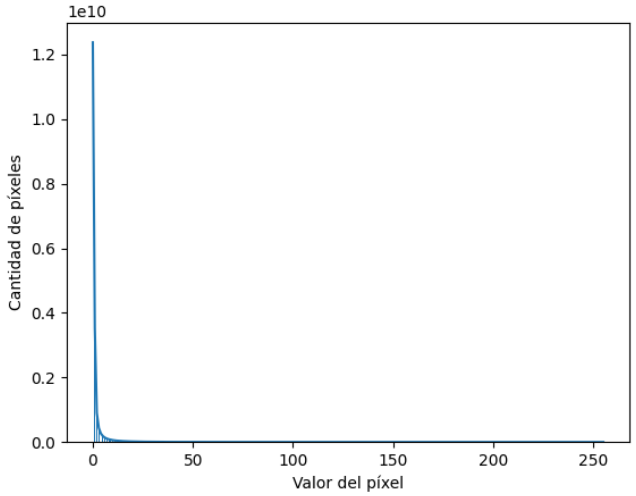
Figura A.2: Histogramas promedio de los videos grabados durante el segundo día.



(a) Primera cámara



(b) Segunda cámara

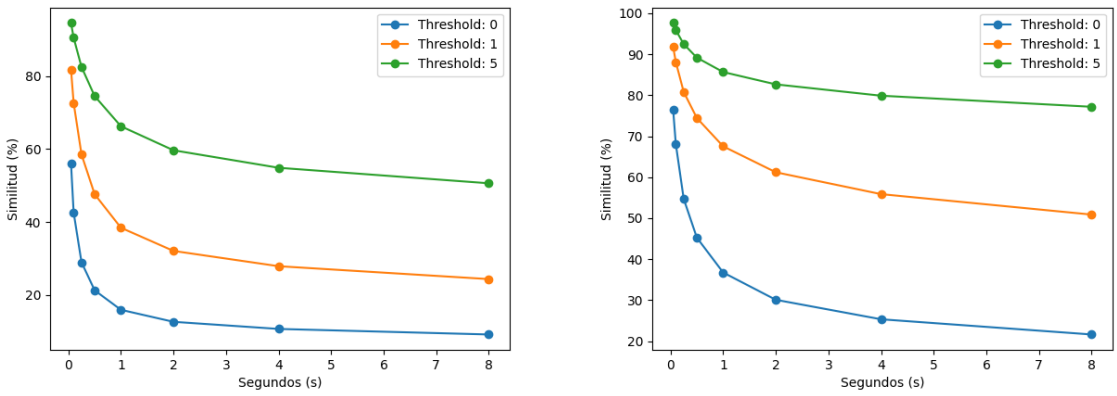


(c) Tercera cámara

Figura A.3: Histogramas promedio de los videos grabados durante el tercer día.

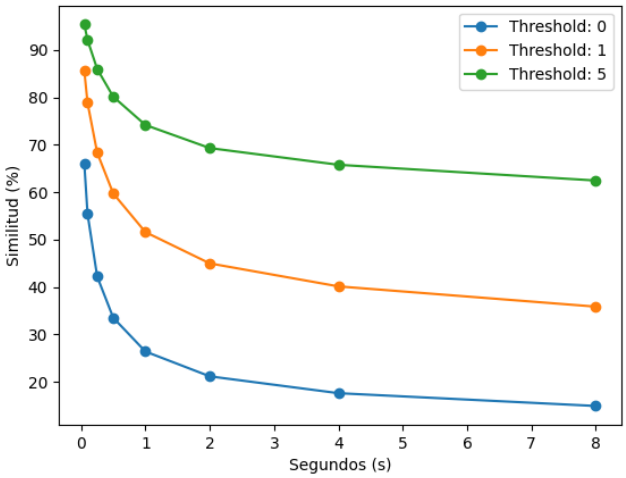
Apéndice B

Curvas de similitud para los videos del conjunto de datos sin reducir

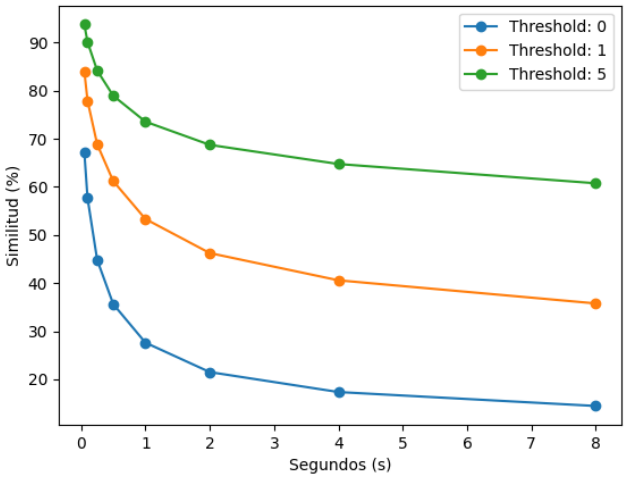


(a) Segunda cámara **(b)** Tercera cámara

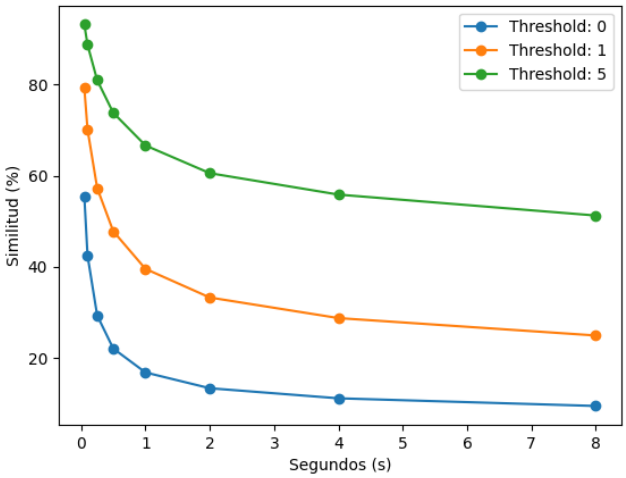
Figura B.1: Porcentaje de similitud respecto al tiempo entre imágenes durante el primer día.



(a) Primera cámara

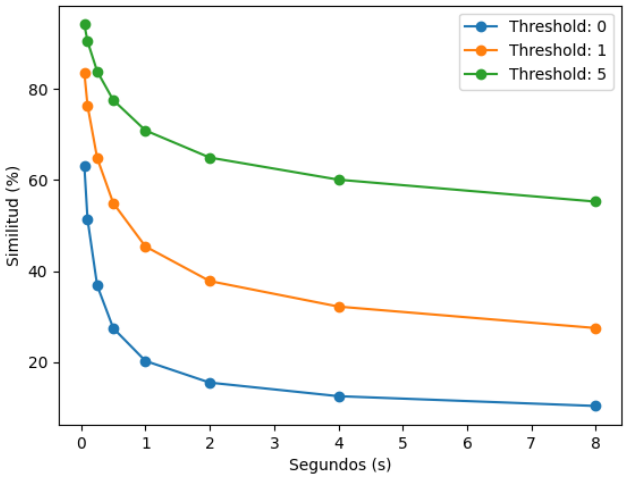


(b) Segunda cámara

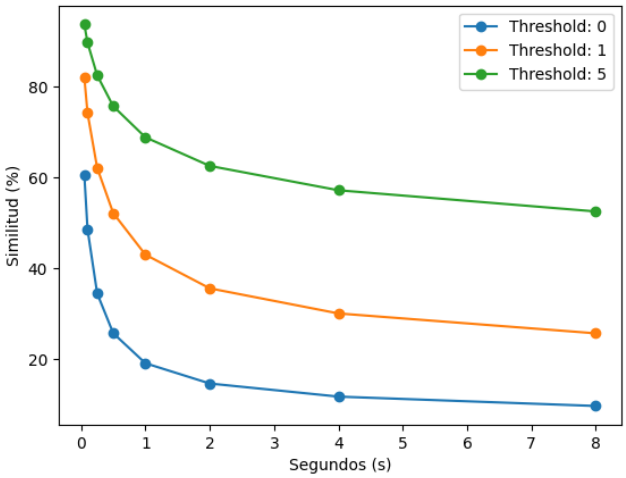


(c) Tercera cámara

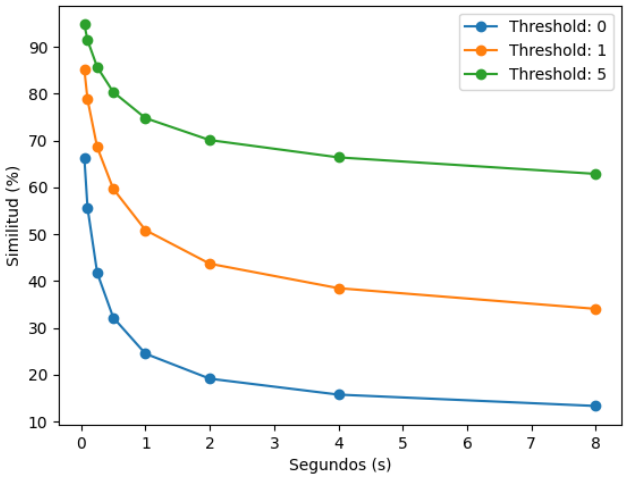
Figura B.2: Porcentaje de similitud respecto al tiempo entre imágenes durante el segundo día.



(a) Primera cámara



(b) Segunda cámara



(c) Tercera cámara

Figura B.3: Porcentaje de similitud respecto al tiempo entre imágenes durante el tercer día.

Apéndice C

Variantes de los modelos estudiantes

Tabla C.1: Variante A del modelo estudiante: 4 644 477 parámetros

Nombre	Entradas	Tamaño kernel		
backbone.stem.rbr_reparam	32	3	3	3
backbone.ERBlock_2.0.rbr_reparam	64	32	3	3
backbone.ERBlock_2.1.conv1.rbr_reparam	64	64	3	3
backbone.ERBlock_3.0.rbr_reparam	128	64	3	3
backbone.ERBlock_3.1.conv1.rbr_reparam	128	128	3	3
backbone.ERBlock_3.1.block.0.rbr_reparam	128	128	3	3
backbone.ERBlock_4.0.rbr_reparam	256	128	3	3
backbone.ERBlock_4.1.conv1.rbr_reparam	256	256	3	3
backbone.ERBlock_4.1.block.0.rbr_reparam	256	256	3	3
backbone.ERBlock_4.1.block.1.rbr_reparam	256	256	3	3
backbone.ERBlock_5.0.rbr_reparam	512	256	3	3
backbone.ERBlock_5.1.conv1.rbr_reparam	512	512	3	3
backbone.ERBlock_5.2.cv1.conv	256	512	1	1
backbone.ERBlock_5.2.cv2.conv	256	512	1	1
backbone.ERBlock_5.2.cv3.conv	256	256	3	3
backbone.ERBlock_5.2.cv4.conv	256	256	1	1
backbone.ERBlock_5.2.cv5.conv	256	1024	1	1
backbone.ERBlock_5.2.cv6.conv	256	256	3	3
backbone.ERBlock_5.2.cv7.conv	512	512	1	1
neck.reduce_layer0.conv	128	512	1	1
neck.Bifusion0.cv1.conv	128	256	1	1
neck.Bifusion0.cv2.conv	128	128	1	1
neck.Bifusion0.cv3.conv	128	384	1	1
neck.Bifusion0.downsample.conv	128	128	3	3
neck.Rep_p4.conv1.rbr_reparam	128	128	3	3
neck.Rep_p4.block.0.rbr_reparam	128	128	3	3

neck.reduce_layer1.conv	64	128	1	1
neck.Bifusion1.cv1.conv	64	128	1	1
neck.Bifusion1.cv2.conv	64	64	1	1
neck.Bifusion1.cv3.conv	64	192	1	1
neck.Bifusion1.downsample.conv	64	64	3	3
neck.Rep_p3.conv1.rbr_reparam	64	64	3	3
neck.Rep_p3.block.0.rbr_reparam	64	64	3	3
neck.downsample2.conv	64	64	3	3
neck.Rep_n3.conv1.rbr_reparam	128	128	3	3
neck.Rep_n3.block.0.rbr_reparam	128	128	3	3
neck.downsample1.conv	128	128	3	3
neck.Rep_n4.conv1.rbr_reparam	256	256	3	3
neck.Rep_n4.block.0.rbr_reparam	256	256	3	3
head.proj_conv	1	17	1	1
head.stems.0.conv	64	64	1	1
head.stems.1.conv	128	128	1	1
head.stems.2.conv	256	256	1	1
head.cls_convs.0.conv	64	64	3	3
head.cls_convs.1.conv	128	128	3	3
head.cls_convs.2.conv	256	256	3	3
head.reg_convs.0.conv	64	64	3	3
head.reg_convs.1.conv	128	128	3	3
head.reg_convs.2.conv	256	256	3	3
head.cls_preds.0	1	64	1	1
name: head.cls_preds.1	1	128	1	1
head.cls_preds.2	1	256	1	1
head.reg_preds.0	68	64	1	1
head.reg_preds.1	68	128	1	1
head.reg_preds.2	68	256	1	1
head.reg_preds_lrtb.0	4	64	1	1
head.reg_preds_lrtb.1	4	128	1	1
head.reg_preds_lrtb.2	4	256	1	1
head.cls_convs.0.conv	16	16	3	3
head.cls_convs.1.conv	32	32	3	3
head.cls_convs.2.conv	64	64	3	3
head.reg_convs.0.conv	16	16	3	3
head.reg_convs.1.conv	32	32	3	3
head.reg_convs.2.conv	64	64	3	3
head.cls_preds.0	1	16	1	1
head.cls_preds.1	1	32	1	1
head.cls_preds.2	1	64	1	1

head.reg_preds.0	68	16	1	1
head.reg_preds.1	68	32	1	1
head.reg_preds.2	68	64	1	1
head.reg_preds_lrtb.0	4	16	1	1
head.reg_preds_lrtb.1	4	32	1	1
head.reg_preds_lrtb.2	4	64	1	1

Tabla C.2: Variante B del modelo estudiante: 3 861 757 parámetros

Nombre	**Entradas**	**Tamaño kernel**		
backbone.stem.rbr_reparam	16	3	3	3
backbone.ERBlock_2.0.rbr_reparam	32	16	3	3
backbone.ERBlock_2.1.conv1.rbr_reparam	32	32	3	3
backbone.ERBlock_2.1.block.0.rbr_reparam	32	32	3	3
backbone.ERBlock_3.0.rbr_reparam	64	32	3	3
backbone.ERBlock_3.1.conv1.rbr_reparam	64	64	3	3
backbone.ERBlock_3.1.block.0.rbr_reparam	64	64	3	3
backbone.ERBlock_3.1.block.1.rbr_reparam	64	64	3	3
backbone.ERBlock_3.1.block.2.rbr_reparam	64	64	3	3
backbone.ERBlock_4.0.rbr_reparam	128	64	3	3
backbone.ERBlock_4.1.conv1.rbr_reparam	128	128	3	3
backbone.ERBlock_4.1.block.0.rbr_reparam	128	128	3	3
backbone.ERBlock_4.1.block.1.rbr_reparam	128	128	3	3
backbone.ERBlock_4.1.block.2.rbr_reparam	128	128	3	3
backbone.ERBlock_4.1.block.3.rbr_reparam	128	128	3	3
backbone.ERBlock_4.1.block.4.rbr_reparam	128	128	3	3
backbone.ERBlock_5.0.rbr_reparam	256	128	3	3
backbone.ERBlock_5.1.conv1.rbr_reparam	256	256	3	3
backbone.ERBlock_5.1.block.0.rbr_reparam	256	256	3	3
backbone.ERBlock_5.2.cv1.conv	128	256	1	1
backbone.ERBlock_5.2.cv2.conv	128	256	1	1
backbone.ERBlock_5.2.cv3.conv	128	128	3	3
backbone.ERBlock_5.2.cv4.conv	128	128	1	1
backbone.ERBlock_5.2.cv5.conv	128	512	1	1
backbone.ERBlock_5.2.cv6.conv	128	128	3	3
backbone.ERBlock_5.2.cv7.conv	256	256	1	1
neck.reduce_layer0.conv	64	256	1	1
neck.Bifusion0.cv1.conv	64	128	1	1
neck.Bifusion0.cv2.conv	64	64	1	1
neck.Bifusion0.cv3.conv	64	192	1	1
neck.Bifusion0.downsample.conv	64	64	3	3

neck.Rep_p4.conv1.rbr_reparam	64	64	3	3
neck.Rep_p4.block.0.rbr_reparam	64	64	3	3
neck.Rep_p4.block.1.rbr_reparam	64	64	3	3
neck.Rep_p4.block.2.rbr_reparam	64	64	3	3
neck.reduce_layer1.conv	32	64	1	1
neck.Bifusion1.cv1.conv	32	64	1	1
neck.Bifusion1.cv2.conv	32	32	1	1
neck.Bifusion1.cv3.conv	32	96	1	1
neck.Bifusion1.downsample.conv	32	32	3	3
neck.Rep_p3.conv1.rbr_reparam	32	32	3	3
neck.Rep_p3.block.0.rbr_reparam	32	32	3	3
neck.Rep_p3.block.1.rbr_reparam	32	32	3	3
neck.Rep_p3.block.2.rbr_reparam	32	32	3	3
neck.downsample2.conv	32	32	3	3
neck.Rep_n3.conv1.rbr_reparam	64	64	3	3
neck.Rep_n3.block.0.rbr_reparam	64	64	3	3
neck.Rep_n3.block.1.rbr_reparam	64	64	3	3
neck.Rep_n3.block.2.rbr_reparam	64	64	3	3
neck.downsample1.conv	64	64	3	3
neck.Rep_n4.conv1.rbr_reparam	128	128	3	3
neck.Rep_n4.block.0.rbr_reparam	128	128	3	3
neck.Rep_n4.block.1.rbr_reparam	128	128	3	3
neck.Rep_n4.block.2.rbr_reparam	128	128	3	3
head.proj_conv	1	17	1	1
head.stems.0.conv	32	32	1	1
head.stems.1.conv	64	64	1	1
head.stems.2.conv	128	128	1	1
head.cls_convs.0.conv	32	32	3	3
head.cls_convs.1.conv	64	64	3	3
head.cls_convs.2.conv	128	128	3	3
head.reg_convs.0.conv	32	32	3	3
head.reg_convs.1.conv	64	64	3	3
head.reg_convs.2.conv	128	128	3	3
head.cls_preds.0	1	32	1	1
head.cls_preds.1	1	64	1	1
head.cls_preds.2	1	128	1	1
head.reg_preds.0	68	32	1	1
head.reg_preds.1	68	64	1	1
head.reg_preds.2	68	128	1	1
head.reg_preds_lrtb.0	4	32	1	1
head.reg_preds_lrtb.1	4	64	1	1

head.reg_preds_lrtb.2	4	128	1	1

Tabla C.3: Variante C del modelo estudiante: 1 166 909 parámetros

Nombre	Entradas	Tamaño kernel		
backbone.stem.rbr_reparam	8	3	3	3
backbone.ERBlock_2.0.rbr_reparam	16	8	3	3
backbone.ERBlock_2.1.conv1.rbr_reparam	16	16	3	3
backbone.ERBlock_2.1.block.0.rbr_reparam	16	16	3	3
backbone.ERBlock_3.0.rbr_reparam	32	16	3	3
backbone.ERBlock_3.1.conv1.rbr_reparam	32	32	3	3
backbone.ERBlock_3.1.block.0.rbr_reparam	32	32	3	3
backbone.ERBlock_3.1.block.1.rbr_reparam	32	32	3	3
backbone.ERBlock_3.1.block.2.rbr_reparam	32	32	3	3
backbone.ERBlock_4.0.rbr_reparam	64	32	3	3
backbone.ERBlock_4.1.conv1.rbr_reparam	64	64	3	3
backbone.ERBlock_4.1.block.0.rbr_reparam	64	64	3	3
backbone.ERBlock_4.1.block.1.rbr_reparam	64	64	3	3
backbone.ERBlock_4.1.block.2.rbr_reparam	64	64	3	3
backbone.ERBlock_4.1.block.3.rbr_reparam	64	64	3	3
backbone.ERBlock_4.1.block.4.rbr_reparam	64	64	3	3
backbone.ERBlock_5.0.rbr_reparam	128	64	3	3
backbone.ERBlock_5.1.conv1.rbr_reparam	128	128	3	3
backbone.ERBlock_5.1.block.0.rbr_reparam	128	128	3	3
backbone.ERBlock_5.2.cv1.conv	64	128	1	1
backbone.ERBlock_5.2.cv2.conv	64	128	1	1
backbone.ERBlock_5.2.cv3.conv	64	64	3	3
backbone.ERBlock_5.2.cv4.conv	64	64	1	1
backbone.ERBlock_5.2.cv5.conv	64	256	1	1
backbone.ERBlock_5.2.cv6.conv	64	64	3	3
backbone.ERBlock_5.2.cv7.conv	128	128	1	1
neck.reduce_layer0.conv	32	128	1	1
neck.Bifusion0.cv1.conv	32	64	1	1
neck.Bifusion0.cv2.conv	32	32	1	1
neck.Bifusion0.cv3.conv	32	96	1	1
neck.Bifusion0.downsample.conv	32	32	3	3
neck.Rep_p4.conv1.rbr_reparam	32	32	3	3
neck.Rep_p4.block.0.rbr_reparam	32	32	3	3
neck.Rep_p4.block.1.rbr_reparam	32	32	3	3
neck.Rep_p4.block.2.rbr_reparam	32	32	3	3
neck.reduce_layer1.conv	16	32	1	1

neck.Bifusion1.cv1.conv	16	32	1	1
neck.Bifusion1.cv2.conv	16	16	1	1
neck.Bifusion1.cv3.conv	16	48	1	1
neck.Bifusion1.downsample.conv	16	16	3	3
neck.Rep_p3.conv1.rbr_reparam	16	16	3	3
neck.Rep_p3.block.0.rbr_reparam	16	16	3	3
neck.Rep_p3.block.1.rbr_reparam	16	16	3	3
neck.Rep_p3.block.2.rbr_reparam	16	16	3	3
neck.downsample2.conv	16	16	3	3
neck.Rep_n3.conv1.rbr_reparam	32	32	3	3
neck.Rep_n3.block.0.rbr_reparam	32	32	3	3
neck.Rep_n3.block.1.rbr_reparam	32	32	3	3
neck.Rep_n3.block.2.rbr_reparam	32	32	3	3
neck.downsample1.conv	32	32	3	3
neck.Rep_n4.conv1.rbr_reparam	64	64	3	3
neck.Rep_n4.block.0.rbr_reparam	64	64	3	3
neck.Rep_n4.block.1.rbr_reparam	64	64	3	3
neck.Rep_n4.block.2.rbr_reparam	64	64	3	3
head.proj_conv	1	17	1	1
head.stems.0.conv	16	16	1	1
head.stems.1.conv	32	32	1	1
head.stems.2.conv	64	64	1	1
head.cls_convs.0.conv	16	16	3	3
head.cls_convs.1.conv	32	32	3	3
head.cls_convs.2.conv	64	64	3	3
head.reg_convs.0.conv	16	16	3	3
head.reg_convs.1.conv	32	32	3	3
head.reg_convs.2.conv	64	64	3	3
head.cls_preds.0	1	16	1	1
head.cls_preds.1	1	32	1	1
head.cls_preds.2	1	64	1	1
head.reg_preds.0	68	16	1	1
head.reg_preds.1	68	32	1	1
head.reg_preds.2	68	64	1	1
head.reg_preds_lrtb.0	4	16	1	1
head.reg_preds_lrtb.1	4	32	1	1
head.reg_preds_lrtb.2	4	64	1	1

Apéndice D

Estadísticas de los modelos resultantes de las pruebas exhaustivas

Tabla D.1: Consumo de memoria y tiempo de inferencia de los modelos resultantes de las pruebas exhaustivas durante el primer día de estudio

N°	GPU run			CPU run		
	t inferencia (s)	Memoria GPU (GB)	RAM (GB)	t inferencia (s)	Memoria GPU (GB)	RAM (GB)
1	7,283186	0,584371	3,504601	3,570478	0,276431	3,487663
2	7,251962	0,610805	3,504574	3,401854	0,282989	3,504601
3	7,381173	0,658626	3,504605	3,524469	0,358837	3,504604
4	7,201849	0,471213	3,504604	3,402099	0,287335	3,504608
5	7,154571	0,567225	3,504605	3,345020	0,261665	3,487276
6	7,283750	0,604501	3,488609	3,550678	0,302999	3,489620
7	6,939703	0,577287	3,487577	3,305572	0,292145	3,488663
8	7,320503	0,607777	3,487587	3,131945	0,284996	3,487583
9	7,451849	0,584501	3,490002	3,389145	0,265135	3,487276
10	6,984516	0,569489	3,485163	3,484166	0,289849	3,451456
11	7,393445	0,646038	3,490102	3,362887	0,295091	3,490177
12	7,200060	0,620567	3,490177	3,555041	0,335902	3,490177
13	6,798769	0,603064	3,497587	3,565411	0,286161	3,504608
14	7,196493	0,599652	3,486543	3,465216	0,296157	3,486511
15	7,386060	0,546516	3,496516	3,346540	0,281665	3,483516
16	7,369144	0,567548	3,486851	3,375318	0,277073	3,486246
17	6,649544	0,585116	3,471651	3,461262	0,246456	3,488160
18	7,215444	0,594805	3,490103	3,496516	0,289847	3,488003
19	7,069546	0,642626	3,500064	3,501560	0,259459	3,504608
20	6,954954	0,545213	3,491312	3,505185	0,251695	3,488004

Tabla D.2: Consumo de memoria y tiempo de inferencia de los modelos resultantes de las pruebas exhaustivas durante el segundo día de estudio

N°	GPU run			CPU run		
	t inferencia (s)	Memoria GPU (GB)	RAM (GB)	t inferencia (s)	Memoria GPU (GB)	RAM (GB)
1	7,369350	0,640743	3,504608	3,486675	0,273563	3,486543
2	7,367232	0,650783	3,504574	3,386355	0,265466	3,489741
3	7,185912	0,662170	3,504604	3,426016	0,290446	3,504604
4	7,286060	0,616434	3,504604	3,308812	0,292387	3,504604
5	7,259123	0,583743	3,504604	3,158144	0,253563	3,487645
6	7,315444	0,647984	3,489594	3,386355	0,295394	3,489622
7	7,194754	0,617133	3,487682	3,438926	0,275123	3,487096
8	7,404046	0,596775	3,487587	3,294094	0,306532	3,487583
9	7,126754	0,590984	3,504599	3,157124	0,291726	3,487632
10	7,465461	0,649541	3,496516	3,316516	0,275616	3,496159
11	7,399907	0,621482	3,490177	3,236146	0,296126	3,465465
12	7,265461	0,661565	3,505681	3,393490	0,280295	3,490177
13	7,065164	0,567225	3,488609	3,344616	0,297898	3,488004
14	6,994584	0,657944	3,504605	3,306461	0,289873	3,483121
15	6,965516	0,596126	3,485163	3,158144	0,287987	3,483110
16	7,369393	0,636345	3,486524	3,204071	0,292703	3,486309
17	7,200060	0,626456	3,483516	3,196541	0,276546	3,479994
18	7,065165	0,616547	3,487682	3,256166	0,289465	3,480010
19	7,165164	0,604695	3,551456	3,312340	0,256457	3,486466
20	7,164569	0,586543	3,490002	3,494910	0,276946	3,479994

Tabla D.3: Consumo de memoria y tiempo de inferencia de los modelos resultantes de las pruebas exhaustivas durante el tercer día de estudio

N°	GPU run			CPU run		
	t inferencia (s)	Memoria GPU (GB)	RAM (GB)	t inferencia (s)	Memoria GPU (GB)	RAM (GB)
1	7,348723	0,618409	3,504604	3,361503	0,295394	3,480010
2	6,862101	0,604255	3,504600	3,308812	0,312030	3,496159
3	6,651612	0,581447	3,504604	3,545278	0,267946	3,504604
4	6,665456	0,575133	3,504605	3,406413	0,301006	3,504611
5	7,206946	0,580893	3,504605	3,351503	0,253563	3,487648
6	7,307138	0,635634	3,489620	3,268923	0,297089	3,489624
7	6,654651	0,640211	3,487679	3,345020	0,296130	3,486466
8	7,206131	0,627287	3,488364	3,496076	0,302873	3,487587
9	6,956161	0,599654	3,488609	3,426016	0,290394	3,489620
10	7,359520	0,586516	3,485163	3,393490	0,306532	3,487096
11	7,016516	0,601919	3,490103	3,157124	0,275616	3,487632
12	7,296283	0,619849	3,504601	3,345020	0,292387	3,451456
13	6,862161	0,629645	3,465465	3,550678	0,280295	3,486309
14	7,096493	0,606156	3,504605	3,158144	0,292145	0,000000
15	6,984590	0,615616	3,488609	3,406414	0,324654	3,490177
16	6,779296	0,635330	3,486508	3,142476	0,331176	3,486312
17	7,363549	0,573621	3,489622	3,308812	0,295091	3,504604
18	7,654654	0,570540	3,490002	3,426016	0,287335	3,486543
19	7,370742	0,600614	3,486543	3,285615	0,330099	3,486456
20	6,979460	0,594556	3,476156	3,393490	0,316161	3,504601

Tabla D.4: Consumo de memoria promedio y tiempo de inferencia promedio de los modelos resultantes de las pruebas exhaustivas

N°		GPU run			CPU run		
		t inferencia (s)	Memoria GPU (GB)	RAM (GB)	t inferencia (s)	Memoria GPU (GB)	RAM (GB)
1	Valor	7,333753	0,614508	3,504604	3,472885	0,281796	3,484739
	Error	0,036734	0,023179	0,000003	0,085869	0,009686	0,003375
2	Valor	7,326238	0,621948	3,504583	3,365674	0,286828	3,496834
	Error	0,052615	0,020564	0,000012	0,040702	0,019203	0,006085
3	Valor	7,287789	0,634081	3,504604	3,498588	0,305743	3,504604
	Error	0,079941	0,037246	0,000000	0,052014	0,038651	0,000000
4	Valor	7,283819	0,554260	3,504604	3,372441	0,293576	3,504608
	Error	0,066033	0,061096	0,000000	0,045027	0,005644	0,000003
5	Valor	7,170062	0,577287	3,504605	3,284889	0,256264	3,487523
	Error	0,067291	0,007209	0,000000	0,089661	0,003819	0,000175
6	Valor	7,302111	0,629373	3,489274	3,401985	0,298494	3,489622
	Error	0,013418	0,018296	0,000471	0,115556	0,003260	0,000002
7	Valor	7,168400	0,611544	3,487646	3,363173	0,287799	3,487408
	Error	0,176955	0,025991	0,000049	0,055934	0,009110	0,000924
8	Valor	7,261919	0,606269	3,493518	3,262055	0,280477	3,487605
	Error	0,046734	0,017809	0,007845	0,165823	0,020383	0,000028
9	Valor	7,178255	0,591713	3,494403	3,324095	0,282418	3,488176
	Error	0,205614	0,006208	0,007232	0,119022	0,012233	0,001031
10	Valor	7,269832	0,601849	3,488947	3,398057	0,290666	3,478237
	Error	0,206333	0,034433	0,005352	0,068519	0,012635	0,019295
11	Valor	7,269956	0,623146	3,490127	3,252052	0,288944	3,481091
	Error	0,179229	0,018050	0,000035	0,084752	0,009434	0,011098
12	Valor	7,253935	0,633994	3,500153	3,431184	0,302861	3,477270
	Error	0,040119	0,019498	0,007068	0,089788	0,023879	0,018253
13	Valor	6,908698	0,599978	3,483887	3,486902	0,288118	3,492974
	Error	0,113625	0,025576	0,013532	0,100791	0,007318	0,008256
14	Valor	7,095857	0,621251	3,498584	3,309940	0,292725	3,484816
	Error	0,082430	0,026082	0,008515	0,125386	0,002598	0,001695
15	Valor	7,112055	0,586086	3,490096	3,303699	0,298102	3,485601
	Error	0,193907	0,029090	0,004753	0,105786	0,018952	0,003240
16	Valor	7,172611	0,613074	3,486628	3,240622	0,300317	3,486289
	Error	0,278116	0,032195	0,000158	0,098508	0,022734	0,000030
17	Valor	7,071051	0,595064	3,481596	3,322205	0,272698	3,490919
	Error	0,305432	0,022688	0,007461	0,108486	0,020041	0,010235
18	Valor	7,311754	0,593964	3,489262	3,392899	0,288882	3,484852
	Error	0,250108	0,018792	0,001118	0,100878	0,001105	0,003475
19	Valor	7,201817	0,615978	3,512688	3,366505	0,282005	3,492510
	Error	0,125665	0,018916	0,027964	0,096120	0,034030	0,008555
20	Valor	7,032994	0,575437	3,485823	3,464528	0,281601	3,490866
	Error	0,093574	0,021621	0,006857	0,050407	0,026523	0,010248

Apéndice E

Estadísticas del modelo maestro

Tabla E.1: Consumo de memoria y tiempo de inferencia del modelo maestro

N		Cantidad parámetros	GPU run			CPU run		
			t inferencia (s)	Memoria GPU (GB)	RAM (GB)	t inferencia (s)	Memoria GPU (GB)	RAM (GB)
1	1,1	18 530 545	7,564536	0,676719	3,496508	3,911378	0,303887	3,488003
	1,2	18 530 545	7,614192	0,707363	3,486622	3,736967	0,320103	3,488004
	1,3	18 530 545	7,568984	0,711363	3,487026	3,991342	0,389427	3,488006
2	2,1	18 530 545	7,576512	0,686516	3,498498	3,945651	0,376156	3,487932
	2,2	18 530 545	7,616516	0,696157	3,488063	3,894896	0,336847	3,496465
	2,3	18 530 545	7,599516	0,706157	3,488652	3,765165	0,349849	3,488016
3	3,1	18 530 545	7,589156	0,710917	3,488001	3,861565	0,356415	3,496126
	3,2	18 530 545	7,600652	0,686156	3,499159	3,909847	0,336846	3,488631
	3,3	18 530 545	7,610981	0,699846	3,479550	3,913213	0,364654	3,496156
4	4,1	18 530 545	7,596516	0,710919	3,488655	3,926156	0,379164	3,488006
	4,2	18 530 545	7,586312	0,706156	3,489490	3,894098	0,369646	3,487965
	4,3	18 530 545	7,609616	0,696156	3,488865	3,701651	0,326847	3,487895
5	5,1	18 530 545	7,616985	0,706352	3,495616	3,865136	0,336156	3,487916
	5,2	18 530 545	7,591956	0,689652	3,488020	3,893651	0,379846	3,488100
	5,3	18 530 545	7,569486	0,710619	3,479995	3,961651	0,346546	3,488615
6	6,1	18 530 545	7,586513	0,696516	3,484654	3,764565	0,349846	3,499515
	6,2	18 530 545	7,601652	0,668652	3,487912	3,798765	0,386516	3,488965
	6,3	18 530 545	7,616156	0,706156	3,466126	3,809165	0,356457	3,479612

Apéndice F

Consumo de memoria RAM física del modelo maestro y el modelo estudiante

Tabla F.1: Consumo de memoria RAM física

N		Maestro		Estudiante	
		CPU run	**GPU run**	**CPU run**	**GPU run**
1	1,1	0,262640	0,259195	0,113757	0,135460
	1,2	0,263122	0,262771	0,112601	0,125165
	1,3	0,265109	0,242283	0,114243	0,112776
2	2,1	0,246516	0,256516	0,126547	0,136517
	2,2	0,254665	0,262652	0,116540	0,116517
	2,3	0,240317	0,276516	0,126517	0,113546
3	3,1	0,269847	0,256160	0,138797	0,126546
	3,2	0,266400	0,246546	0,131652	0,136546
	3,3	0,249652	0,250561	0,126155	0,111351
4	4,1	0,246667	0,246565	0,116516	0,116155
	4,2	0,236516	0,269495	0,123127	0,126520
	4,3	0,265417	0,240317	0,131652	0,111952
5	5,1	0,258495	0,264950	0,126157	0,136157
	5,2	0,249652	0,256565	0,139849	0,126516
	5,3	0,250916	0,262652	0,119550	0,131652
6	6,1	0,260593	0,276156	0,119519	0,126216
	6,2	0,264651	0,246566	0,130955	0,116517
	6,3	0,259592	0,266515	0,126459	0,131617