UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

TESIS

UN ENFOQUE BASADO EN TÉCNICAS DEEP LEARNING PARA EL RECONOCIMIENTO DE MATRÍCULAS EN TIEMPO REAL

PARA OBTENER EL TÍTULO PROFESIONAL DE:

LICENCIADO EN CIENCIA DE LA COMPUTACIÓN

ELABORADA POR:

CRISTIAN FERNANDO AGUIRRE JANAMPA

0000-0003-2142-6396

ASESOR:

Msc. YURI NUÑEZ MEDRANO

0000-0002-4175-3637

LIMA - PERÚ

2025

Citar/How to cite	Aguirre Janampa [1]
Referencia/Reference	[1] C. Aguirre Janampa, *"Un enfoque basado en técnicas Deep Learning para el reconocimiento de matrículas en tiempo real"* [Tesis de pregrado]. Lima (Perú): Universidad Nacional de Ingeniería, 2025.
Estilo/Style:	IEEE

Citar/How to cite	(Aguirre, 2025)
Referencia/Reference	Aguirre, C. (2025), *Un enfoque basado en técnicas Deep Learning para el reconocimiento de matrículas en tiempo real.* [Tesis de pregrado, Universidad Nacional de Ingeniería]. Repositorio Institucional UNI.
Estilo/Style:	APA (7ma ed.)

Agradecimientos

Agradezco a mi familia por la paciencia y la confianza depositada en mí en la elaboración de este trabajo de investigación.

A mi asesor por el continuo apoyo, sus consejos y consideraciones que aportaron a este proyecto.

A todos los profesores de mi querida universidad que siempre tuvieron ese espíritu de enseñanza hacia los alumnos y que sin su apoyo este trabajo no estaría completo.

Resumen

La Inteligencia Artificial (IA) ha sido ampliamente estudiada en áreas como la medicina, economía e ingeniería. Una de sus aplicaciones más relevantes es el Reconocimiento Automático de Matrículas (Automatic License Plate Recognition, ALPR), utilizado en la seguridad pública y el control vehicular. Este sistema propuesto automatiza la verificación de matrículas, permitiendo identificar automóviles no registrados o robados, lo cual mejora la eficiencia en la gestión del tránsito en tiempo real. En Perú, su implementación sería útil para reforzar la seguridad vial y apoyar a las autoridades en la vigilancia vehicular.

El presente trabajo tiene como objetivo desarrollar un sistema basado en Deep Learning para el reconocimiento de matrículas peruanas en tiempo real. Se emplearán los modelos YOLO para la detección de matrículas y Lenet-5 para el reconocimiento de números y letras. La integración de ambos modelos busca optimizar la precisión y eficiencia en distintos escenarios, tanto con vehículos inmóviles como en movimiento.

Los resultados obtenidos, basados en pruebas realizadas en la puerta N° 5 de la Universidad Nacional de Ingeniería (UNI), han mostrado una precisión del 62.82 % utilizando el modelo Yolov4-tiny combinado con Lenet-5. El sistema propuesto sirve como base para futuras investigaciones y tiene un gran potencial para ser implementado en estaciones de peaje, estacionamientos y para la identificación de vehículos robados en Perú, proporcionando un mecanismo automatizado y eficaz para mejorar la seguridad vial y el control vehicular.

Palabras clave - Deep Learning, Yolo, Lenet-5

Abstract

Artificial Intelligence (AI) has been widely studied in areas such as medicine, economics, and engineering. One of its most relevant applications is Automatic License Plate Recognition (ALPR), used in public safety and vehicle control. This system automates the verification of license plates, allowing the identification of unregistered or stolen vehicles, which improves the efficiency of real-time traffic management. In Peru, its implementation would be useful to enhance road safety and support authorities in vehicle surveillance.

The aim of this work is to develop a Deep Learning-based system for the real-time recognition of Peruvian license plates. YOLO models will be used for license plate detection and Lenet-5 for the recognition of numbers and letters. The integration of both models aims to optimize accuracy and efficiency in various scenarios, including stationary and moving vehicles.

Preliminary results, based on tests conducted at Gate No. 5 of the National University of Engineering (UNI), showed an accuracy of 62.82 % using the Yolov4-tiny model combined with Lenet-5. The proposed system serves as a foundation for future research and has great potential to be implemented in toll stations, parking lots, and for the identification of stolen vehicles in Peru, providing an automated and effective mechanism to enhance road safety and vehicle control.

Keywords - Deep Learning, Yolo, Lenet-5

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Índice de Acrónimos

ALPR	Automatic License Plate Recognition
NN	Neural Network
FFNN	Feed Forward Neural Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
VRAM	Video Random Access Memory
API	Application Programming Interface
FPS	Frames Per Second
UNI	Universidad Nacional de Ingeniería

I. Introducción

En esta parte se presenta el inicio del trabajo: la motivación por la que se decidió desarrollarlo, los objetivos que se espera cumplir y la estructura de la tesis. La Sección A nos abre el panorama de cómo la Inteligencia Artificial puede solucionar el problema planteado del reconocimiento de las matrículas en el Perú y, dadas las investigaciones pasadas, da una idea o sugerencia que es un método factible en la solución del problema. La Sección B muestra los objetivos de acuerdo a las competencias académicas del trabajo. Finalmente, la Sección C muestra cómo está estructurado el trabajo, dividido en capítulos.

A Motivación

La Inteligencia Artificial y la subárea de la Visión Computacional han aportado numerosas tecnologías que resuelven problemas reales de la sociedad desde hace varios años, tanto a nivel internacional [8], [9] como nacional [10], [11], por mencionar algunos. Así, un ejemplo de este tipo de casos es el problema del Reconocimiento de Matrículas, más conocido en el mundo como *Automatic License Plate Recognition* (ALPR). Para contextualizar, existen diversos estudios acerca de esta tecnología a nivel global [3], [12],[13], [14], [15], cada uno detallando sus necesidades en el país de residencia. Sin embargo, es importante señalar que en el Perú existen pocas investigaciones. En este sentido, una de las contribuciones relevantes es la del ingeniero Ramírez [16], la cual se centra en el ALPR usando técnicas de Deep Learning en los campos de detección y reconocimiento de placas de automóviles.

El impacto social y práctico de implementar un sistema de reconocimiento automático de matrículas en Perú es significativo. En el ámbito práctico, la automatización del monitoreo del tráfico a través de estos sistemas permitiría un control más eficiente y preciso de las normativas. De hecho, el uso de ALPR en este panorama podría no solo optimizar el monitoreo del tráfico, sino también facilitar la identificación de infractores en tiempo real. Esto, a su vez, contribuiría a una mejor gestión de las normativas vigentes. Sin embargo, es importante señalar que actualmente el control vehicular en Lima sigue dependiendo en gran medida de la intervención manual de las autoridades. Esta dependencia limita la efectividad y la cobertura de estas regulaciones. Desde una perspectiva social, el ALPR puede mejorar la seguridad vial al facilitar la identificación rápida de infractores y vehículos con antecedentes penales o deudas por multas. Esto contribuye, en consecuencia, a un entorno más seguro en las carreteras. En conjunto, estos factores no solo modernizan la infraestructura de control vehicular, sino que también generan un beneficio tangible para los ciudadanos, mejorando su movilidad y seguridad.

De igual forma, la creciente congestión vehicular es un problema común, especialmente en avenidas principales. A medida que se profundiza en esta problemática, un caso en particular es la llamada ley de orden de tráfico *Plan Pico y Placa* [17] que opera en algunos sectores y avenidas principales de la capital, diseñada para regular el tráfico en horarios específicos. Esta reglamentación, al centrarse en la organización del tránsito, abre la puerta a la aplicación de tecnologías como ALPR, que podrían automatizar la identificación de vehículos permitidos en determinadas vías y horarios. De este modo, se contribuye a un control más eficiente del tráfico, lo que resulta fundamental para aliviar la congestión en las horas pico.

Además existe otro proyecto vehicular para camiones el cual se restauró el 16 de Septiembre del 2020 en el Perú que multa a los camiones que incumplan la regla de transitar por otro carril que no les corresponda [18]. El plan es el siguiente: los camiones que terminan su placa en número impar pueden circular en todos los carriles los días Lunes, Miércoles y Viernes de 6:30 am a 10:00 am y los que terminan en par los días

Martes, Jueves y Sábado en el mismo horario. Sin embargo, en el carril izquierdo pueden circular exclusivamente los camiones de Lunes a Sábado de 6:00 am a 9:00 pm sin restricciones. Esta medida vehicular representa un ejemplo de un caso de uso real al cual se podría aplicar un ALPR para la detección de las placas admitidas. No obstante, es relevante mencionar que, actualmente, se encuentra suspendido debido al inicio de la pandemia del COVID-19. A pesar de que las medidas del gobierno para controlar la pandemia ya terminaron, el *Plan Pico y Placa* sigue sin reanudarse [17].

En este contexto, dentro del Deep Learning existen diversos algoritmos que resuelven numerosos problemas de acuerdo a las necesidades de cada investigación o aplicación. De este modo, el presente trabajo pretende entender el funcionamiento de estos algoritmos, lo cual es fundamental para poder aprovechar al máximo sus capacidades. A través de la exploración de sus mecanismos subyacentes, observaremos cómo estas técnicas basadas en Inteligencia Artificial proporcionan resultados muy adecuados. Así, se pone de manifiesto su relevancia para dar soporte a varios proyectos que nuestra sociedad pretende resolver, abriendo nuevas posibilidades en diversos campos de aplicación.

Así, puede establecer que surgen grandes interrogantes de cuáles algoritmos Deep Learning son los más óptimos para este trabajo. Como trabajos previos, cabe mencionar que la detección de objetos ha sido de relevancia recientemente por parte de la comunidad IA. Así mismo, es así como surgen modelos tales como YOLOv3 [6], Yolov4 [7] y Fast-R-CNN [19] entre otros muchos. Estos modelos han mejorado su exactitud en el ámbito de la detección de objetos, lo que a su vez ha llevado a una mayor necesidad de explorar y comparar su rendimiento en diferentes aplicaciones. Por otro lado, la clasificación de imágenes no se ha quedado atrás y existen grandes aportaciones como son los modelos VGG16 [20], Resnet50 [21], MobileNet [22], LeNet-5 [23], entre otros, que son capaces de clasificar hasta 1000 tipos de objetos en imágenes. Es importante destacar que, aunque ambos enfoques, detección y clasificación, son fundamentales dentro del campo del Deep Learning, la diversidad de modelos disponibles hoy en día ofrece variadas soluciones a nuestro problema.

Integrando ambas cuestiones, entonces se pretende, en el presente trabajo, encontrar el mejor sistema integrado de reconocimiento de las matrículas de los autos y clasificador de vehículos, particularmente autos particulares y camiones en el Perú, utilizando técnicas y modelos de Deep Learning optimizados y con una alta precisión. Sin embargo, es importante destacar que el sistema diseñado tiene la capacidad de reconocer matrículas de otros países. No obstante, esta capacidad es limitada. Específicamente, no se incluye el reconocimiento en países donde se emplean letras diferentes a las utilizadas en el Perú, así como aquellos que presentan signos particulares o exóticos. Por lo tanto, la efectividad o precisión en estas matrículas extranjeras no es la misma que en el Perú. Esto se debe a que el formato de las letras presenta variaciones significativas, lo que resulta en una disminución de la efectividad.

B Objetivos

El objetivo principal de este trabajo es diseñar e implementar un sistema basado en técnicas de Deep Learning que permita reconocer matrículas de autos peruanos en imágenes y videos en tiempo real, evaluando su desempeño y viabilidad en escenarios reales.

Es así que se tienen los objetivos específicos:

- **Analizar la precisión de Yolov3 y Yolov4**, detectores de objetos, para el reconocimiento de los vehículos y las matrículas en autos de Perú.
    - *Indicador de éxito:* Comparar el rendimiento de ambos modelos, tanto en su versión original y tiny(pequeña) en términos de exactitud, para así seleccionar el modelo óptimo para el sistema ALPR. Donde se considera un mínimo de $90\,\%$ como viable.
- **Comparar diversos modelos de redes neuronales** (como Alexnet y Lenet-5) en la clasificación de caracteres (dígitos del 0-9 y letras mayúsculas del alfabeto), mediante la métrica de la precisión para cada caracter de la placa.

    - *Indicador de éxito:* Evaluar y comparar el rendimiento de diferentes arquitecturas CNN en la clasificación de caracteres (presentes en las placas peruanas) e identificar el mejor de ellos. Se considera un mínimo de $90\,\%$ como viable.

- **Evaluar la efectividad del sistema en situaciones reales**, como su implementación en estacionamientos, talleres mecánicos o ingresos a establecimientos, midiendo su precisión promedio en cada caso.
    - *Indicador de éxito:* Implementar el sistema y realizar pruebas en escenarios no controlados y establecer una línea base de rendimiento para sistemas similares en el contexto peruano.
- **Analizar e interpretar los errores** obtenidos durante los experimentos, identificando los factores que afectan el desempeño del sistema, como condiciones de iluminación, calidad de las imágenes y variaciones en las matrículas, y proponiendo soluciones en el modelo y en el diseño de pruebas para reducirlos.
    - *Indicador de éxito:* Analizar la relación entre factores y dificultades del escenario de prueba y el rendimiento del sistema, proponer estrategias de mejora basado en los errores encontrados como trabajo futuro.

C Estructura de la Tesis

Para brindar al lector una noción global del contenido de esta tesis, ahora se hace un breve resumen de cada punto de la presente investigación.

- **Introducción:** Este capítulo presenta la introducción al trabajo: la motivación por la que se decidió desarrollarlo, los objetivos que se espera cumplir y la estructura del seminario. La Sección A nos abre el panorama de cómo la Inteligencia Artificial puede solucionar el problema planteado del reconocimiento de las matrículas en el Perú y dada las investigaciones pasadas dan una idea o sugerencia que es un método factible en la solución del problema. La Sección B muestra los objetivos de acuerdo

a las competencias académicas del trabajo. Finalmente, la Sección C muestra cómo está estructurado el trabajo, dividido en capítulos.

- **Estado del Arte y fundamentos teóricos:** Este capítulo muestra la teoría relacionada a las redes neuronales y también los trabajos relacionados que se han desarrollado en el área del Reconocimiento Automático de Matrículas de automóviles, tanto en el idioma local como en el idioma extranjero (inglés).
- **Recursos y tecnologías usadas:** Este capítulo describe los recursos empleados en el presente trabajo, tanto bienes tangibles (laptops, celulares) e intangibles (Sistema Operativo, lenguaje de programación, APIs, etc.) y la justificación del uso de cada uno.
- **Diseño y arquitectura del sistema:** Este capítulo detalla la estructura y diseño del sistema, así como la función y relación entre cada uno de ellas. En la Subsección 1) y Sección C se presenta la base de datos empleado en el entrenamiento y testeo de los módulos de detección de matrículas y reconocimiento de letras y números, respectivamente. La Sección D presenta el módulo de detección de las matrículas. La Sección E el procesamiento y segmentación de las matrículas para obtener sus letras y números. Finalmente la Sección F presenta el módulo de reconocimiento de letras y números.
- **Ambiente de Experimentación y Pruebas:** Este capítulo muestra el diseño de las pruebas realizadas en la Puerta N° 5 de la Universidad Nacional de Ingeniería. Se detalla los materiales utilizados, posición y ángulo de materiales y la implementación respectiva, usada para llevar a cabo correctamente todas las pruebas y que sirvan como referencia a otros trabajos relacionados.
- **Experimentación y Resultados:** Este capítulo presenta los resultados obtenidos del entrenamiento de los modelos usados como son Yolov4, Yolov3 y LeNet-5. Cada modelo se entrenó bajo distintas variaciones y fueron escogidos las mejores personalizaciones que dieron mejores desempeños. En la primera sección se

muestra los resultados del entrenamiento y predicción con el modelo Yolov3 y Yolov4. En la Sección C se visualizan los diferentes resultados de distintos procesamientos a las matrículas. Finalmente, en la última sección se comparan los resultados de distintos entrenamientos para hallar el mejor modelo de reconocimiento de letras y números.

- **Conclusiones y Trabajo Futuro:** Este capítulo detalla las conclusiones recogidas de los resultados de las experimentaciones y pruebas con el sistema propuesto. Por último, se describen las posibles modificaciones, experimentaciones y desarrollos que podrían mejorar el sistema propuesto por algún lector interesado en la tesis realizada.

II. Estado del Arte y fundamentos teóricos

Este capítulo muestra la teoría relacionada a las redes neuronales y también los trabajos relacionados que se han desarrollado en el área del Reconocimiento Automático de Matrículas de automóviles, tanto en el alfabeto local como en el alfabeto extranjero (inglés).

La primera sección desarrolla las neuronas artificiales, la red neuronal (NN) conectada y de sentido hacia adelante (*feed-forward neural network*, FFNN), así como las redes neuronales convolucionales (CNNs). En la segunda sección detallaremos las investigaciones que tienen relación con la tesis y además sirven como una referencia a tomar en cuenta en la comparación de resultados obtenidos y al momento de discutirlos.

A Fundamento teórico

Respecto al fundamento teórico, se desarrollarán los conceptos necesarios para el correcto entendimiento del presente trabajo. Se comenzará con lo básico de las redes neuronales hasta llegar a las CNNs que son la base de nuestros modelos propuestos.

1) Artificial Neural Network (ANN) - Red Neuronal Artificial

El primer indicio de las neuronas artificiales fue introducido por Warren McCulloch et al. [24]. El paper muestra cómo las neuronas biológicas se pueden representar como un modelo computacional usando lógica proposicional. Después de unos años, fueron presentadas más redes neuronales y técnicas de entrenamiento. A continuación se presentan los conceptos que son la base de las ANN.

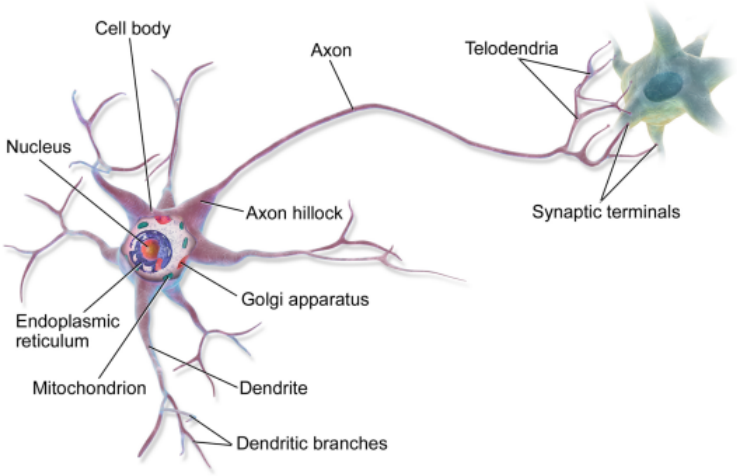


FIGURA I: Composición de una neurona biológica. Fuente: O'Really [1]

Neurona biológica

La neurona biológica (neurona) es una célula especial que solo se ubica en el cerebro y en cantidades enormes de miles de millones. Los componentes principales de la neurona son el núcleo, las dendritas y el axón. La Figura I ilustra a la neurona y sus componentes principales. El axón es un elemento importante, ya que es la vía de conexión entre las neuronas y es recibido en las dendritas de cada una de ellas. En la conexión entre el axón y la dendrita existe una estructura llamada sinapsis [25], [1] (synapses) que es quien regula las señales entre neuronas mediante los neurotransmisores. Las neuronas, al recibir las señales de otras mediante la sinapsis, pueden enviar señales a otras neuronas de acuerdo a un cierto patrón o intensidad de señales que recibieron de otras.

Neurona artificial

Siguiendo el proceso mencionado en la sub-sección anterior donde una neurona se estimula o emite un impulso a otras neuronas si recibe una cierta cantidad de señales de las demás neuronas, se tiene la neurona artificial que simula este comportamiento [1]. En la Figura II se muestra dónde una neurona se activa de acuerdo a un criterio establecido, es decir, se activa si recibe una o varias entradas (*inputs*) de otras neuronas y estas cumplen

una cierta condición (*proposición*). La primera neurona C es una *identidad* de la neurona A. La segunda neurona C es una conjunción (and) de A y B. La tercera neurona C es una disyunción (or) de A y B. Y la última neurona C es una conjunción (and) de A y B, pero con negación de B.

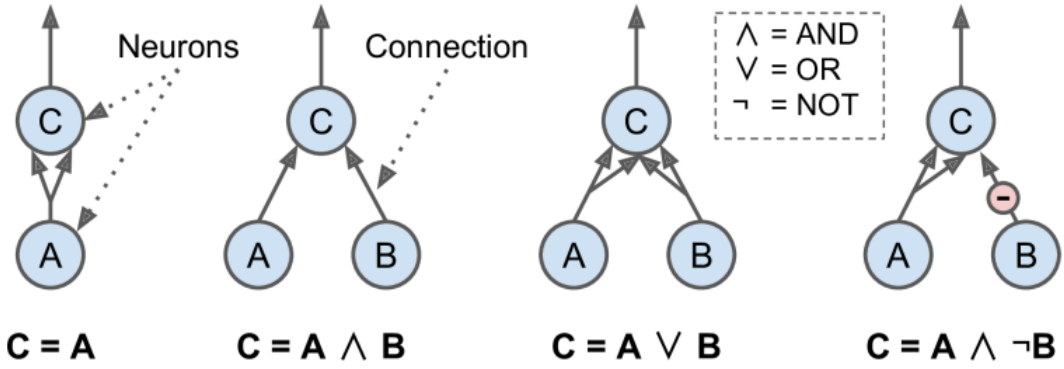


FIGURA II: Neuronas artificiales. Fuente: O'Really [1]

Perceptron

El Perceptrón es una ANN básica y nos da una idea de cómo están compuestas y desarrolladas redes más grandes. Fue creado en 1957 por Frank Rosenblatt [26]. La red neuronal tiene diferentes capas, entre las cuales está una capa de entrada (*inputs*) y una de salida (*output*) y la diferencia con la neurona artificial presentada en la Figura II es que las entradas ahora son números y tienen que ser multiplicadas por los pesos $(weights)$ de la capa siguiente. Finalmente, se suma cada producto y se aplica una función de activación (*activation function*) cuyo resultado **y** es la salida esperada $(output)$. La ecuación de este proceso está representada por:

$$\mathbf{z} = \sum_{i=1}^{n} x_i w_i = \mathbf{x}^T \mathbf{w} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$
$$\mathbf{y} = f(\mathbf{z})$$

La Figura III muestra el proceso en el Perceptron, desde las señales de entrada $(inputs)$ hasta la salida $(output)$.

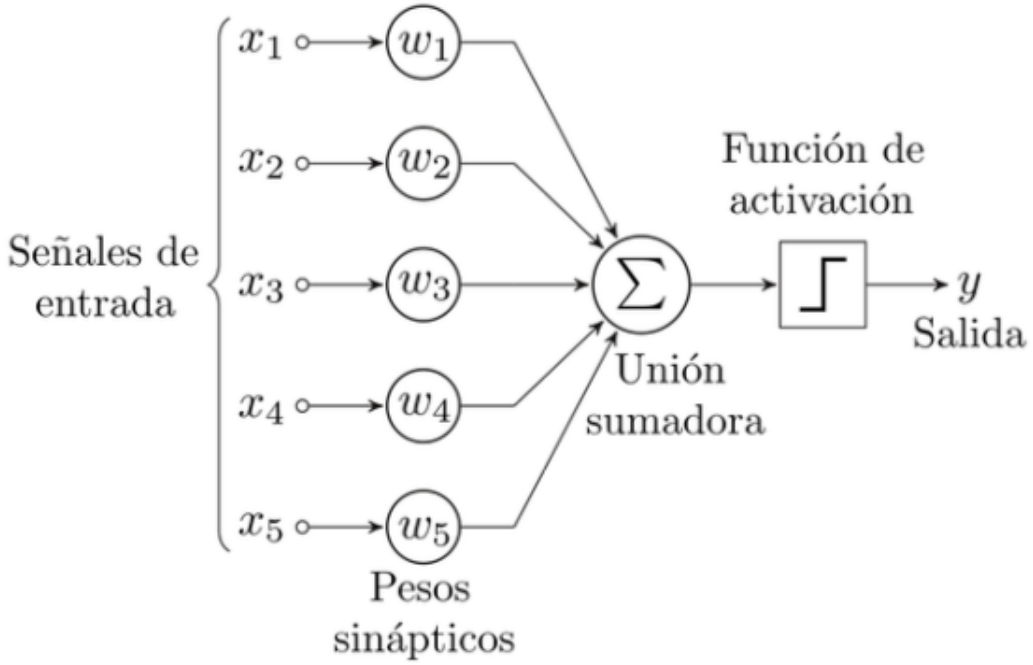


FIGURA III: Neuronas artificiales. Fuente: Blogthinkbig [2].

2) Feed-forward Neural Network (FFNN)

La Red Neuronal Prealimentada hacia Adelante es una red neuronal de varias capas con la particularidad de que no tiene ciclos o bucles en una capa [27], siempre va hacia adelante. Una red prealimentada posee una capa input (entrada), output (salida) y capas intermedias u ocultas, que es el Perceptron, la red hacia adelante más sencilla, que lo detallamos en la sección anterior. Una red neuronal más robusta y confiable es la que tiene varias capas ocultas (*hidden layers*) para poder extraer más características (*features*) de los datos de entrada. La Figura IV visualiza una red neuronal prealimentada hacia adelante con una capa intermedia.

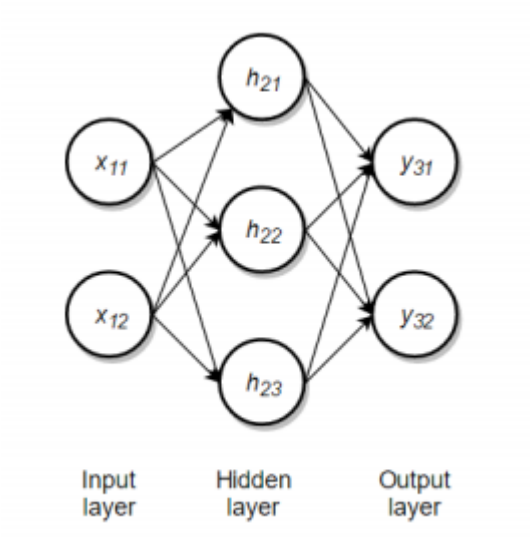


FIGURA IV: Red Neuronal Prealimentada hacia Adelante. Fuente: Hogne Jorgensen [3].

A diferencia del Perceptrón, en las ecuaciones para calcular la salida de una neurona (intermedia o salida) primero se tiene que especificar la ubicación de la neurona por medio de un subíndice al momento de multiplicar cada entrada con el peso que une a la neurona de salida [3]:

$$\mathbf{h}_{ji} = \sum_{i=1}^{n} x_{ji} w_{ji} + b_j$$

donde $j = 2, 3, 4, ..., N$, N = número total de capas de la red , n = número de neuronas en la capa j y b_j es una constante que se añade a cada capa llamada $bias$ que ayuda a diferenciar posibles salidas de **h**.

Luego se aplica la función a **h** y se tiene como resultado el output de la neurona en la capa j neurona i:

$$\mathbf{y}_{ji} = f(\mathbf{h}_{ji})$$

Función de activación

Esta función es una parte importante de la red neuronal, ya que define la salida de una neurona artificial dados los inputs de otras neuronas que se conectan a ella y alcancen

un cierto umbral [3]. En otras palabras, la neurona se activa si las señales recibidas de otras neuronas alcanzan un umbral definido. Este caso binario de activación no resulta útil en las redes neuronales actuales, que demandan funciones diferenciables para un mejor entrenamiento de la red neuronal. Todas las redes investigadas y usadas en este trabajo usan al menos 1 de estas 3 funciones de activación, las cuales son las más esenciales y punto de partida para otras funciones más avanzadas, como se verá y detallará más adelante.

Las funciones de activación más representativas son las siguientes:

- **Sigmoidea** (Sigmoid):

$$f(x) = \frac{1}{1 + e^{-x}} \tag{II.1}$$

- **ReLU** (Rectified Linear Unit):

$$f(x) = \text{máx}(0, x) \tag{II.2}$$

- **Parametric ReLU**:

$$f(x) = \text{máx}(ax, x) \tag{II.3}$$

Representadas gráficamente, se tienen las 3 funciones descritas en la Figura V

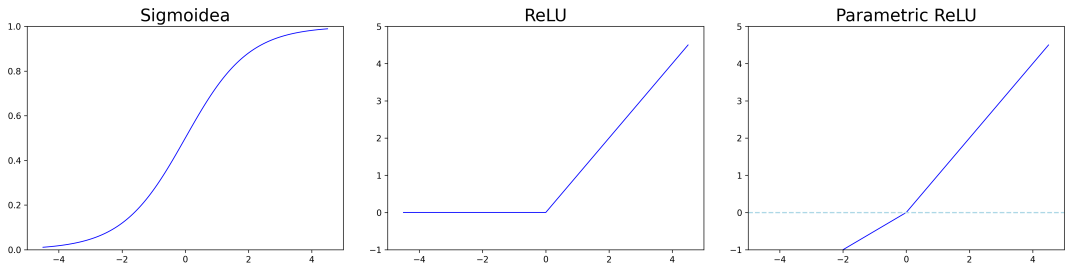


FIGURA V: Funciones de activación

Función de Costo

Dado un par (x_i, y_i) en la cual y_i es la clase real de x_i. Debería o se necesita una función que calcule qué tan bien el modelo alcanza la clase deseada para la entrada x_i, y además que pueda disminuir el error al calcularla. El modelo alcanza un valor para la

entrada x_i de acuerdo al valor de sus pesos, entonces se necesita que la función ajuste el valor de los pesos para que dé el valor correcto. Esta función es la *Función de Costo* (Loss Function, en inglés) la cual es diferente para cada tipo de modelo [28]. Sin embargo, hoy en día existen funciones de costo que han sido probadas para varios modelos y han dado buenos resultados. Las funciones de costo más usadas en una variedad de modelos de detección de imágenes (incluido los modelos Yolo que se explicarán más adelante) son [3]:

- **Mean Square Error (MSE)**: Usado generalmente en problemas de regresión:

$$E(W) = \frac{1}{N}\sum_{i=1}^{N}\|t_i - y_i\|^2 \tag{II.4}$$

  Donde W es la matriz de pesos, N es la cantidad de datos, t_i es la predicción real y y_i es la predicción del modelo.

- **Cross Entropy**: Empleado en problemas de clasificación de múltiples clases:

$$E(W) = \frac{1}{N}\sum_{i=1}^{N}[t_i \log y_i + (1 - t_i)\log(1 - y_i)] \tag{II.5}$$

  Donde W es la matriz de pesos, N es la cantidad de clases, t_i es la clase correcta y y_i es la clase predicha por el modelo.

Backpropagation

Antes de ver el algoritmo de Backpropagation, es necesario conocer el algoritmo de optimización *Gradient Descent*. Este algoritmo lo que hace es reducir la función de costo (*costo*), modificando los pesos de la red [3]. En cada paso de entrenamiento, un cambio ΔW se le suma al actual valor de W hasta reducir lo más que se pueda el *costo*. Ahora, hay un pequeño problema al reducirlo, que es el de caer en un mínimo local y no en el mínimo global, que es lo deseado; el mejor caso posible es cuando el *costo* es 0. Pero este caso es lo ideal, que en la práctica no se refleja en lo absoluto. El algoritmo no se detiene cuando el *costo* es 0, sino que lo hace cuando alcanza cierto umbral o un cierto número de iteraciones.

El algoritmo de Backpropagation se usa junto con el algoritmo de *Gradient Descent* para redes neuronales con 1 o más capas ocultas, ya que su función es actualizar los pesos de acuerdo a la función de costo obtenida en cada iteración. Lo que hace el algoritmo es *ir hacia atrás* [29] llevando consigo el error obtenido y actualizando los pesos en toda la red. La Ecuación II.6 da una forma de representar el algoritmo de *Gradient Descent*. Y la Ecuación II.7 muestra la ecuación para calcular el valor actualizado del peso con el algoritmo de backpropagation.

$$\Delta E \approx \Delta W^T \nabla E(W) \tag{II.6}$$

Donde ΔE es la variación del costo, ΔW^T es la matriz de pesos y $\nabla E(W)$ es el error obtenido con los pesos actuales.

$$w_{ij}* = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} \tag{II.7}$$

Donde $w_{ij}*$ es el nuevo valor de un peso en la posición ij, w_{ij} es el valor actual del peso ij, η es una constante llamada *learning rate* que se define para medir el tamaño del paso en cada actualización de los pesos y $\frac{\partial E}{\partial w_{ij}}$ es la derivada parcial del costo respecto al peso ij.

BatchNormalization y Dropout

Ambos son métodos regularizadores en una red neuronal. Su función es reducir el *overfitting* generado en el entrenamiento por los datos. Overfitting (sobreajuste) hace que el modelo solo reconozca los datos que ha entrenado, pero tiene un error considerable ante datos nuevos. Ambos regularizadores tienen su particularidad que se detalla a continuación:

- **BatchNormalization**: Propuesto por Sergey Ioffe y Christian Szegedy [30]. Es un regularizador que soluciona el problema de *internal covariate shift*. Este problema viene por el hecho de que al actualizar los pesos, la salida de cierta capa también afecta a la siguiente capa y así sucesivamente. Entonces se debe aplicar un método

que regule o mantenga la tendencia de reducción del costo ante posibles datos extraños o que difieran en gran medida a los demás. Esto lo hace BatchNormalization, normaliza las entradas para una capa de la red, ajustando su media y varianza.

- **Dropout**: Es utilizado para minimizar el *overfitting* en una red neuronal. Propuesta por Geoffrey Hinton [31] en 2012, se obtiene mejoras sorprendentes sobre redes neuronales incrementado su accuracy de $1\,\%$ a $2\,\%$. Parece poco incremento, mas en redes con $95\,\%$ de efectividad resulta una tarea difícil el incrementar dicho accuracy en unos cuantos decimales.

  La idea de *Dropout* es eliminar neuronas en una iteración con una probabilidad de p (llamada *dropout rate* que generalmente $p = 50\,\%$). Una vez que pasa la iteración la neurona regresa a su posición. La Figura VI representa la diferencia entre una red sin dropout y con dropout.

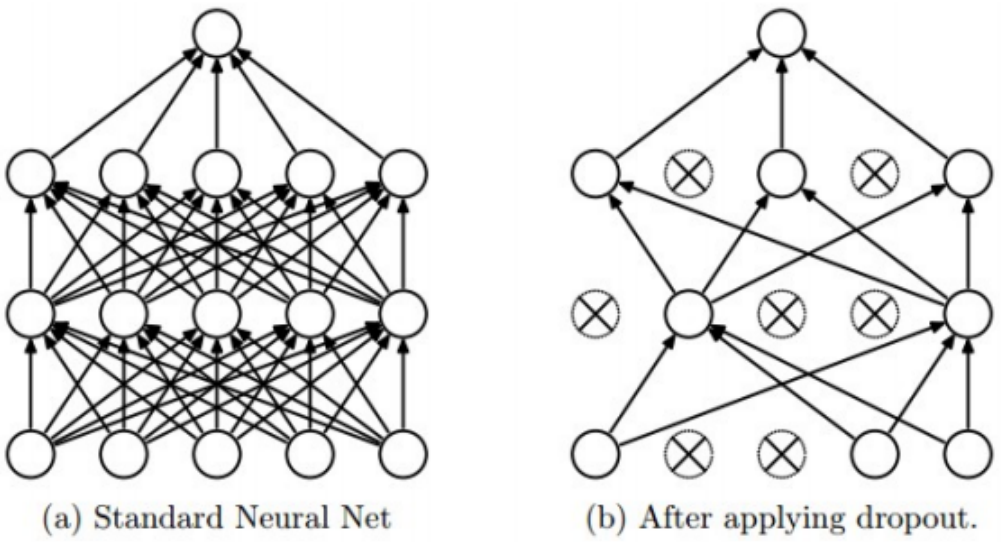


FIGURA VI: Dropout. Fuente: Jørgensen, Hogne [3].

3) Red Neuronal Convolucional - Convolutional Neural Network (CNN)

Estas redes neuronales tienen su origen en los estudios sobre la estructura de la corteza visual en los gatos, realizados por David H. Hubel et al. [32]. A partir de estas

investigaciones, se descubrió que en dicha corteza visual existen pequeñas regiones llamadas campos receptivos locales (*local receptive fields*, en inglés). Estos campos receptivos reaccionan a estímulos visuales de una región del campo visual, lo que permite entender cómo se procesan las señales visuales en el cerebro. Años después, Yann LeCun et al. [23] introdujo por primera vez la CNN para el reconocimiento de números escritos, utilizando este principio de respuesta a estímulos locales en sus modelos.

Capa Convolucional

Es el componente esencial de una CNN. En dicha capa, en vez de estar presentes las neuronas convencionales, están los filtros que no se conectan con todas las entradas, sino con los campos receptivos. La Figura VII muestra cómo un kernel opera sobre un campo receptivo de la capa de entrada (*Input layer*). Un kernel aplica una transformación lineal a dicho campo receptivo y se van obteniendo los elementos de la matriz resultante. La Figura VIII refleja la operación que realiza cada filtro ($kernel$) a una región (*matrix*) para obtener la salida.

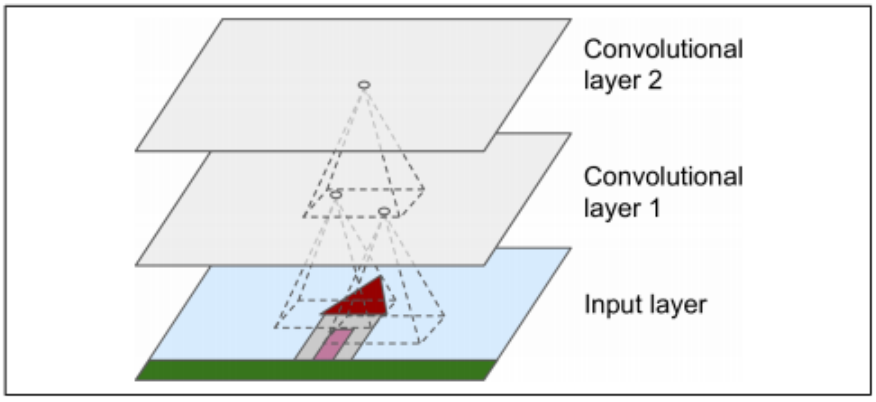


FIGURA VII: Campos selectivos. Fuente: O'Really [1]

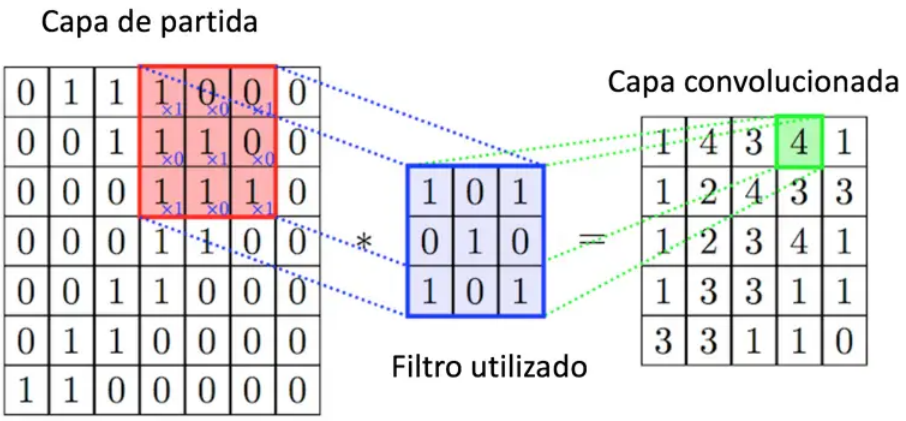


FIGURA VIII: Aplicación del filtro a una región del Input Layer. Fuente: Diegocalvo [4].

Capa Pooling

En una arquitectura de una CNN, que se detallará en la siguiente sub-sección, es común agregar capas Pooling después de una o varias capas convolucionales. La idea de una Capa Pooling es reducir el tamaño de los *feature map* (este término se refiere a las matrices resultantes después de aplicar los filtros a la capa anterior). Esta reducción de tamaño se basa en una extracción estadística para no perder información, generalmente se usa *MaxPooling* [33] (capa que toma el valor máximo de cada región de la matriz) donde se escogen los valores máximos de subconjuntos y el *AveragePooling* [3] (capa que toma el valor promedio de cada región de la matriz) donde se obtienen los promedios de subconjuntos. La Figura IX visualiza la operación de la capa Pooling. El término *stride* es el tamaño del paso que el filtro avanza sobre la matriz principal. Un *stride* de 2 significa que avanzará cada 2 filas o 2 columnas.

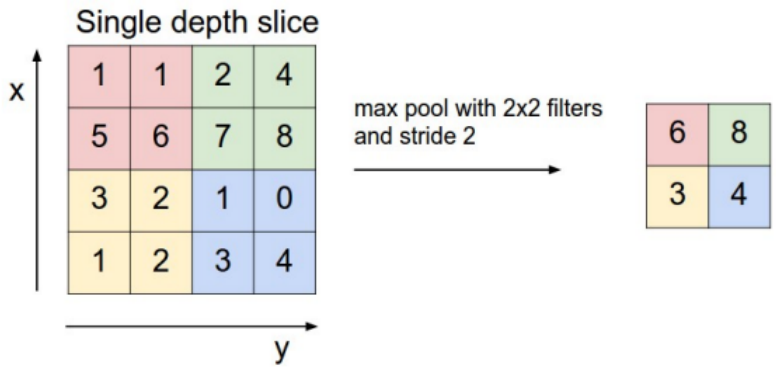


FIGURA IX: Capa Pooling. MaxPooling. Fuente: Jørgensen, Hogne [3]

Arquitectura de una Red Neuronal Convolucional (CNN)

Esta arquitectura CNN tiene 3 grandes aspectos importantes: *campos receptivos* que se usan para obtener información de una región del campo visual o una imagen como entrada (*input*), los pesos se comparten a través del filtro, los *kernels* del filtro , que no son más que matrices, operan sobre cada submatriz de la entrada y el Pooling, la cual disminuye los parámetros sin perder información y minimizando el trabajo de procesamiento de la máquina. La Figura X visualiza la arquitectura AlexNet, la cual consta de 5 capas convolucionales, 3 capas MaxPooling, 2 capas Fully Connected y 1 capa de softmax como output. Las capas Fully Connected (FC, por su siglas en inglés) son del tipo *feed-forward* donde cada una de las neuronas de una capa está enlazada con la capa anterior. La capa softmax, que es la capa output (*salida*) es una capa FC con la particularidad de que tiene tantas neuronas como clases diferentes tiene la data de entrenamiento y en cada neurona tiene un resultado entre 0 y 1 que es la probabilidad de que dicha clase sea la correcta.

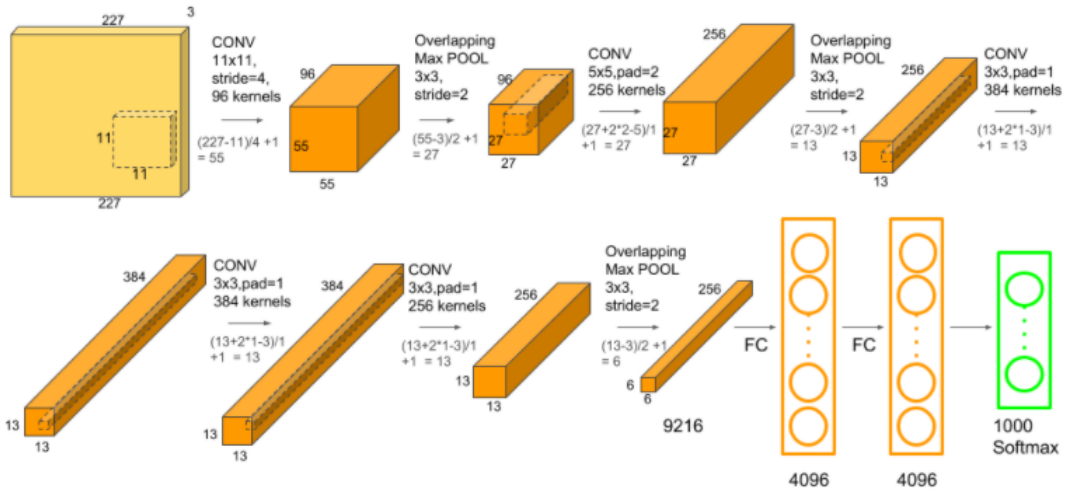


FIGURA X: Arquitectura del modelo AlexNet. Fuente: Satya Mallick [5].

4) Transfer Learning - Transferencia de aprendizaje

Es una herramienta poderosa del Deep Learning propuesta por Stevo Bozinovski [34] que se basa en aplicar el conocimiento adquirido en un entrenamiento anterior con un dataset general (comúnmente uno grande) a una tarea nueva o particular. Esta técnica es muy usada para aplicaciones de clasificación de imágenes debido a que existen varios modelos preentrenados en la actualidad que ya han sido entrenados con datasets de grandes cantidades de datos; por ejemplo, están los datasets *ImageNet, COCO dataset y OpenImages*.

En la actualidad, no se podría seguir mejorando o superando los modelos de Deep Learning sin el *Transfer Learning* dado que esta técnica ahorra tiempo para iniciar desde un punto de partida ya avanzado (entrenado). Sin embargo, debe haber una relación entre los pesos preentrenados y el modelo a usar, es decir, el dominio debe ser el mismo. Usualmente se usa el mismo modelo en el que fue entrenado para usarlo luego con los pesos, pero con un dataset nuevo y específico. Además de validar que los pesos preentrenados extraen las características o información que sirva para nuestro nuevo entrenamiento. Por ejemplo, no es posible usar pesos preentrenados de

un modelo *Faster R-CNN* [35] para entrenar un nuevo dataset para el modelo Yolov4 [7]. Además, si analizamos contextos diferentes, no es razonable usar pesos preentrenados de clasificación de imágenes para un nuevo entrenamiento de detección de objetos.

En todos los modelos investigados, usados y personalizados en este trabajo se hace uso del *Transfer Learning* para reentrenar un modelo en base a un modelo general previo, es decir, inicializando los pesos de nuestro nuevo modelo con valores ya entrenados previamente sobre un dataset mucho más grande.

Para redes neuronales desarrolladas desde cero, los pesos se inicializan de manera aleatoria en un pequeño rango. En cambio, el *Transfer Learning* iniciará los pesos de acuerdo a los valores obtenidos del modelo preentrenado; de esta manera se produce la transferencia de aprendizaje.

5) Data Augmentation

Uno de los factores que aumenta la efectividad de un modelo es la cantidad de datos, mejor aún si estos datos abarcan todo un contexto del cual se quiere enseñar al modelo. Sin embargo, no siempre se tiene tal cantidad de datos necesarios, pero la técnica del *data augmentation*, inicialmente conocida como *distorsión elástica* [36], puede resolver el problema. Esta técnica consiste en crear más datos a partir de uno, esto se logra añadiendo distintas perturbaciones o variaciones al dato para obtener muchos datos más. Para modelos en general de clasificación de imágenes se pueden realizar variaciones en una imagen como bajándole la resolución, invirtiéndola, girándola unos cuantos grados, oscureciéndola, entre otras variaciones posibles.

6) Arquitecturas de Redes Neuronales Convolucionales para la Detección de Objetos

Las nuevas arquitecturas para la detección de objetos fueron iniciadas por el año 2014 [37], las cuales dieron un nuevo enfoque para solventar y superar las arquitecturas que hasta ese momento estaban presentes. Es así que, a partir de ese año en adelante,

se han ido formando nuevas arquitecturas, cada vez superiores a sus antecesores, que consiguen mejores resultados en concursos globales en la detección de objetos [38].

Una clasificación de estos tipos de arquitecturas es la siguiente: *Two-Stage Object Detection* y *One-Stage Object Detection*. Se clasifica así porque en estas arquitecturas se tienen 2 procesos fijos: **1)** Obtener las mejores regiones donde puede estar el objeto y **2)** Clasificar el objeto que se encuentra dentro de la región obtenida del proceso anterior. Los de tipo *One-Stage* realizan estas 2 tareas en conjunto, lo cual les permite procesar la imagen de forma rápida (incluso para un entorno de detección en tiempo real como la propuesta en este trabajo), mientras que la *Two-Stage* lo hace de forma separada, y por lo tanto demanda más tiempo de procesamiento.

A continuación repasaremos las arquitecturas *CNN* más representativas en el mundo de la visión computacional y la detección de objetos.

Region-based CNN

Este algoritmo propuesto por Ross Girshick et al. [37] consta de 3 módulos principales y 2 procesos de optimización:

- **Módulo 1 - Regiones Propuestas:** En este paso, el algoritmo escoge las mejores áreas o regiones que contengan un objeto. Lo hace mediante el proceso llamado *Selective Search* [39], el cual es el encargado de captar las regiones propuestas a contener un objeto, y esto mediante combinar o trasponer las imágenes en diferentes colores, texturas y tamaños para finalmente obtener unas regiones bien marcadas en las cuales podrían haber objetos.

- **Módulo 2 - Extracción de Características:** Luego de obtener las regiones propuestas (pueden ser muchas dependiendo de la imagen, aproximadamente 2000) del paso anterior, se extraen las características del objeto por medio de una red neuronal CNN, como por ejemplo el modelo *AlexNet* preentrenado o una red CNN personalizada extractor de características. Hay que realizar un preprocesamiento, que es el conjunto de técnicas aplicadas a los datos antes de ingresar a la red, si fuera necesario a las regiones

propuestas para adecuar su tamaño a lo que espera el modelo CNN como input. Como resultado se tiene un vector de características del objeto encontrado.

- **Módulo 3 - Clasificación del Objeto:** Con el vector se puede clasificar los objetos de las regiones. En el trabajo de Ross Girsjick se usa el modelo SVM (Máquinas de Vectores de Soporte) para la clasificación de clases. Las SVM son un tipo de modelo de aprendizaje supervisado que se utiliza principalmente para la clasificación. Otro aspecto que se ha pasado por alto es la falta de métodos que integren SVM con técnicas de selección de características, lo que podría mejorar la interpretabilidad y precisión del modelo.

- **"Bounding Box Regression":** Esta mejora se usa cuando el algoritmo ya tiene la región propuesta y el objeto clasificado. Entonces se toman las coordenadas del centro de la región y las longitudes del ancho y alto. Con estos datos, se entrena un modelo de regresión para así ir mejorando las regiones de cada clase, es decir, en qué lugar es más probable que se encuentre el objeto clasificado en la región propuesta.

- **"Non-Maximum Suppression"(NMS):** Finalmente, el algoritmo, ante tantas regiones propuestas con objetos clasificados dentro de cada región, tiene que escoger cuál es la más representativa y, además, corregir las regiones superpuestas que representen el mismo objeto. Para esto, usa la técnica del NMS, que elimina, por cada objeto clasificado, las regiones con bajo score y solo queda la de mayor score. Para este proceso, usa la métrica IoU (*Intersection over Union*) para determinar si existe la superposición de regiones.

Fast R-CNN

Un año después, en 2015, Ross Girshick [19] publica su trabajo con la optimización de su arquitectura anterior, lo cual llama *Fast R-CNN* porque optimiza el tiempo de entrenamiento del algoritmo a la par que mejora el mAP (*Mean Average Precision*) con un $66\,\%$ frente al $62\,\%$ de R-CNN sobre el dataset *PASCAL VOC 2012 [38]*.

Debido a las deficiencias del modelo R-CNN como la lentitud del entrenamiento, dado que por cada región de interés realiza la extracción de características, entonces repite numerosas veces la misma operación. Luego, por cada región se clasifica el objeto (usando ahora un clasificador *softmax*), realiza el ajuste del *bounding box* y aplica el *NMS*. Todo esto, el algoritmo Fast R-CNN lo realiza en un solo módulo, simplificando notoriamente el tiempo de entrenamiento y el de inferencia (etapa de testeo).

Un aspecto importante se desarrolla: La arquitectura recibe como parámetro una imagen de entrada y un conjunto de regiones de interés (*Regions of interest - ROI*) con Selective Search [39] y por cada ROI se crea un mapa de características que luego pasa a otro vector de características general (*ROI Pooling*) que comparte la información entre todos los ROI y finalmente a las capas *softmax* para clasificar el objeto y *de regresión* para predecir su bounding box. De esta manera se integra en un solo paso, en vez de los 2000 pasos del *R-CNN*, y mejora notablemente la etapa de entrenamiento.

Faster R-CNN

Unos meses después, Shaoqing Ren, et al. [35] cambian de metodología en la familia de los modelos R-CNN, particularmente en la obtención de las regiones propuestas, dado que este paso era el que más demoraba y además era independiente de la etapa de clasificación del objeto. Este trabajo desarrolla una red convolucional que puede predecir las regiones propuestas para ya no usar los métodos tradicionales como el Selective Search [39]. La red desarrollada la denominan: **Region Proposal Network (RPN)**. El cambio mostró resultados sorprendentes en el tiempo de inferencia: 140 ms por cada imagen y esto aún mejorando la exactitud del algoritmo con un 73.2 % respecto al dataset usado en [19].

Además del desarrollo del RPN, la idea es cómo acoplar este módulo con el algoritmo Fast-RCNN para que no genere cuello de botella en un lado, no sean independientes y puedan incluso compartir características y predecir en conjunto la región propuesta y el objeto predicho dentro de la región. Esto se logra compartiendo capas

convolucionales entre ambos algoritmos. Primero se entrenan independientemente, luego se integran en 2 capas convolucionales, del lado del RPN serán sus últimas capas y de Fast-RCNN sus primeras capas. Luego se entrenan en conjunto, pero se mantienen estas 2 capas, mientras que las otras capas, tanto del RPN como de Fast-RCNN se ajustan mediante el entrenamiento. Con esta metodología, entonces el RPN no genera retraso, dado que está conectado al Fast-RCNN, por ende, el tiempo de inferencia se incrementa sustancialmente.

Mask R-CNN

En el año 2017, K He et al [40] publican Mask R-CNN, el cual es una transformación de Faster R-CNN a un modelo de segmentación de imagen. Es decir, ahora este algoritmo, además de detectar objetos, puede segmentarlos y para esta operación usa los píxeles de la imagen de entrada. El nuevo modelo logra un mAP de 35.7% sobre el dataset COCO [41] superando a otros modelos similares en el campo de la segmentación de imágenes. Además, los autores demuestran que la mejora en Faster R-CNN es realizada principalmente incluyendo 2 nuevos componentes: **ROIAlign** y **Feature Pyramid Network (FPN)**.

ROIAlign es una mejora al ROIPooling que capturaba o predecía una región propuesta y redondeaba los puntos(coordenadas) encontrados para el *bounding box*, lo cual no es factible para capturar los píxeles dado que si, por ejemplo, tenemos un objeto pequeño en la imagen, la segmentación por píxel no sería precisa. Entonces, se tenía que usar una estrategia diferente que no redondee los valores y capture bien los píxeles que corresponden al objeto: *interpolación bilineal* la cual encuentra la distancia promedio de los 4 píxeles más cercanos y así encontrar el píxel deseado, sin incurrir en el redondeo.

El otro componente **FPN** es una adición al backbone network(red preentrenada inicial, extractor de características, donde se usaron el ResNet y ResNeXt) en sus últimas capas de salida: Redimensiona el resultado del backbone en diferentes tamaños y escalas

y así, con las últimas capas, extraer diferentes tipos de salidas, tanto para objetos grandes como pequeños.

B Antecedentes y trabajos previos

En esta sección se describen las investigaciones más relacionadas y vinculadas a este trabajo respecto a un sistema ALPR usando técnicas de Inteligencia Artificial y Visión Computacional en los idiomas español (prioritarios dado que es el mismo idioma que este proyecto), inglés y portugués.

Si bien el modelo Yolo ha estado actualizándose en versiones desde el momento en que se empezó a elaborar este trabajo hasta ahora, es así que está el trabajo de Juan Terven y Diana Cordova-Esparza [42] que resumen la historia de modelos que surgieron de Yolo (desde Yolov1 por Josep Redmon et al. [43] hasta su última versión Yolo-NAS por la empresa Deci [44]), se tiene poca información respecto a investigaciones o aplicaciones con ALPR aplicado a países con las últimas versiones, algunas de ellas son ; por ende, se escogieron Yolov3 por Josep Redmon y Ali Farhadi [6] y Yolov4 por Alexey Bochkovskiy et al. [7]. Esto debido también a que aún se han estado tomando como base estas versiones para elaborar otras más complejas, por ejemplo, el trabajo de Chuyi Li et al. [45] donde publica el modelo Yolov6, toma como referencia parte de la estructura del modelo de Yolov4.

Además, es importante mencionar que hay trabajos que usan versiones anteriores de Yolo para aprovechar los pocos recursos que se necesitan para su entrenamiento y predicción, tal es el caso de la investigación sobre la detección de construcciones en riesgo o lugares hostiles con el uso de minidrones elaborada por Sayani Sarkar y Nathan Johnson [46] en donde utilizan Yolov4-tiny debido a que es un modelo pequeño pero eficiente, capaz de ejecutarse sobre sistemas con pocos recursos.

En el ámbito de ALPR, las tesis elaboradas por el Ingeniero Mundaca [47] y Espinoza [48] son trabajos realizados en el Perú y tienen la misma filosofía: detectar la

placa de una imagen de entrada por medio de técnicas de procesamiento de imágenes, tales como algoritmos de detección de líneas (Hough transform) y de contornos, detección de los caracteres por medio de proyecciones horizontales y verticales para saber en dónde hacer los recortes y obtener los caracteres uno por uno. Finalmente, reconocen cada carácter por medio de un OCR (Optical Character Recognition) el cual compara cada carácter con plantillas ya predefinidas de los caracteres de los dígitos y el abecedario de letras mayúsculas.

Si bien el problema de reconocimiento de placas vehiculares (ALRP - Automatic License Plate Recognition) es global, cada país tiene su propio diseño, sus consideraciones y sus normas en relación a las placas; es por esto que en cada uno de ellos se pueden desarrollar los sistemas ALRP de acuerdo a estas consideraciones gubernamentales de cada país. Es así como el trabajo de Saghaei [13] propone un ALRP con pruebas de placas de su país Irán. A pesar de que el autor no da muchos detalles de los algoritmos usados, se interpreta que no usa técnicas de Deep Learning para la detección de las placas, sino técnicas de visión computacional como transformación a escalas de grises, normalización de los píxeles, detección de contornos, entre otros, y OCR para el reconocimiento de los caracteres. Otro trabajo similar fue hecho en Malasia por Ibrahim et al. [14] en donde desarrolla una aplicación de ubicación de matrículas vehiculares con las mismas técnicas de visión computacional para el procesamiento de las matrículas de los autos que describe el paper anterior, pero estudia otras técnicas en la fase de registro de caracteres como son: *template matching*, *neural network*, *chain code* y *hidden markov model*. De manera similar, el estudio de Utsha Saha et al. [49] desarrolla un sistema de ALPR optimizado para matrículas de Bangladesh, evaluando distintas variantes del modelo YOLOv8 para mejorar la precisión y eficiencia del reconocimiento en este contexto específico. Por otro lado, el trabajo de Tarqui Pisaya [50] propone un sistema basado en la biblioteca OpenALPR para la identificación de matrículas vehiculares en Bolivia, aprovechando herramientas como OpenCV y Tesseract OCR. Sin embargo, su enfoque presenta limitaciones en escenarios con variabilidad en los formatos de placas y condiciones ambientales adversas, lo que

abre la necesidad de explorar modelos de detección más robustos y adaptados al contexto peruano.

Las investigaciones más relevantes que están relacionadas con el presente trabajo se exponen a continuación. Éstos merecen ser analizados con detalle debido a que son la fuente de conocimiento directa que ayudó a realizar el sistema propuesto. El paper *YOLOv3: An Incremental Improvement* [6] es la base del sistema de detección de objetos que se usa para identificar las matrículas. El artículo *Gradient-Based Learning Applied to Document Recognition*[23] describe el modelo base que se usa para el reconocimiento de los caracteres y números. El Trabajo Fin de Máster *Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional* [12] desarrolla un modelo de reconocimiento de caracteres y números en las matrículas, pero no detecta las matrículas en una imagen. Por último, la tesis *Automatic License Plate Recognition using Deep Learning Techniques* [3] es el trabajo que está más relacionado, pero se basa particularmente en matrículas del país de Noruega. Adicionalmente, el artículo *EAND-LPRM: Enhanced Attention Network and Decoding for Efficient License Plate Recognition under Complex Conditions* [51] propone un modelo que optimiza el reconocimiento de caracteres mediante una red de atención mejorada y un mecanismo de decodificación eficiente, donde muestra mejoras en la precisión respecto a modelos previos.

1) YOLOv3: An Incremental Improvement

Este reporte técnico escrito por Josep Redmon y Ali Farhadi [6] da cuenta de los resultados obtenidos con el modelo YOLOv3 ante su antecesor YOLOv2 [52] y los demás modelos populares como RetinaNet, ResNet, Inception, entre otros. El resultado obtenido por YOLOv3 es de 33.0 mAP (mean Average Precision, una métrica que evalúa la precisión de un modelo en tareas de detección) con un tiempo de inferencia de 51 ms sobre el dataset COCO [41]. Las mejoras más representativas del modelo son las siguientes:

1. **Detector:** La nueva arquitectura es más grande, por ende es un poco más lento, pero más fuerte, esto porque hace las detecciones en 3 diferentes escalas, inspirado en el ya mencionado *Feature Pyramid Network*; es decir, realiza las detecciones para una imagen en 3 diferentes tamaños, esto conlleva a más capas y más tiempo. Es en este apartado que tiene como salida el output final del modelo. Por ejemplo si el modelo recibe un input de imágenes de $416x416$, la salida final será de : $[(52, 52, 3, (4+1+numClasses)), (26, 26, 3, (4+1+numClasses)), (13, 13, 3, (4+1+numClasses))]$, donde 4 por la cantidad de coordenadas del anchor box (2 para las coordenadas x, y respecto al anchor box y 2 para las coordenadas x, y del centro del anchor box respecto a la celda general), 1 del valor de la confidencia de la presencia de un objeto y numClasses la cantidad de clases entrenadas.
2. **Extractor de características:** También llamado *Darknent-53* (la Figura XI muestra su arquitectura) es comparado con otros modelos similares, demostrando su ventaja ante ellos. Este extractor mejora a su antecesor haciendo uso de bloques residuales, el cual es una técnica útil cuando se quiere agrupar varias capas y hacer la red neuronal más grande. La arquitectura de YOLOv3 posee 106 capas en total y está entre los más usados para la tarea detección de objetos tanto para objetos grandes y pequeños. En el presente trabajo se hace uso del modelo mencionado para realizar las pruebas de detección de matrículas de los automóviles.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

FIGURA XI: Arquitectura del extractor de características de YOLOv3 (*Darknet-53*). Fuente: Joseph Redmon [6]

3. Con Yolov3 se cambia el sentido de una clasificación multi-clases a multi-label. Esto es porque en la versión anterior se clasificaban los objetos y se tenía una clase exacta como resultado. Con Yolov3 se tiene en cuenta que pueden estar varios tipos de objetos en la imagen, entonces cada objeto tendrá su score. Por ejemplo, si hubieran 10 tipos de objetos entrenadas, entonces cada objeto tendrá su score para cada objeto encontrado.
4. **Anchor boxes:** Esta técnica ya usada en la versión previa de Yolo se usa principalmente para no buscar el bounding box del objeto en toda la imagen completa. En cada salida del detector se muestran resultados en 3 escalas diferentes, cada una de ellas se dividen en celdas y cada celda tiene 3 anchor boxes en diferentes escalas. Con esto, cada anchor box es un candidato a ser el bounding box de una celda y ya no de toda la imagen completa, con lo cual se facilita su ubicación en la imagen completa.

5. **Loss Function:** La función de costo para el modelo Yolov3 toma en cuenta las 4 coordenadas mencionadas (x, y, w, h), pero teniendo en cuenta si se encontró un objeto dentro del bounding box (obj). Dado la naturaleza de la fórmula de ser una regresión, se usa **MSE** en estas 2 primeras partes. Las siguientes 2 secciones hace referencia si se encontró un objeto en dichas coordenadas. Para el boundig box verdadero la *objetividad* siempre será 1, dado que la data real siempre tendrá el objeto marcado correctamente. La última parte es un problema de clasificación con n clases, se usa una función **Binary cross-entropy - (Entropía cruzada binaria)** para tener el score de cada clase.

$$
\begin{aligned}
F_{costo} =& \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left[(x_i - x_i')^2 + (y_i - y_i')^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{w_i'} \right)^2 + \left(\sqrt{h_i} - \sqrt{h_i'} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} (C_i - C_i')^2 \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{noobj}} (C_i - C_i')^2 \\
&+ \sum_{i=0}^{S^2} 1_{i}^{\text{obj}} \sum_{\text{classes}} (p_i(c) - p_i'(c))^2
\end{aligned}
\tag{II.8}
$$

2) YOLOv4: Optimal Speed and Accuracy of Object Detection

Esta nueva versión de Yolo, publicada en Abril del año 2020 por A Bochkovskiy, CY Wang y HYM Liao [7], fue diferente a las anteriores debido a que son otros autores los que realizaron este trabajo que las versiones previas, pero finalmente fue aceptada por la comunidad como la cuarta versión del modelo Yolo. En ese momento, las investigaciones sobre los modelos de detección de objetos eran variadas. Era de imaginarse entonces que se aplicarían y probarían estas nuevas y mejores técnicas y estrategias sobre los módulos

o componentes del modelo Yolov3. La figura XII muestra cómo se descompone un modelo de detección de objetos óptimo.

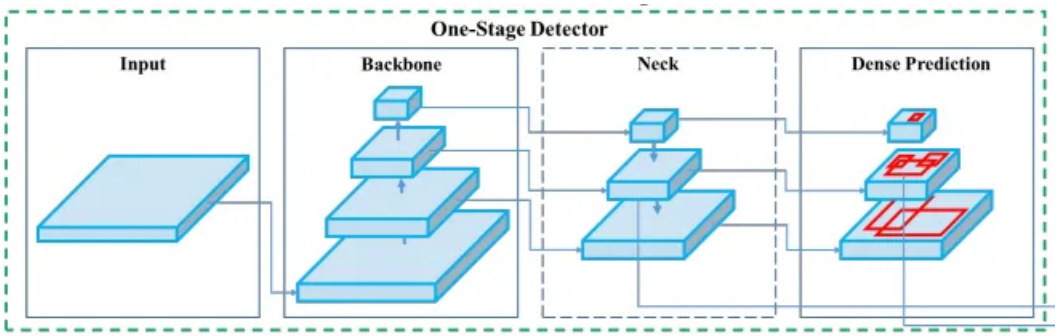


FIGURA XII: Componentes de un modelo detector de objetos. Fuente: Yolov4 [7]

Los autores obtuvieron los siguientes resultados en base a las pruebas en cada uno de los componentes:

1. Se realizaron experimentaciones con varias técnicas y nuevos algoritmos del momento para incluir a la versión Yolov3 y así mejorar tanto la precisión como el tiempo de inferencia (por consiguiente los FPS del modelo). En ese sentido agruparon varias técnicas en 2 grupos: *Bag of Freebies* y *Bag of Specials*.
2. **Bag of Freebies:** Estas mejoras están orientados a mejorar el *accuracy* sin incrementar el costo o tiempo de inferencia. Las mejoras o cambios de estrategia se aplican en la fase de entrenamiento. Entre las técnicas aplicadas estan: *CutMix* [53], *Mosaic Data Augmentation*, *DropBlock*[54], *CIoU-loss* [55], *Cross mini-Batch Normalization* y *Self-Adversarial Training*.
3. **Bag of Specials:** Este conjunto de técnicas en el post-procesamiento aumentan considerablemente el *accuracy* pero tiene un costo de inferencia pequeño extra al momento de la detección del objetos. Algunas de las ténicas y métodos más importantes son: *Mish Activation* [56], *Cross-Stage partial Connections*, *SPP-Block* [57], *SAM-Block* [58], *PAN path aggregation block* [59] y *DIoU-NMS* [55].

4. El modelo final de Yolov4 después de todas las estrategias usadas fue con las siguientes técnicas:
    - *Backbone:* CSPDarknet53 [60], BoF (CutMix, Mosaic Data Augmentation, DropBlock y Class Label smoothing) y BoS (Mish Activation y Cross-Stage partial Connections, etc.)
    - *Neck:* SPP y PAN
    - *Head:* Yolov3, BoF (CIoU-loss, DropBlock, Mosaic Data Augmentation, Self-Adversial Training, etc.) y BoS (Mish Activation, SPP-Block, SAM-Block, PAN path aggregation block y DIoU-NMS)
5. Se aseguró que el entrenamiento del modelo y las predicciones puedan ser hechas con GPU de uso comercial habitual, asequible para cualquier persona. Esto porque todas las pruebas experimentales fueron hechas con un GPU GTX 1080Ti y RTX 2080Ti.

En conclusión, el modelo YOLOv4 representa una versión mejorada de su predecesor, ya que incorpora nuevos métodos tanto en la fase de entrenamiento (*Bag of Freebies*) como en la fase de detección (*Bag of Specials*), los cuales contribuyen a incrementar el *accuracy* del modelo.

3) Gradient-Based Learning Applied to Document Recognition

Este paper elaborado por Yann LeCun et al. [23] muestra y explica las ventajas que tienen las CNNs sobre las tradicionales. Más aún, se expone una red neuronal llamada LeNet-5, la cual tiene una variedad de capas convolucionales que se usan para la tarea de reconocimiento de caracteres. Muestra a la comunidad científica este modelo que ya estaba siendo usado por empresas bancarias para leer caracteres millones de veces al mes. Compara los resultados del error obtenido de LeNet-5 con otros modelos clasificadores de la época, como un clasificador lineal, K vecinos más próximos (*k-nearest neighbors*), un clasificador polinómico, LeNet-1 (pequeña red convolucional) y con LeNet-4

(red neuronal parecida a LeNet-5) .Describe en detalle el modelo LeNet-5 que se usa como base del sistema de reconocimiento de letras y números elaborado en este trabajo.

Sin embargo, este estudio se enfoca en el reconocimiento de caracteres en documentos con condiciones controladas, sin abordar los desafíos específicos del reconocimiento de caracteres en matrículas vehiculares, donde factores como iluminación variable, ángulos de captura y ruido pueden afectar el desempeño del modelo. Además, el estudio no considera la integración de LeNet-5 con un sistema de detección de matrículas, lo que es fundamental para la implementación de un sistema ALPR completo en entornos reales.

4) Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional

Este trabajo desarrollado por Núñez Sánchez-Agustino, Francisco José [12] está relacionado con el presente trabajo en el procesamiento de las matrículas y reconocimiento de letras y números, mas no en la detección de las matrículas. Es más, el autor deja como trabajo futuro o mejora de su sistema agregar un modelo de reconocimiento de objetos. Un aspecto importante es que el sistema propuesto fue evaluado con matrículas de España (*país del autor*) y matrículas de algunos formatos estadounidenses. Los resultados mostrados fueron prometedores con las matrículas españolas con un porcentaje de aciertos del $100\,\%$, pero con la consideración de que la muestra fue de 10 imágenes y en condiciones muy favorables para su reconocimiento. Sin embargo, los resultados fueron bajos con placas estadounidenses, dando un porcentaje de $60\,\%$.

Esto evidencia la necesidad de un sistema más robusto que considere una mayor variabilidad en los datos, tanto en términos de diversidad de matrículas como en condiciones reales de captura, donde factores como iluminación, inclinación y calidad de imagen afectan el rendimiento del reconocimiento. Además, la ausencia de un módulo de detección previo limita la escalabilidad y aplicabilidad del sistema en un entorno ALPR completo. En este trabajo se abordan estas limitaciones mediante la integración de un

modelo de detección basado en YOLOv4-tiny, en conjunto con un módulo de procesamiento intermedio y una red LeNet-5 adaptada para la clasificación de caracteres en matrículas vehiculares.

5) Sistema para reconocimiento automático de placas vehiculares mediante una biblioteca OpenALPR

Este proyecto de grado desarrollado por Jessica Tarqui [50] se implementa un sistema que captura imágenes de vehículos (de Bolivia) y utiliza OpenALPR para su reconocimiento en tiempo real. Esta herramienta está escrita en C++ y utiliza OpenCV y Tesseract OCR para detectar y extraer los caracteres de las matrículas, además que es de fácil integración en sistemas web para la gestión del tráfico. La metodología empleada se basa en técnicas de segmentación, procesamiento de imágenes y reconocimiento óptico de caracteres (OCR) para identificar las placas vehiculares. Sin embargo, este enfoque depende en gran medida de la calidad de las imágenes, las condiciones de iluminación y las particularidades de las placas en diferentes regiones, lo que puede afectar la precisión del sistema. Si bien OpenALPR ha demostrado ser eficiente en distintos entornos, la variabilidad en los formatos de las matrículas y la influencia de factores externos pueden impactar su desempeño.

Los resultados no son claros, pero deja en evidencia la factibilidad (a nivel de implementación y costos) de colocar un sistema ALPR en estacionamientos vehiculares.

6) Automatic License Plate Recognition using Deep Learning Techniques

Esta tesis elaborada por Jørgensen, Hogne [3] estudia los sistemas propuestos de reconocimiento de matrículas de esa época. Estudia tanto la precisión y la rapidez de cada sistema (con rapidez se pretende tener unos altos FPS en una grabación en vivo) y los evalúa para compararlos luego con su sistema propuesto. En la segunda parte, describe su sistema: el trabajo se centra en 2 modelos YOLOv2, uno para la detección de las matrículas y el otro para la detección de las letras y números. Un aspecto importante es

que el dataset usado para entrenar y testear su modelo de detección de matrículas fueron placas de automóviles del estado de Noruega. Sus resultados fueron sorprendentes en relación a la precisión (*accuracy*) obtenida: su modelo de detección de matrículas alcanzó un $99.8\,\%$ y el modelo de detección de caracteres obtuvo un $97.8\,\%$, en ambos casos con 30 FPS. Si bien los resultados son sorprendentes, el trabajo se centró en testear los modelos independientes y no como se propone en este trabajo. Sin embargo, el autor comenta que no lo toma en cuenta porque el punto de integrar ambos modelos debería ser un trabajo aparte, dada su complejidad. Esto deja una brecha en la literatura, ya que no se analiza el impacto de la integración entre la detección y el reconocimiento de caracteres en un flujo de procesamiento unificado. Además, el uso de YOLOv2 podría no ser la opción más eficiente para dispositivos con recursos limitados, lo que motiva la exploración de modelos más ligeros como YOLOv4-tiny, empleado en esta investigación. Asimismo, el dataset utilizado se enfoca únicamente en matrículas noruegas, lo que plantea la necesidad de evaluar el desempeño de los modelos en otros formatos de matrículas, como los utilizados en Perú.

7) Deep Learning System for Automatic License Plate Detection and Recognition

El trabajo realizado por Selmi et al. [15] hace uso de 2 redes neuronales convolucionales en 2 de las 3 etapas del sistema propuesto: En la primera etapa, el sistema detecta la placa por medio de filtro Gaussian blur (un método para suavizar imágenes y reducir el ruido), binarización (convertir una imagen a solo dos tonos, generalmente blanco y negro), detección de contornos (identificación de los límites de los objetos en la imagen) y filtro geométrico para luego pasar las posibles placas detectadas a la primera red neuronal para verificar si dicho objeto es o no una verdadera placa. El autor detalla que esta red neuronal consta de 4 capas, 2 capas convolucionales para la extracción de características (proceso que permite identificar patrones relevantes en las imágenes) y 2 capas fully-connected (donde cada neurona de una capa está conectada a todas las neuronas de la siguiente capa) que terminan en un output (salida) de 2 clases,

placa y no-placa. Si la predicción de la placa tiene un score mayor o igual a 0.7 se considera que es una placa y si es menor a 0.7 será considerada no-placa. Luego, en la segmentación de caracteres no se usa una red neuronal, más bien se usan técnicas a la placa obtenida como maximizar el contraste (aumentar la diferencia entre los colores para mejorar la claridad), extracción jerárquica y filtro geométrico para poder obtener todos los caracteres de matrícula. El último paso es reconocer los caracteres obtenidos mediante una red neuronal convolucional. El autor no menciona si toma alguna referencia para la elaboración de la red, se entiende que lo desarrolló desde cero y que tiene las siguientes características: 4 capas convolucionales con función de activación ReLU (que introduce no linealidades en el modelo), donde 3 de ellas tienen su capa MaxPooling (un método de reducción de la dimensión espacial de las características), 2 capas fully-connected de 1024 y 37 neuronas y la capa de salida con 37 clases (10 clases para los dígitos, 26 clases de A-Z y una clase no-carácter).

Los resultados los toma por separado para cada módulo y no los integra como se propone en esta tesis. El módulo de detección de la placa obtiene un $93.8\,\%$ de precisión y en el módulo de clasificación de caracteres un $96.2\,\%$ de *accuracy*. Sin embargo, este enfoque presenta algunas limitaciones que esta investigación busca abordar. En primer lugar, la segmentación de caracteres basada en técnicas tradicionales puede ser sensible a variaciones en iluminación y degradación de las matrículas, lo que impacta la precisión del reconocimiento. En contraste, este trabajo emplea un modelo de Deep Learning específico para dicha tarea, lo que podría mejorar la robustez del sistema. Además, el uso de redes personalizadas en la clasificación de caracteres deja abierta la cuestión de si arquitecturas probadas como LeNet-5 pueden ofrecer un mejor desempeño en términos de generalización. Finalmente, la evaluación de Selmi et al. [15] se realizó en un entorno controlado y no se analiza el rendimiento en matrículas de diferentes países, lo que refuerza la necesidad de un estudio aplicado a formatos peruanos, como el que se desarrolla en esta tesis.

8) YOLOv8 for Bangla License Plate Recognition: Advancing Real-Time Object Detection in Localized Contexts

Este trabajo desarrollado por Utsha Saha et al. [49], investigadores de la North Dakota State University analiza el uso del modelo YOLOv8 para la detección y reconocimiento de matrículas en Bangladesh, con el objetivo de mejorar la precisión y eficiencia en tiempo real dentro de un contexto localizado. Para ello, los autores compilaron y expandieron un conjunto de datos que abarca diversas categorías de vehículos y condiciones ambientales, permitiendo evaluar la adaptabilidad del modelo. Se entrenaron cinco variantes del modelo YOLOv8 (YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l y YOLOv8x) y se analizaron utilizando métricas de desempeño como la precisión promedio (mAP), precisión, recall y puntaje F1.

Los resultados mostraron que YOLOv8x alcanzó el mejor desempeño, logrando un mAP50 de 0.96 y un mAP50-95 de 0.75, lo que indica una alta precisión en la detección de matrículas bangladesíes en comparación con versiones más ligeras del modelo (como Yolov8n, pero que son más rápidas para usar en tiempo real). Se concluye que YOLOv8 es una opción eficaz para sistemas de ALPR en entornos desafiantes, ofreciendo un equilibrio entre precisión y eficiencia computacional. Los autores sugieren que futuras investigaciones podrían ampliar el conjunto de datos e integrar el modelo YOLOv8 en sistemas de reconocimiento automático de matrículas (ALPR) para mejorar su aplicabilidad en gestión del tráfico y seguridad.

Si bien este sistema propuesto no fue probado en tiempo real, entonces no se puede realizar una comparación exacta con los resultados que esta tesis brinda, además lógicamente por la detección de placas de diferentes países.

9) EAND-LPRM: Enhanced Attention Network and Decoding for Efficient License Plate Recognition under Complex Conditions

Trabajo desarrollado por Shijuan Chen et al. [51] donde propone un modelo denominado EAND-LPRM, que busca mejorar la precisión y eficiencia del reconocimiento automático de matrículas en entornos complejos (se hace el uso imágenes con placas vehiculares de China en situaciones reales). Para ello, introduce una red de atención mejorada y un mecanismo de decodificación optimizado, con el objetivo de superar las limitaciones de los métodos tradicionales, los cuales suelen ser vulnerables a condiciones adversas como iluminación deficiente, ángulos extremos y ruido en las imágenes.

Los resultados experimentales indican que el modelo EAND-LPRM logró una precisión del 94 %, lo que representa una mejora significativa respecto a modelos previos como CRNN (81 %), ResNet18(88 %) y ResNet50(88 %). Además, su eficiencia computacional lo hace adecuado para aplicaciones en tiempo real, como vigilancia y control vehicular. Se concluye que la combinación de atención mejorada y decodificación optimizada permite un reconocimiento más robusto y rápido en escenarios desafiantes.

Sin embargo, este trabajo se enfoca exclusivamente en el reconocimiento de caracteres de la matrícula y no en su detección dentro de la imagen, tal como lo hace esta investigación, donde se abordará esta brecha mediante el uso de técnicas basadas en YOLO para optimizar la detección de matrículas en imágenes y videos en tiempo real, mejorando así el desempeño integral del sistema propuesto.

C Conclusiones

La revisión literaria abordada en este capítulo es tomada como una base de modelos que se personalizarán en este trabajo, tales como YOLOv3, YOLOv4 y LeNet-5. Además de ser punto de referencia para realizar las comparaciones pertinentes, tanto para el sistema en general como para los modelos de reconocimiento de matrículas o de caracteres que son parte del sistema integral.

Entonces, el presente trabajo se centra en aprovechar las investigaciones mencionadas para proponer un sistema que reconozca matrículas de automóviles en el Perú y contribuya a la comunidad con otro sistema, además de los ya existentes propuestos por otros autores en su país de residencia y sirva como base a otras investigaciones que deseen proponer su sistema en sus países.

III. Recursos y tecnologías empleadas

En este capítulo se detallan los recursos usados en el presente trabajo, tanto bienes tangibles (laptops, celulares) e intangibles (Sistema Operativo, lenguaje de programación, APIs, etc.) y la justificación del uso de cada uno. La Sección A detalla las herramientas físicas como las unidades de cómputo. La Sección B muestra las principales librerías y software usados para la mejor y útil forma de desarrollar el trabajo.

A Recursos

Aquí se presenta el hardware usado en esta tesis, así como el Sistema Operativo si tal recurso fuera un sistema de cómputo. Tales recursos fueron el equipo disponible hasta la fecha, ya que en un futuro se estima obtener más recursos para que más pruebas sean realizadas al sistema.

Debido a la alta demanda de procesamiento y almacenamiento de los datos en la Memoria Principal (RAM, por sus siglas del inglés) por las redes neuronales para su entrenamiento, se destaca que fue imprescindible contar con la Unidad de Procesamiento Gráfico (GPU) junto al sistema de cómputo, ya que optimiza el tiempo para el entrenamiento de la NN a una escala de 10 veces, aproximadamente, más rápido que en una Unidad de Procesamiento de Cómputo (CPU). Lo que tomaría 10 días de entrenamiento con CPU, pudo realizarse en aproximadamente 1 día con GPU.

Tomando en cuenta el contexto anterior, los recursos pueden limitar las configuraciones o parámetros de la NN. Un ejemplo práctico es el conjunto de datos a procesar en una iteración de entrenamiento; mientras se tenga una mayor Memoria Virtual

de la GPU (VRAM, por sus siglas en inglés), más conjunto de datos se puede procesar en una iteración, siendo el entrenamiento más rápido.

En este sentido, se puede tener una idea de la importancia de los recursos al entrenar, evaluar y comparar modelos de NN. Mientras se limiten estos recursos, más lento puede ser el desarrollo de un sistema o modelo de red neuronal.

A continuación son detallados los equipos computacionales utilizados para desarrollar el siguiente trabajo:

Laptop Acer Aspire 3 A315-55G-56A7

Las primeras pruebas del desarrollo de trabajo fueron realizadas con este tipo de laptop. Cuenta con una GPU de gama baja, esto lo contrarresta en que es de bajo consumo energético, ideal para portátiles. Con esta laptop fueron entrenadas y hechas las pruebas de la red neuronal de detección de letras (mayúsculas) y números que son los que vienen en la matrícula de los autos. En la Tabla I se encuentran sus especificaciones.

TABLA I: Especificaciones Laptop Acer Aspire 3 A315-55G-56A7.

Recurso	Características
Procesador	Intel Core i5-8265U CPU@1.60GHz, 8 proc lógicos
Memoria principal	12 GB
Almacenamiento	128 GB SSD y 1 TB HDD
GPU	Nvidia GeForce MX230
Sistema Operativo	Ubuntu 18.04.1 LTS x86_64

Notebook HP Omen 15-ce002la

Las últimas pruebas realizadas en la Puerta 5 de UNI para detectar las placas de los vehículos que ingresaban al campus fueron hechas con esta notebook. Posee un procesador de gama media-alta, el cual es necesario para que los FPS obtenidos por el detector no sean bajos cuando se detecta en tiempo real. Con esta notebook se logró la demo de cómo funciona el sistema en una situación real: vehículos ingresando al campus y registrando sus caracteres de placa. Opcionalmente, una persona puede estar viendo la

pantalla de la laptop para ver cómo el sistema va detectando y guardando las placas. En la Tabla II se visualizan las especificaciones técnicas de esta notebook.

TABLA II: Especificaciones Notebook HP Omen 15-ce002la.

Recurso	Características
Procesador	Intel Core i7-7700HQ 2.80GHz, 8 proc lógicos
Memoria principal	12 GB
Almacenamiento	1 TB HDD
GPU	Nvidia GeForce GTX 1050 4GB
Sistema Operativo	Ubuntu 20.04.1 LTS x86_64

Computadora de Escritorio

Para la segunda parte de las pruebas realizadas con los videos de YouTube o videos grabados desde el celular o una cámara PoE se usó una computadora de escritorio con una GPU de gama media-alta; esta ventaja del GPU facilitó la rapidez en la hora de ejecución del sistema, además de mostrar un alto FPS, lo cual se vio reflejado en los videos procesados por el sistema. Con esta computadora no se entrenaron las redes neuronales por una razón en particular: debido a que el entrenamiento dura aproximadamente 2 días o más, el coste energético estaba limitado. Por tal motivo, los entrenamientos fueron realizados por el clúster Jaguar que se detallará después. En la Tabla III se encuentran las especificaciones de la computadora de escritorio:

TABLA III: Especificaciones computadora de escritorio.

Recurso	Características
Procesador	AMD Ryzen 5 1600AF
Memoria principal	16 GB de 3000 MHz
Almacenamiento	500 GB SSD
GPU	Nvidia GeForce RTX 2060 Super
Sistema Operativo	Ubuntu 20.04.1 LTS x86_64

Clúster Jaguar

El clúster llamado Jaguar de propiedad de la UNI está compuesto por un nodo maestro y dos esclavos. Ha sido utilizado un mes y medio después del inicio. Su uso fue

de vital importancia dado que cuenta con una GPU más potente y con más memoria que la laptop descrita en la subsección anterior. Con este clúster fue entrenada la NN para la detección de las matrículas usando el modelo Yolov3, además de procesar los vídeos de prueba en nuestra red neuronal. En la Tabla IV se dan las especificaciones de uno de los nodos esclavos.

TABLA IV: Especificaciones Clúster Jaguar.

Recurso	Características
Procesador	2xProc. Intel Xeon E5-2650 2.2GHz, 30MB Cache
Memoria principal	(64GB) 8 x 8GB RDIMM, 2400MT/s
Almacenamiento	2x 300GB 15K RPM SAS 12Gbps 2.5”
GPU	Tesla K80 12GB VRAM
Sistema Operativo	Ubuntu 18.04.4 LTS x86_64

Servidor FC-UNI

El servidor de la Facultad de Ciencias de la UNI es un moderno equipo de escritorio que posee una tarjeta gráfica moderna, el cual fue obtenido específicamente para entrenar redes neuronales. El uso de este equipo con GPU fue para los entrenamientos de los modelos YOLO, dado que este proceso demora días y el servidor está encendido las 24 horas; es el escenario ideal para este requerimiento. Más aún, dado que tiene una de las últimas GPU del mercado, se usó este equipo para el método GridSearch que se verá en el Capítulo VI. La Tabla V muestra las especificaciones del servidor.

TABLA V: Especificaciones Servidor FC-UNI

Recurso	Características
Procesador	Intel Core i7-13700 2.1GHz, 30MB Cache
Memoria principal	(64GB) 2 x 32GB DDR5, 5200MHz
Almacenamiento	2TB NV2 NVMe PCIe 4.0 M.2
GPU	Gigabyte GeForce RTX4070Ti 12GB VRAM
Sistema Operativo	Windows 11 de 64 bits

Cámara PoE Amcrest

Las cámaras PoE son cámaras de seguridad que son usadas como cámaras de video vigilancia que pueden estar prendidas las 24 horas del día. Las siglas PoE (Power over Ethernet) indican que estas cámaras reciben el suministro eléctrico sobre el cable de red Ethernet que a la vez es por donde envían las grabaciones o frames que está capturando la cámara. Es decir, solo necesita un cable por donde se conecta a la red y recibe la energía para estar operando. Estas cámaras son usadas para las grabaciones que serán procesadas por el sistema y posteriormente para usarlas en tiempo real con el sistema. Actualmente se tienen 2 cámaras PoE en las cuales se usó la cámara 1(A) para las grabaciones y más adelante se usarán las cámaras 1 y 2 en diferentes sitios estratégicos. La Tabla VI y la Tabla VII contienen las especificaciones de la cámara 1 y cámara 2, respectivamente. Las cámaras fueron adquiridas con el propósito de obtener la mejor resolución posible de los vehículos y sus placas, además de que cuentan con el protocolo RTSP para la transmisión en tiempo real.

La Figura XIII muestra las cámaras mencionadas.

(A) Cámara Amcrest. Modelo: IP8M-2496EB-28MM

(B) Cámara Amcrest. Modelo: IP8M-2454EW

FIGURA XIII: Fotografías de las cámaras PoE usadas en la investigación.

TABLA VI: Especificaciones Cámara Amcrest 1.

Recurso	Características
Resolución máxima	4k UltraHD a 15 FPS (3840x2160)
Conectividad	RJ-45 (10/100Base-T)
Almacenamiento	Tarjeta microSD (hasta 128GB)
Temperatura soportada	-22°F - 140°F
Dimensiones	18.03 x 7.11 x 7.11 cm
Peso	0.54 kilogramos
Modelo	IP8M-2496EB-28MM

TABLA VII: Especificaciones Cámara Amcrest 2.

Recurso	Características
Resolución máxima	4k UltraHD a 15 FPS (3840x2160)
Conectividad	RJ-45 (10/100Base-T)
Almacenamiento	Tarjeta microSD (hasta 128GB)
Temperatura soportada	-22°F - 140°F
Dimensiones	12.20 x 8.90 cm
Peso	0.50 kilogramos
Modelo	IP8M-2454EW

Celular Samsung A30

El celular fue utilizado para realizar las grabaciones de los autos en las calles de Lima. Después de realizar las grabaciones, con el archivo `.mp4` se procede a probarlo en nuestro sistema de red neuronal para observar si predice bien o no y en qué medida lo hace. Las especificaciones se encuentran en la referencia [61].

Router Huawei WS318n

De los recursos usados para las pruebas de grabaciones y video en vivo realizadas en la puerta N°5 de la UNI fue un router Huawei. Dado su tamaño pequeño y además que cumple con las especificaciones requeridas mínimas o básicas de la cámara PoE (velocidad de transferencia 10/100 Mbps) fue escogido este router para crear la pequeña red que pueda conectar la cámara con la laptop. La referencia [62] muestra las especificaciones del router.

Inyector PoE TL-POE150S

La cámara PoE necesita un adaptador para que, por medio de un cable de red, se le otorgue energía y conectividad; entonces, este inyector PoE va a ayudar con esta dificultad. Su tamaño pequeño de 80.8*54*24 mm es útil para ser usado en cualquier ubicación; además, su velocidad de transferencia de hasta 1000 Mbps es más que suficiente para el propósito de las pruebas con la cámara PoE. Sus especificaciones se hallan en la siguiente referencia [63].

(A) Router Huawei WS318n

(B) Inyector TL-POE150S

FIGURA XIV: Herramientas usadas para la implementación del ambiente de pruebas.

B Software empleado

A continuación son detalladas las herramientas informáticas (software) utilizadas; así como el porqué de su uso para este trabajo en particular.

1) Python

Python es el lenguaje de programación más usado en el contexto del Deep Learning, debido a su fácil entendimiento y uso. Además de contar con muchas librerías elaboradas por grandes comunidades y desarrolladores en todo el mundo que son para

mejorar, resolver ciertos problemas o tareas específicas. Por tanto, este lenguaje de programación se ajusta perfectamente al desarrollo de nuestro trabajo.

2) OpenCV

La librería OpenCV está disponible para varios lenguajes de programación, incluido Python. Fue usada la versión 4 de OpenCV debido a que es la más actualizada hasta el momento y cuenta con un enorme soporte de la comunidad debido a su popularidad. Es una de las librerías más útiles porque tiene muchas herramientas para procesar y transformar imágenes y vídeos.

Para el presente trabajo fue necesario hacer un preprocesamiento de las imágenes de las matrículas de los coches para que nuestra red de detección de reconocimiento de letras y números tenga una precisión alta.

3) Tensorflow 2.4, Keras y CUDA v11

Tensorflow es un módulo o librería que puede estar integrado en Python y está diseñado para el desarrollo de modelos de ML y DL. Por su simplicidad al uso y alto número de funciones aplicadas a las NN, fue escogida esta librería para el desarrollo de nuestro sistema propuesto. Uno de los módulos integrados en Tensorflow 2.4 es Keras, el cual ofrece la construcción de modelos de NN de forma sencilla. Por tanto, para el trabajo se ha aprovechado todo el potencial de estas librerías para la creación de modelos de NN en nuestro ámbito de aplicación. Asimismo, dado que el procesamiento de aprendizaje para los modelos propuestos exige una considerable demanda de hardware, es necesario el uso de tarjetas gráficas y librerías que la usen para nuestro propósito. En este sentido, la librería **CUDA** nos sirve para que el proceso sea más rápido, aprovechando la capacidad de la tarjeta gráfica. El entrenamiento de un modelo que nos demoraría días, sería cuestión de horas con una tarjeta gráfica Nvidia y la librería CUDA.

Si bien es cierto que existen versiones actualizadas de Tensorflow hasta la versión 2.13, las limitaciones en nuestra tarjeta gráfica (por ser de modelos antiguos) del clúster

Jaguar y la computadora de escritorio nos hacen elegir la versión 2.4, además de que en esta versión ha sido probada y validada su compatibilidad con los modelos del sistema propuesto.

4) Anaconda

Anaconda es un framework gestor de paquetes para APIs muy utilizado en la computación científica debido a que cuenta en sus repositorios con una amplia gama de librerías para tal fin. Cuenta entre sus librerías un Cuaderno (*Notebook*) que sirve para observar gráficamente los datos utilizados u obtenidos después de un análisis de datos. Así como también se ha utilizado para importar las librerías necesarias como TensorFlow, Keras, NumPy, etc. Finalmente, Anaconda fue utilizado también para la construcción de la NN de reconocimiento de números y letras.

5) Otras librerías y recursos

Además de las librerías anteriores, se han utilizado otras librerías del ámbito científico menores, pero no menos importantes. Estas son:

- **NumPy**: Esta librería fue usada para realizar operaciones algebraicas en los datos, como redimensionar matrices, cambiar el tipo de dato en un vector, etc.
- **Matplotlib**: Matplotlib fue usada para mostrar gráficos de datos, además para mostrar gráficamente los resultados obtenidos del entrenamiento de nuestra NN de reconocimiento de letras y números.
- **Tensorboard**: Esta es una herramienta que acompaña a Tensorflow. Es otra opción a Matplotlib ya que también muestra los resultados de un entrenamiento, pero proporcionando más detalles sobre estos.

IV. Diseño y arquitectura del sistema

En el presente capítulo se detalla la estructura y diseño del sistema, así como la función y relación entre cada uno de ellos. En la Subsección 1) y Sección C se presenta la data usada para el entrenamiento y testeo de los módulos de detección de matrículas y reconocimiento de letras y números, respectivamente. La sección Sección D presenta el módulo de detección de las matrículas. La Sección E el procesamiento y segmentación de las matrículas para obtener sus letras y números. Finalmente, la Sección F presenta el módulo de reconocimiento de letras y números.

A Sistema propuesto de detección de matrículas del Perú

El sistema propuesto e integrador de los modelos mencionados anteriormente (*Yolov3-Tiny o Yolov4-Tiny y LeNet-5*) y, además, del procesamiento necesario de las placas está representado en la Figura XV la cual resume el proceso de reconocimiento de las matrículas.

NOTA: Es importante señalar que la detección de la placa final (para el caso de detección por video o en tiempo real) se basa en la moda de una serie de detecciones en un corto periodo de tiempo (milisegundos). Es decir, en un escenario en tiempo real, cuando el auto avanza, en cada instante el sistema va prediciendo su placa, y en un lapso determinado se halla la moda (la placa más detectada) que sería la placa final predicha.

Cada proceso será detallado en las siguientes secciones para su mejor entendimiento.

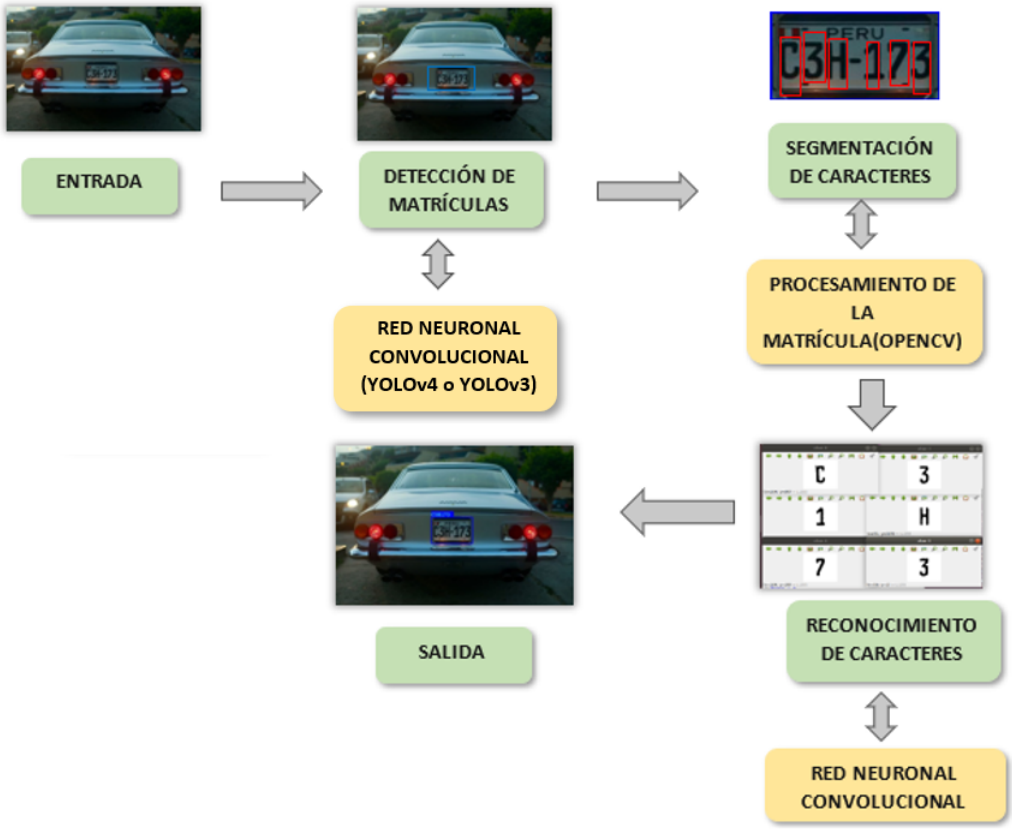


FIGURA XV: Proceso del sistema propuesto.

B Descripción de los datos de detección de matrículas

1) Primera iteración

Para un buen sistema de detección de matrículas, es necesario tener un buen número de datos, en este caso imágenes con matrículas, y dado que la característica principal de las matrículas es igual en la mayoría de países, es decir, una placa metálica rectangular de tamaño aproximado de 30 cm de ancho y 15 cm de alto (caso en el Perú), entonces se usó una fuente con matrículas de distintos países. La fuente para recolectar estas imágenes es del dataset *OpenImages* ubicadas en su página web 1.

La página web en mención tiene una gran utilidad debido a que sus datos ya están etiquetados de acuerdo a su uso en una red neuronal, es decir, hay imágenes

para entrenamiento, validación y testeo. Exactamente posee $9,011,219$ imágenes de entrenamiento, $125,436$ de validación y $41,620$ de testeo.

Además, todas las imágenes están divididas en 600 clases, es decir, existe una gran variedad de éstas, entre las cuales se encuentran las matrículas de automóviles que están con el nombre de *Vehicle registration plate*.

Se agruparon los datos de la siguiente manera:

- **Imágenes de entrenamiento:** 5368
- **Imágenes de validación:** 1113

Una herramienta útil creada por Angelo Vittorio [64] fue usada para recolectar todas las imágenes y además crear un archivo *csv* donde se colocan las coordenadas de las matrículas dentro de la imagen. Este proceso es importante ya que así requiere el sistema YOLO tener sus entradas (*inputs*).

Para el sistema propuesto se usaron todas las imágenes disponibles que hay en dicha página web sobre matrículas de automóviles, tanto los 5368 para el entrenamiento como los 1113 para la validación.

2) Segunda iteración

Para un segundo conjunto de datos se recolectaron imágenes mediante la cámara Amcrest que estuvo ubicada en la puerta N° 5 de la UNI. La cámara, al estar prendida por varias horas, guardaba las grabaciones en una memoria SD externa. Posteriormente, con los videos, se procedió a obtener imágenes de los autos (las mejores tomas posibles). En total se recolectaron 402 imágenes, de las cuales se agruparon así:

- **Imágenes de entrenamiento:** 352
- **Imágenes de validación:** 50

Todas las imágenes tienen una dimensión de 3360 x 2100, esto gracias a la cámara Amcrest que graba en 4K. La siguiente imagen es una muestra de lo recolectado:

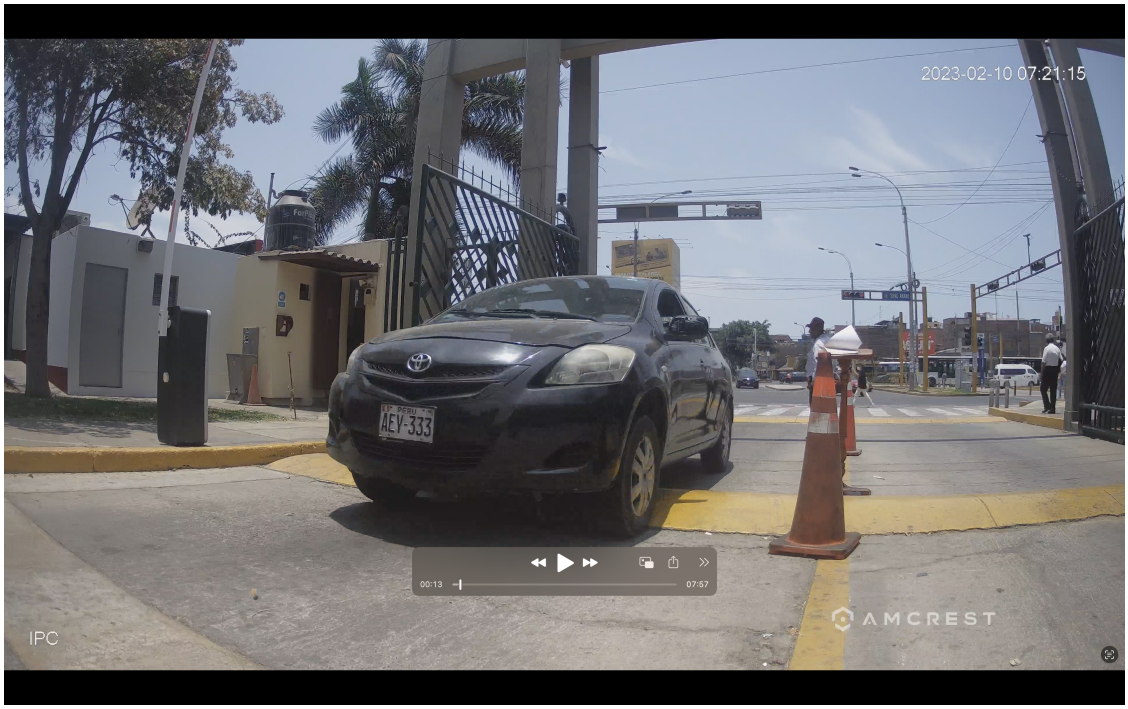


FIGURA XVI: Imagen recogida de la cámara Amcrest

Todas las imágenes fueron etiquetadas manualmente mediante el uso de programas de edición de imágenes y preparadas para ser usadas en el entrenamiento por el modelo Yolo.

C Descripción de los datos de reconocimiento de caracteres alfanuméricos

En los datos del módulo de reconocimiento de letras y números se utilizó el dataset **Chars74k**. Este dataset consta de 3 tipos de datos:

- *Creadas por ordenador*: Estas imágenes de letras y números fueron creadas por un computador, con varias fuentes y características diferentes (negrita, itálica, etc). Existen 62992 imágenes de este tipo.
- *Escritas en una tablet*: Una persona escribió con su mano las letras y números en una tablet. Hay 3410 imágenes de este tipo.

- *Del mundo real*: Cuando nos referimos al mundo real decimos que son imágenes sacadas de marcas en los productos, letreros, publicidad en las calles, etc. Se encuentran 7705 imágenes de este tipo.

Para nuestro sistema se usaron imágenes del tipo *Creadas por ordenador* y *Escritas en una tablet* debido a que éstos son de características iguales a las letras y números de las matrículas, esto hace referencia a que las matrículas son letras negras con fondo blanco y los datos de estos tipos también lo son. Sin embargo, dentro del tipo de datos *Creadas por ordenador* se encuentran imágenes exóticas como en la Figura XVII que no nos sirven para nuestro sistema. Luego de ser eliminadas manualmente estas imágenes exóticas, quedaron un total de **36787 imágenes** del tipo *Creadas por ordenador* y **1971 imágenes** del tipo *Escritas en una tablet*.

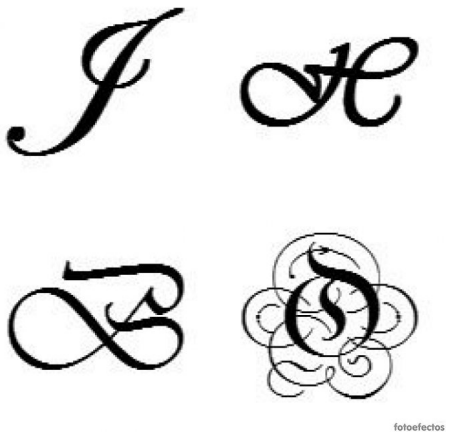


FIGURA XVII: Caracteres inservibles dentro de los datos.

D Modelo de detección de matrículas

Para el módulo de detección de las matrículas de los automóviles, los modelos Yolov3, Yolov4, Yolov3-tiny y Yolov4-tiny fueron usados. Como se menciona en el Capítulo II estos modelos son unos de varios que están popularizados en internet y que cualquiera puede hacer uso de ellos. Los modelos tienen una red neuronal profunda que posee muchas capas ocultas (*hidden layers*) y realizan predicciones multi-escala, es decir, pueden

detectar objetos grandes, medianos y pequeños. Si bien se experimenta con dichos modelos, el modelo Yolov3 y Yolov4 no están desarrollados para funcionar en sistemas embebidos o con pocos recursos; más bien, necesitan de una GPU para poder ejecutar el sistema en un tiempo aceptable. Sin embargo, los modelos Yolov3-tiny y Yolov4-tiny, los cuales se verán más adelante que tienen mejores resultados, obtienen un mejor rendimiento en sistemas embebidos y menor precisión que Yolov3, pero realizando una comparación de costo-precisión, los modelos tiny poseen mejores resultados.

1) Justificación del modelo

Existen 2 razones por las cuales se usa el modelo YOLOv3-tiny y/o YOLOv4-tiny en el sistema propuesto.

Ventajas sobre otros modelos

La razón principal es la comparación realizada de modelos de detección de objetos que en el momento de empezar este trabajo estaban presentes. En la Tabla VIII [7] se encuentran las métricas: *Mean Average Precision (mAP)* y los FPS que son las más importantes para estas arquitecturas de detección en tiempo real (sobre el dataset COCO [41]), dado que se requerirá que no operen solamente sobre GPU, sino sobre CPU que realizan un cómputo menor y que pueden estar presentes en ordenadores comunes así como en dispositivos pequeños. Viendo la relación *mAP - FPS* tenemos los modelos Yolo como la mejor elección para resolver nuestro problema del *ALPR*.

TABLA VIII: Comparación de modelos de detección de objetos

Modelo	mAP	FPS
Yolov4	43,5 %	62
Yolov3	42,4 %	45,5
RetinatNet [65]	37 %	37
Faster-RCNN [66]	39,8 %	9,4
SSD [67]	25,1 %	43
LRF [68]	37,3 %	31,3

Otra razón por la cual el modelo Yolo-tiny fue escogido es por su óptimo desempeño en sistemas embebidos o pequeños. Ofrece una precisión aceptable detectando las placas vehiculares y lo más importante es que muestra un alto FPS con respecto a otros modelos detectores de objetos, incluyendo sus versiones mayores. Este modelo es el "hermano pequeño de Yolo", dado que es similar a éste, pero tiene una red neuronal más pequeña (tanto en el backbone como en el detector) para que pueda procesarse más rápido, además de que Yolo ofrece una detección eficaz de objetos grandes, medianos y pequeños y Yolo-tiny tiene algunas limitaciones al detectar objetos pequeños, pero esto no influye en demasía en el sistema que se verá en el Capítulo V.

Adicionalmente, el proceso de detección de una imagen es un poco complejo en estos modelos, ya que introduce un nuevo término al sistema: *anchor box*. Este nuevo enfoque permite la detección en varias escalas de la imagen, lo cual lo hace la alternativa precisa para objetos en diferentes tamaños. Esta técnica de aprendizaje empieza desde el core de Yolov3-tiny: Darknet19. Si bien es más conocido Darknet53 por ser extractor de características de Yolov3, el core anterior es más ligero y lo hace ideal para sistemas embebidos. La estructura del modelo son 7 capas convolucionales, 6 capas de MaxPooling y otras capas convolucionales pequeñas para la concatenación y salida final del output, la cual sale en 2 escalas: de 13x13 y de 26x26 para objetos grandes y medianos, respectivamente. La Figura XVIII muestra la red neuronal mencionada.

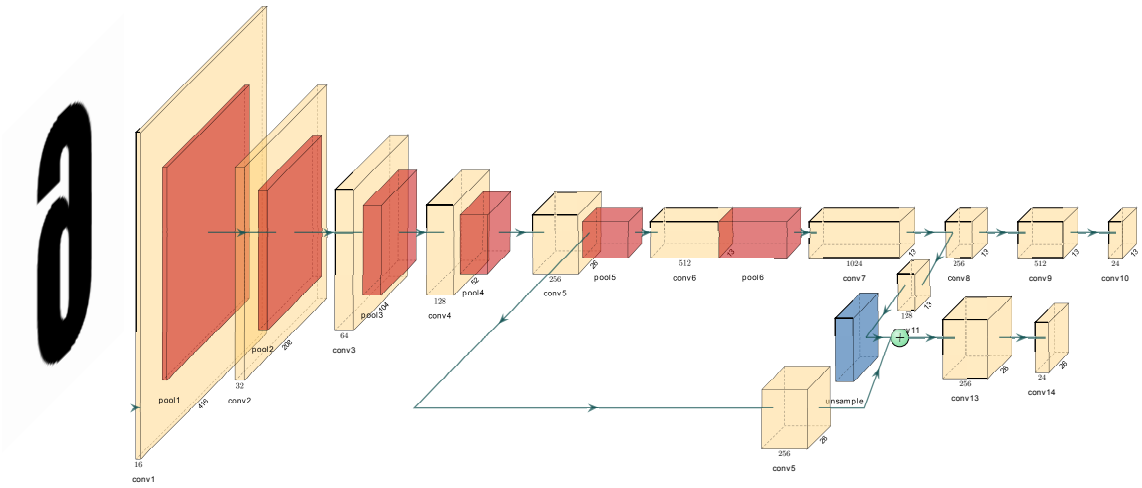
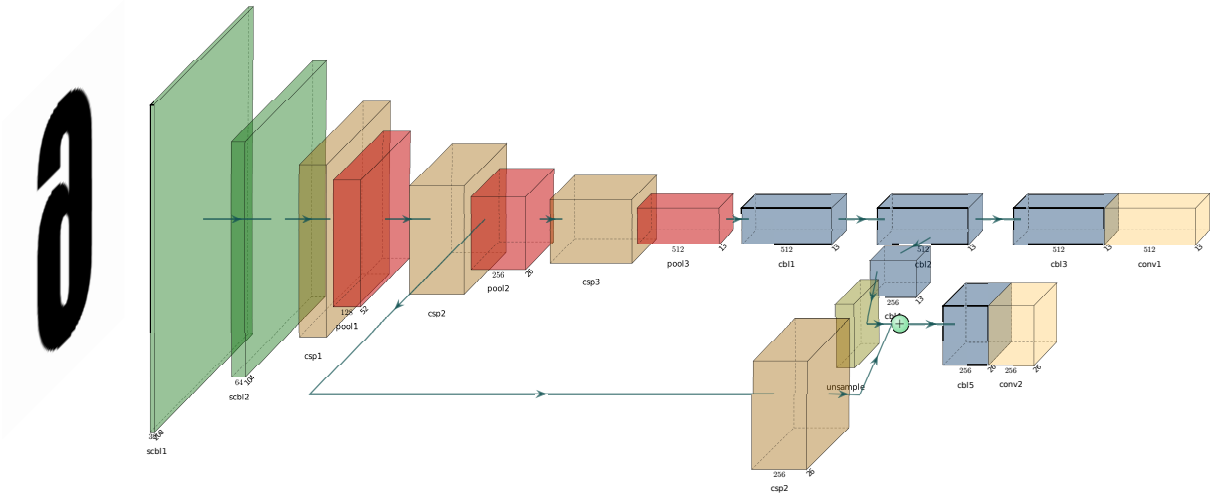
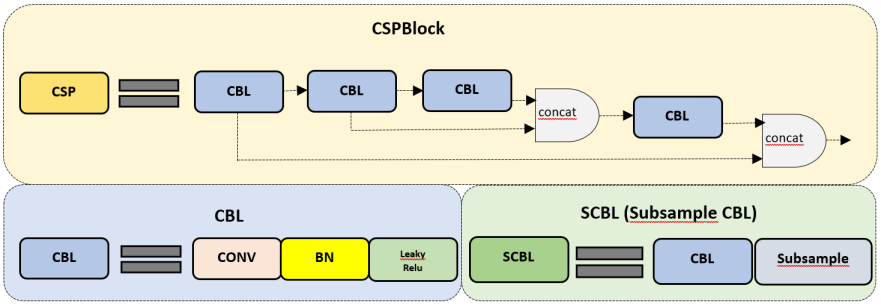


FIGURA XVIII: Red neuronal Darknet19.

En el caso de Yolov4-tiny, su arquitectura es diferente de la de Yolov3-tiny, dado que introduce el concepto de CSPB (Bloques de *Cross-Stage-Partial-Connection*), es decir, reemplaza cada capa convolucional por bloques de capas (Capa Convolucional, Batch Normalization y Leaky RelU). La similitud radica en que ambos modelos tienen como output las mismas 2 escalas: 13x13 y 26x26.



(A) CSPDarknet53-tiny



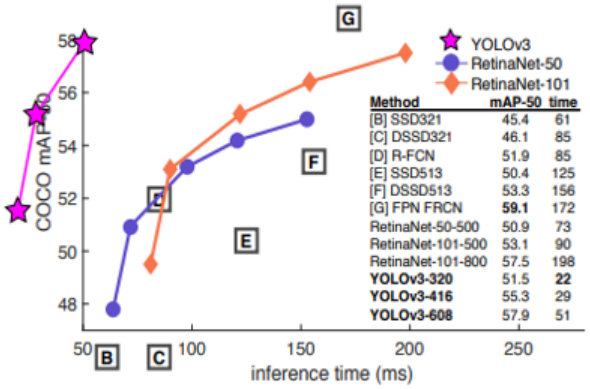
(B) Bloques CBL, SCBL y CSPBlock

FIGURA XIX: Red neuronal CSPDarknet53-tiny.

Fácil uso de implementación

Así como es uno de los mejores detectores de objetos por su rapidez, también es práctico porque se puede personalizar fácilmente para detectar objetos particulares, lo que se tiene que buscar entonces es una data de miles de imágenes con la cual el modelo pueda aprender. Por lo tanto, el modelo es escogido por ser práctico y adecuado a resolver el caso del reconocimiento de las matrículas de los automóviles. Todo el código fuente y las instrucciones básicas se encuentran en la dirección url del autor [1]. Si bien la fuente está desarrollada en C y CUDA, existen repositorios con código abierto del entrenamiento y testeo para Yolov3, Yolov3-tiny, Yolov4 y Yolov4-tiny escritos en Python [2] [3] [4] para un mejor entendimiento del modelo. El sistema propuesto toma como base estos códigos para la

detección de las placas y se le incorpora la segmentación y reconocimiento de caracteres. El modelo tiene su complejidad para entenderlo completamente, tiene varios aspectos a tomar en cuenta como son el *anchor box*, función de costo, nms (*non-max suppression*), entre otros. Existen algunos sitios web que explican lo que el paper original no muestra o detalla [5] [6]. La Figura XX muestra cómo el modelo Yolo, en este caso su versión 3, siempre estuvo a la vanguardia y entre los mejores modelos para detectar objetos por su exactitud y velocidad.



Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

FIGURA XX: Ventajas del modelo YOLO frente a los competidores. Fuente: Joseph Redmon [6]

Ya decidido el por qué se usa este modelo, no se especificará la arquitectura de ésta debido a que ya se menciona en la Subsección 1), mas es necesario mostrar cuales fueron los parámetros o configuraciones base que se usaron para los entrenamientos iniciales (y que se detallará en el Capítulo VI):

- **Batch size**: 8 o 16
- **Input size**: 416
- **Data augmentation**: Activo
- **Transfer learning**: Activo
- **Learning rate inicial**: 0.0001
- **Learning rate final**: 0.000001

- **Épocas (epochs)**: 20, 30, 50, 80 o 100 (depende si es Yolo o Yolo-tiny)

2) Comparativa de modelos propuestos

La complejidad computacional de cada modelo puede analizarse considerando tanto el N° de parámetros como la cantidad total de operaciones de punto flotante (FLOPs - Floating Point Operations). Estas métricas permiten estimar el tamaño del modelo y la carga computacional requerida durante la inferencia y/o entrenamiento.

En la Tabla IX se presenta una comparación entre los modelos evaluados. Como se puede observar, los modelos "tiny" (YOLOv3-tiny y YOLOv4-tiny) poseen una cantidad significativamente menor de parámetros y FLOPs en comparación con sus versiones completas, lo cual los hace más adecuados para aplicaciones en tiempo real o dispositivos con recursos limitados. Por otro lado, YOLOv3 y YOLOv4 requieren más operaciones, pero pueden ofrecer una mayor precisión debido a su mayor capacidad de representación.

TABLA IX: Comparación de complejidad de modelos

Modelo	N° params	FLOPs
Yolov4-tiny	$5,880,324$	$6,40B$
Yolov3-tiny	$8,676,244$	$5,46B$
Yolov4	$64,003,990$	$59,71B$
Yolov3	$61,576,342$	$65,38B$

E Procesamiento y segmentación de las matrículas

Para esta sección es muy importante tener un conocimiento de todas las características de las matrículas (placas) de los automóviles en el Perú antes de trabajar con ellas. El sistema está basado principalmente para el territorio peruano para placas de **vehículos livianos y pesados** que, a comparación de las placas de vehículos menores, es un poco más grande. Recordar que en el Perú existen varios tipos de placas para vehículos, tanto menores o livianos como pesados, donde estos últimos tienen variaciones como servicio para el transporte particular, urbano, interprovincial o de carga, entre otros. Por

tal motivo, es necesario conocer las características de la placa peruana de estos tipos de vehículos antes de su procesamiento y segmentación.

1) Características de las placas de vehículos livianos y pesados en el Perú

La fuente oficial donde recae esta información se encuentra disponible para todo público [2]. A continuación, se detallan todas las características importantes:

- Las dimensiones de la matrícula son de $300mm \times 150mm \pm 2mm$ (ancho $\times$ altura). La relación de aspecto entonces es de $1:2$ (altura : ancho).
- Las dimensiones de las letras y números dentro de las matrículas son $35mm \times 80mm \times 10mm$ (ancho $\times$ altura $\times$ espesor). La relación de aspecto es de $16:7$ (altura : ancho).
- La relación de aspecto de la matrícula con un carácter es de $80/150 = 8:15$ respecto a la altura y de $35/300 = 7:60$ respecto a la anchura.
- Los bordes de la matrícula son del mismo tamaño en cada lado y es de $4mm$.
- La parte superior de la matrícula es de $35mm$, entonces *parte inferior = total_altura - borde - parte superior*, se calcularía así: $150mm - 4mm - 35mm = 111mm$.
- Entonces la relación de aspecto entre la **parte superior y el total** de la matrícula es $39:150$. Y la relación entre la **parte inferior y el total** es $111:150$.

Gráficamente, algunas mediciones hechas se representan en la Figura XXI.

FIGURA XXI: Mediciones sobre matrículas de autos livianos y pesados.
Fuente: Wikipedia

2) Segmentación de las matrículas

Cuando el modelo YOLO detecta las matrículas, éste guarda la imagen y llama al módulo de procesamiento y segmentación. Una vez que se tiene dicha imagen se procede a transformarla para que el sistema pueda extraer los números y letras. Entonces existe una fase de procesamiento en la cual se *limpia* y *clarifica* las letras y números. Luego se procede a extraer cada uno y se guarda en una carpeta. Esta fase de procesamiento es importante porque nuestro modelo YOLO detectará varias matrículas diferentes, es decir, en diferentes ángulos, proximidades y resoluciones. Además, las matrículas en las calles no siempre están nuevas o limpias, generalmente habrá manchas u obstrucciones que dificulten la extracción de los caracteres, entonces por estos motivos es importante una fase que elimine las dificultades y facilite dicha extracción. Luego se procede a segmentar la matrícula y obtener los caracteres. La Figura XXII muestra este proceso y cada parte es explicada en el siguiente párrafo.

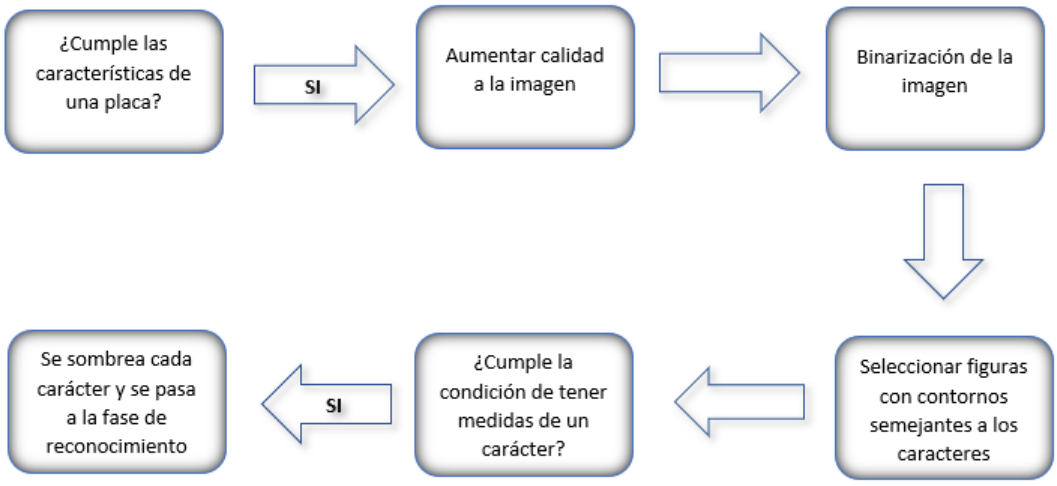


FIGURA XXII: Diagrama del proceso de segmentación de la matrícula.

El procesamiento de la imagen es como sigue:

1. Verificar si la matrícula cumple la relación de aspecto de una matrícula de vehículo liviano y pesado del Perú ($1 : 2 = 0.5$) más un rango de error de ± 0.2. No es difícil ver que el rango de aceptación entonces es $(0.3 \ ; \ 0.7)$. Esto con la finalidad de asegurarnos de que la imagen es una matrícula.
2. Luego de que se verifica que es una matrícula, se procede a redimensionarla a una matrícula con altura 150 y el ancho se obtiene dividiendo 150 entre la *relación de aspecto*.
3. Dado que ya se tiene relación de aspecto entre la altura de la matrícula y la parte superior, definido en la sub-sección anterior, se procede a eliminar dicha fracción de la imagen.
4. Ahora que se tiene la imagen recortada donde están las letras y números, se desea hacer la imagen más nítida y que se mejoren los detalles, esto para *clarificar* las letras y números. Esto es posible con la función de la librería OpenCV `detailEnhance`.
5. Ya obtenida la imagen mejorada, se procede a convertirla a escalas de grises, esto porque se pretende que los caracteres lleguen a ser de color negro con fondo blanco

igual que los datos escogidos para nuestro modelo de reconocimiento mostrado en la Sección C. Para realizar tal cambio a escalas de grises se usa la función `cvtColor` de OpenCV.

6. El propósito de pasar a escala de grises fue para que posteriormente se binarize la imagen, es decir, convertir la gráfica a blanco y negro. La función `threshold` en Opencv binariza la gráfica a escala de grises, y pide un parámetro llamado `thresh` en la cual el valor de un píxel si es mayor que dicho thresh se convierte en blanco o negro dependiendo de otro parámetro de la función.
7. Luego de ser procesado y mejorado la imagen, comienza la parte de segmentación. Para tal propósito primeramente se deben encontrar las figuras con contornos o lados de un tamaño aproximado al de las letras y números de las placas. Esto se realiza fácilmente con la función `findContours` de OpenCV.
8. Una vez ya obtenida las figuras con los contornos sombreados, para pasar a la siguiente etapa se seleccionan solo las figuras que cumplan la condición de tener un área mínima y máxima, estos parámetros de área mínima y máxima han sido obtenidos empíricamente a través de varias experimentaciones, con lo cual validamos aún más que se tratan de letras y números.
9. Luego se dibuja un cuadrado que contiene a la figura para que pase a otro filtro. Dicho filtro es que tal cuadrado tenga una altura mínima, este parámetro mínimo ha sido obtenido también empíricamente a base de experimentaciones. Si logra cumplir tal condición de altura mínima entonces podemos estar más seguros que dicha figura es una letra o número.
10. Luego de tener todas las letras o números encuadradas se procede a realizar un recorte a la matrícula cuadro por cuadro para obtener todos los caracteres deseados. A cada carácter se le añade más fondo blanco en cada lado, esto para asemejar la imagen al dataset entrenado (*Chars74k*).

11. Finalmente, se ordena cada imagen obtenida de acuerdo a su coordenada horizontal y se guarda, si se desea, en una carpeta cada una de ellas.

La Figura XXIII muestra las etapas descritas del procesamiento de la matrícula una vez que recibe la imagen del modelo YOLO:

1. **Proceso 0**: Imagen Original.
2. **Proceso 1**: Redimensionamiento de la imagen si cumple la proporción mencionada de ser una matrícula vehicular.
3. **Proceso 2**: Parte superior suprimida de la imagen.
4. **Proceso 3**: Mayor nitidez y conversión a escalas de grises.
5. **Proceso 4**: Binarización de la imagen.
6. **Proceso 5**: Sombreado de contornos y encuadrado de letras y números.

FIGURA XXIII: Proceso de segmentación de una matrícula peruana.

F Modelo de reconocimiento de caracteres

El último módulo del sistema es del reconocimiento de los caracteres, es decir, clasificar cada letra o número, si es *la A*, *la H* o el *7*. Para tal motivo es necesario un

modelo clasificador de imágenes, particularmente un clasificador de letras mayúsculas y números. Se usaron 3 modelos para comparar cuál de ellos era el mejor clasificador. Éstos fueron: **un modelo de elaboración propia**, el **modelo AlexNet** y el **modelo LeNet-5**. Las salidas obtenidas se muestran en el Capítulo VI. Fueron realizadas las pruebas necesarias para medir la efectividad de cada modelo y fue el **modelo LeNet-5** quien tuvo una mejor precisión que los otros. La arquitectura fue ligeramente cambiada para que se ajuste a la aplicación requerida de reconocer las letras y números. Dicha variación fueron cambios en las capas o algunos parámetros de éstos, finalmente quedó así:

1. **Primera Capa**: Capa Convolucional con kernel $(5, 5)$, 64 filtros, una gráfica en escala de grises como entrada (*input*) con dimensión $(64, 64)$, no padding y la función *ReLU*.
2. **Segunda Capa**: Capa MaxPooling con kernel $(2, 2)$.
3. **Tercera Capa**: Capa Convolucional con kernel $(3, 3)$, 64 filtros, no padding y función de activación *ReLU*.
4. **Cuarta Capa**: Capa MaxPooling con kernel $(2, 2)$.
5. **Quinta Capa**: Capa Convolucional con kernel $(3, 3)$, 64 filtros, no padding y función de activación *ReLU*.
6. **Sexta Capa**: Capa MaxPooling con kernel $(2, 2)$.
7. **Septima Capa**: Capa Flatten, cuya función es agrupar todos los valores de los filtros en un vector.
8. **Octava Capa**: Capa Dense, con 1024 neuronas y función *ReLU*.
9. **Novena Capa**: Capa Dropout, que ayuda a prevenir el *overfitting*.
10. **Décima Capa**: Cada BatchNormalization que ayuda a normalizar la capa siguiente.
11. **Undécima Capa**: Capa Dense de salida $(output)$ con 36 neuronas (36 clases, 26 letras y 10 números) con función de activación *Softmax*.

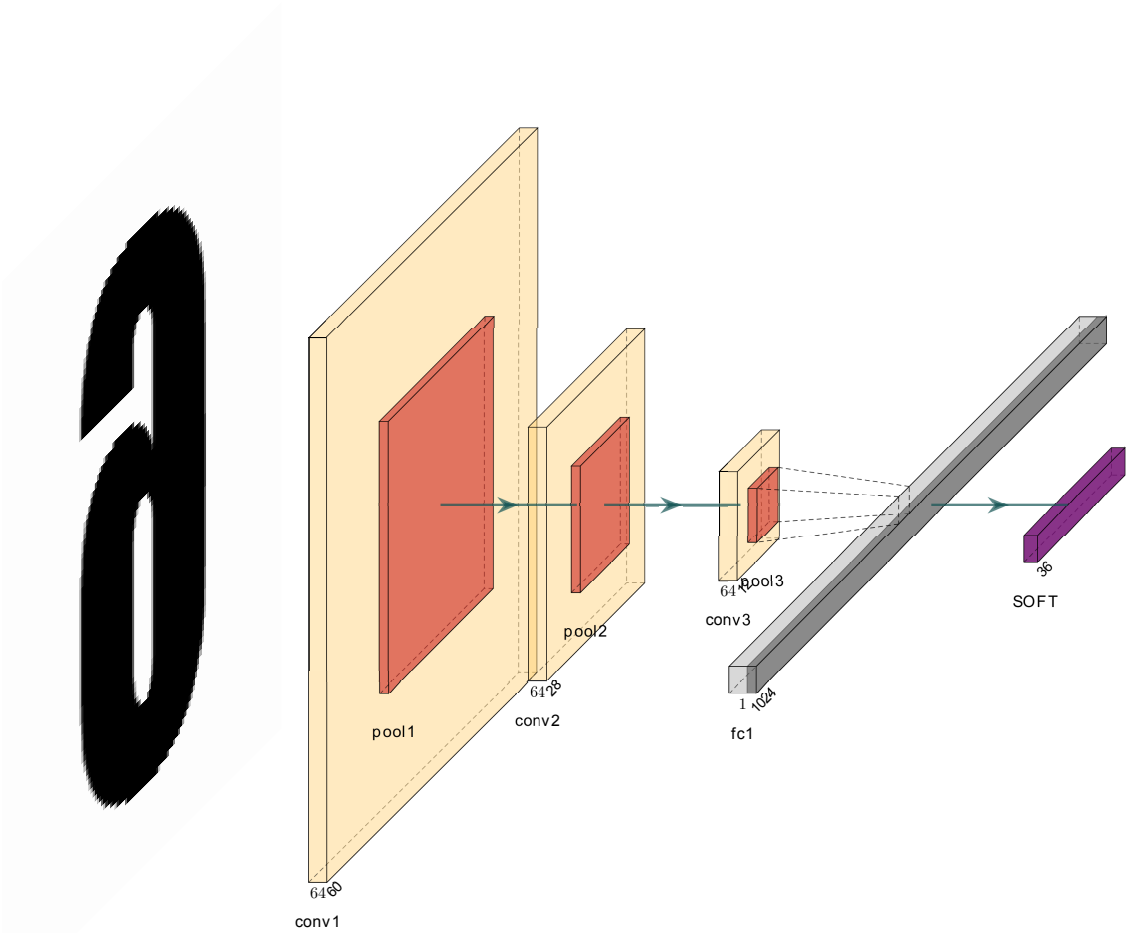


FIGURA XXIV: Modelo LeNet-5 personalizado.

En este contexto, ya comprobada la efectividad del modelo para esta tarea de clasificación de caracteres, es necesario analizar cada variación del modelo, esperando que pueda optimizarse un poco más. Los resultados de estas variaciones del modelo están también en el Capítulo VI donde describimos que el NN final tiene el modelo presentado en la Figura XXIV, además se muestra la cantidad de variables (*números a operar*) en cada capa.

G Criterio de evaluación de Pruebas

Un punto importante a resaltar es la forma de evaluación que aplicaremos al modelo propuesto. Para este sentido, se dan las siguientes consideraciones:

- Una predicción acertada es que el modelo prediga correctamente la placa y también todos los valores alfanuméricos. Este será nuestro predicción correcta o **True Positive (TP)**.
- Una predicción negativa o **Negative (N)** es cuando el sistema no logra detectar la placa o no logra detectar la cantidad exacta de caracteres que hay en la placa.
- Una predicción correcta por el sistema pero errónea realmente o **False Positive (FP)** es cuando el sistema detecta la placa y la cantidad exacta de caracteres de la placa, pero no todos los caracteres son los correctos.
- La precisión del sistema o **Precision** es un punto tan importante como el *Accuracy* o incluso más debido a que esta métrica mide de todas las predicciones hechas por el sistema, cuales los ha realizado correctamente. Pero hay dos puntos importantes sobre esta métrica: **1.-** Las predicciones hechas por el sistema son en base a que el detector de placas capturó y enmarcó correctamente la placa. Esto debido a que la placa no estaba deteriorada, doblada, girada bruscamente o con materiales encima de la placa que dificulte su detección. **2.-** Si el sistema logra detectar la placa, el preprocesamiento de la placa no debe tener impedimentos para segmentar y separar los caracteres y esto sucede cuando la placa no esta manchada, agrietada, o deteriorada. Para el caso de detección en vivo se debe tener en cuenta las dificultades del ambiente de pruebas, como se detallará más adelanta en la Subsección 2) para el caso de la Puerta N°5 de la UNI.

$$Precision = \frac{TP}{TP + FP} \tag{IV.1}$$

- La exactitud final o **Accuracy** de nuestro sistema cuando detecta correctamente la placa y caracteres viene dado por la siguiente fórmula:

$$Accuracy = \frac{TP}{TP + N + FP} \tag{IV.2}$$

Entonces con esta formula abarcamos los casos que el sistema no pudo detectar la placa o todos los caracteres (N) más los casos donde si predijo bien (TP) y no predijo bien (FP). Si bien es cierto que existen 2 modelos o módulos en el sistema propuesto, se ha pensado esta métrica con el propósito de medir la eficacia de todo el sistema integral, incluyendo el módulo de procesamiento de la imagen que es el intermediario entre los 2 modelos mencionados.

La fórmula propuesta del Accuracy evaluará el modelo en diferentes escenarios: predicciones en imágenes, predicciones en videos y por último en tiempo real (particularmente el lugar experimentado - Puerta N° de la UNI)

- Para el módulo de detección de placas *(dp)* la métrica más importante es la del **Recall**, dado que queremos saber que tantas placas a detectado el sistema en base al total de placas verdaderas. La fórmula es la siguiente:

$$Recall_{dp} = \frac{TP_{dp}}{TP_{dp} + FN_{dp}} \tag{IV.3}$$

Donde $Recall_{dp}$ es el Recall de la detección de placas, TP_{dp} son las placas bien predecidas y FN_{dp} son las placas que no fueron predecidas por el detector.

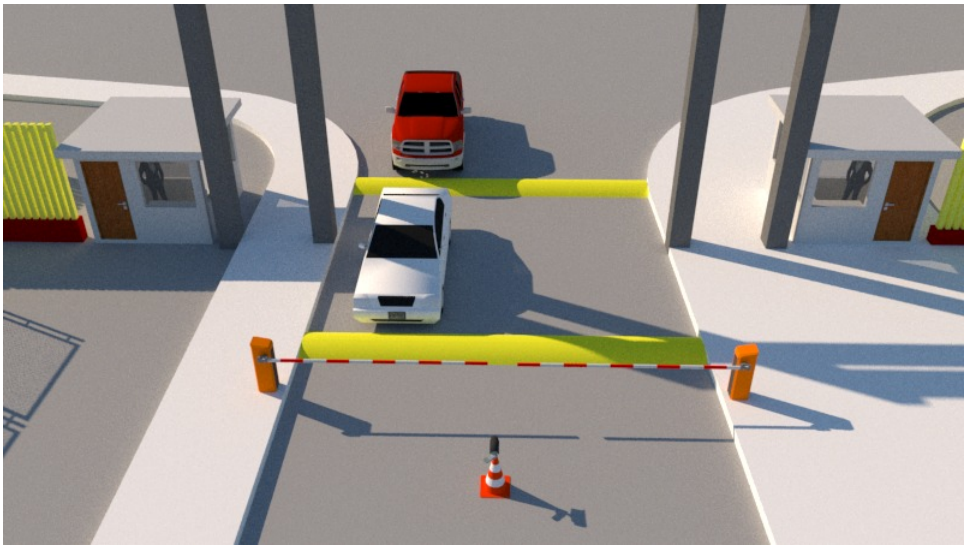
V. Ambiente de Experimentación y Pruebas

Este capítulo muestra la configuración del ambiente en donde se realizaron las pruebas en video que la cámara capta, así como su correcta implementación del sistema propuesto. La Sección A detalla el diseño del ambiente: muestra cómo está distribuido cada recurso, cómo están conectados cada uno y por qué se eligió el escenario de la puerta N°5 de la UNI. La Sección B presenta una guía de implementación para que el ambiente pueda ser desplegado correctamente, ya sea en la entrada del campus como en este caso o en algún otro lugar. Adicionalmente, se detallan las consideraciones a tomar en cuenta y las dificultades que pueden ocurrir al momento de la implementación.

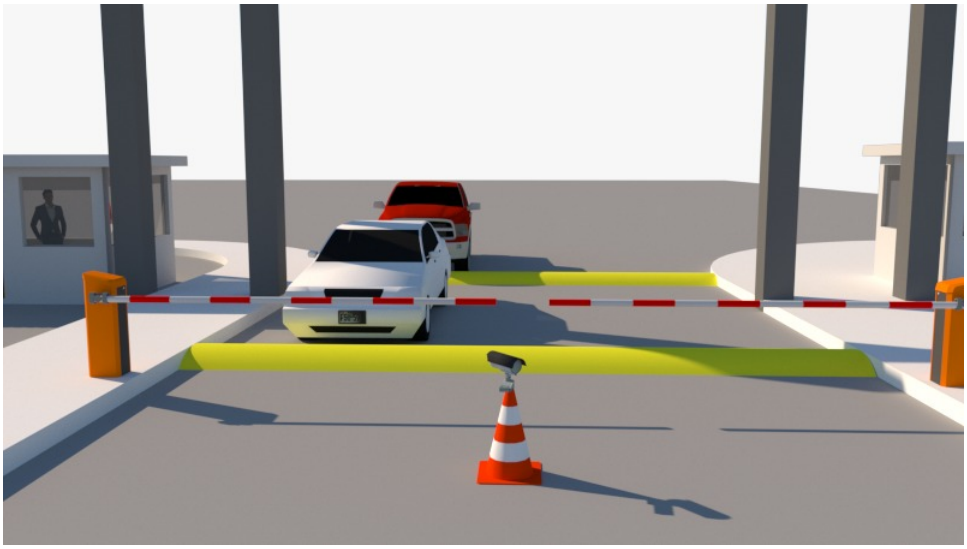
A Diseño de Ambiente de Prueba - Puerta N°5 de la UNI

El diseño del ambiente tenía que ser un espacio donde haya una circulación vehicular recurrente para así poder aprovechar la mayor cantidad de placas a predecir por el sistema. El lugar escogido fue la Puerta N°5 de la UNI, dado que es la principal entrada vehicular al campus de la universidad, por lo que su afluencia vehicular es grande y más aún los días de semana. La Figura XXV muestra el acceso vehicular desde dentro del campus y la posición y ángulo de la cámara usada para las pruebas.

Ahora, si bien la Puerta N°5 tiene 2 vías (entrada y salida), las pruebas están principalmente para capturar las placas que ingresen al campus, debido a la precisión y velocidad de reconocimiento que puede influir en detectar 2 placas al mismo tiempo. Además, es recomendable tener 2 cámaras en caso de que se tenga que detectar ambas vías para una mejor disponibilidad y rapidez en la detección de la matrícula.



(A) Vista general de la cámara dentro del campus de la UNI



(B) Cámara girada 45° observando las placas

FIGURA XXV: Puerta N°5 - Universidad Nacional de Ingeniería.

Las imágenes anteriores son una representación del ambiente de trabajo puesto en marcha por meses, trayendo todas las muestras posibles, tanto para la evaluación con video en vivo como con las grabaciones recogidas de la cámara.

A continuación, las medidas aproximadas entre los objetos significativos (cámara, vehículo objetivo y cono) para las pruebas en 2 diferentes ángulos:

Ángulo Frontal

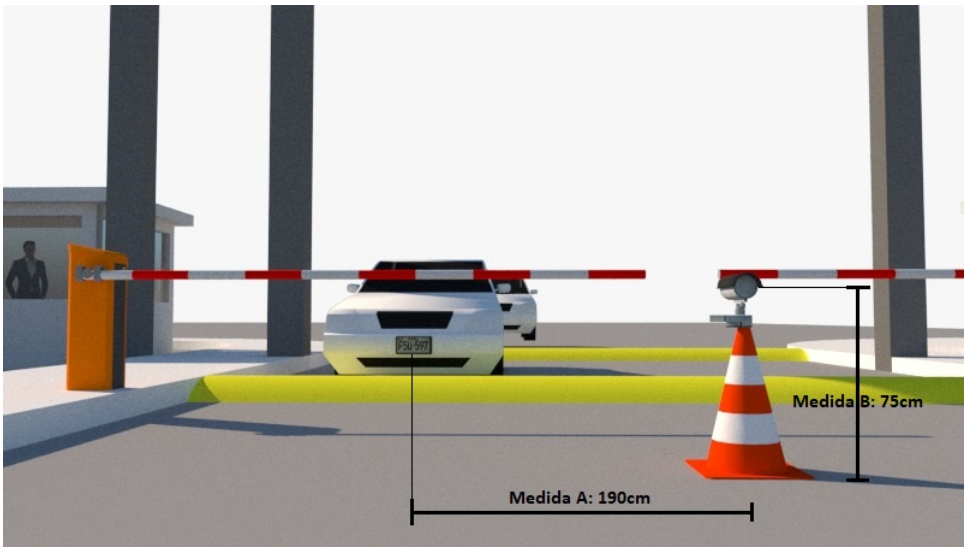


FIGURA XXVI: Medidas en la vista frontal de la cámara y la placa vehicular.

Las medidas son las siguientes:

- **Medida A:** 190cm
- **Medida B:** 75cm

Ángulo Vertical

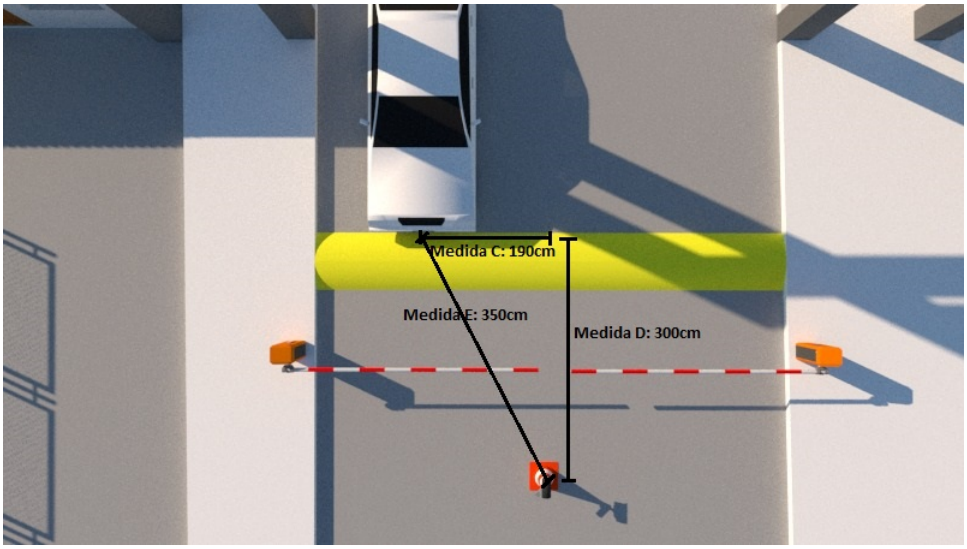


FIGURA XXVII: Medidas en la vista superior de la cámara y la placa vehicular.

Sus medidas correspondientes son las siguientes:

- **Medida C:** 190cm
- **Medida D:** 300cm
- **Medida E:** 350cm

La posición de la cámara fue variando a lo largo de las pruebas realizadas; se tuvieron además 2 ángulos adicionales diferentes en los que se realizaron las pruebas:

(A) La cámara encima de la reja principal de la puerta N°5 y enfocando hacia abajo (ángulo de inclinación vertical de -45° aproximadamente), la cual fue descartada debido a que el ángulo era muy desfavorable para detectar las placas.

(B) La cámara atrás de la reja principal y apoyado en una silla o cono. La silla estaba puesta a un costado de la entrada, no directamente en la vía, sino en la vereda derecha de la reja, observándolo desde afuera. La Figura XXVIII muestra la posición de la cámara respecto a los vehículos. Este escenario se fue probando hasta encontrar el ideal, la cual se observa en la Figura XXV.

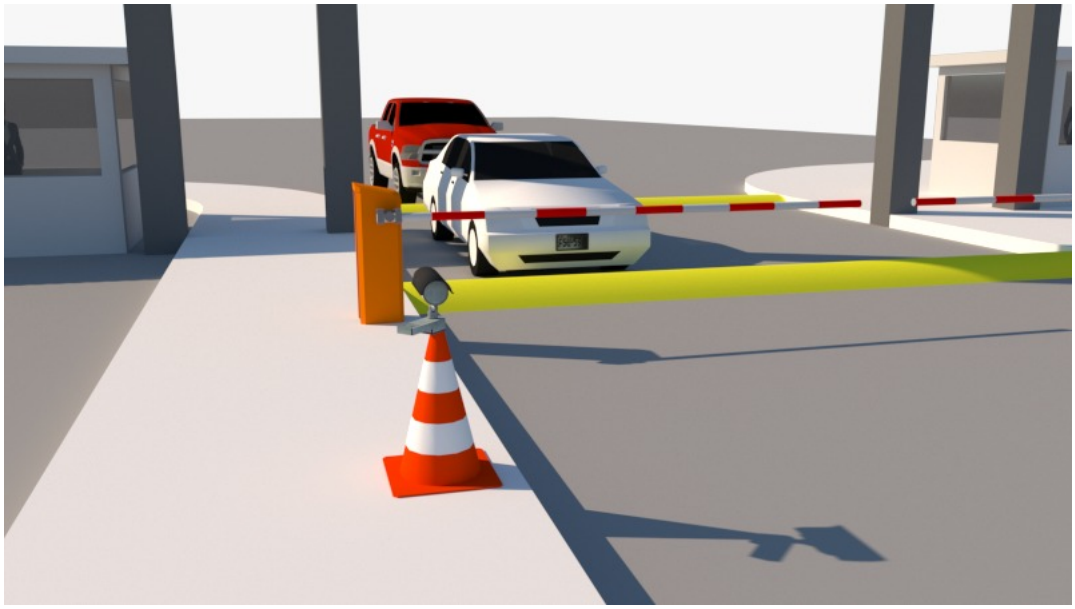


FIGURA XXVIII: Posición de la cámara en Pruebas iniciales dentro del campus de la UNI.

B Implementación de ambiente

La puesta en marcha del ambiente tiene una secuencia sistemática para desplegarlo correctamente. Como parte del despliegue se incluye el sistema propuesto en modo prueba en la laptop, que se detallará más adelante. Mencionar que se tuvieron 3 formas de implementación por cada ubicación de la cámara, la cual se especificó en la Sección A, pero finalmente se mantuvo la mostrada en la Figura XXVa, siendo esta la que perduró por más tiempo que las anteriores. Los materiales para el despliegue son los siguientes:

- *Cámara PoE*: El recurso principal para obtener los videos de los carros ingresando al campus de la universidad.
- *Laptop*: Fue usada la laptop descrita en Tabla A, la cual se usa para ejecutar el programa de detección de placas por medio de un terminal Linux (la laptop cuenta con SO Ubuntu).

- *Router básico*: Para la creación de la red personal que sirve como el nexo y conexión entre la cámara y la laptop que ejecuta el sistema.
- *Inyector PoE*: Importante para dar energía y conectividad a la cámara solo por medio de un cable de red.
- *Cables de red*: El cable principal aproximadamente 15 metros para poder distribuir la cámara sin ningún problema. Otros cables de 2 metros para conexiones entre router a laptop y del inyector al router.
- *Cono vehicular*: Como soporte de la cámara.
- *Otros materiales adicionales*: Como extensión de enchufe, cinta de embalaje para asegurar la cámara al cono, entre otros.

La Figura XXIX muestra la puesta en marcha de la cámara, luego de implementar e integrar todos los recursos y dejar listo el ambiente de pruebas. Los pasos de la implementación se detallan en los siguientes párrafos.

FIGURA XXIX: Configuración de conexión de red.

La implementación es llevada a cabo en la siguiente secuencia:

1. Conectar la laptop con el router con un cable de red.
2. Conectar el inyector con el router con un cable de red.
3. Conectar el inyector con la cámara con un cable de red (este cable debe ser el más grande que los otros).
4. Sujetar firmemente la cámara encima del cono vehicular con cinta embalaje.
5. Colocar el cono en medio de la vía y después de la entrada a la puerta N°5 - UNI. Además que sujetar el cable de red con el piso usando la cinta de embalaje (esto para que al pasar los autos no se lleven el cable y la cámara)
6. Dar corriente al router, inyector y laptop por medio de la extensión.
7. Correr el sistema en la laptop previamente configurado con la ruta correcta de la cámara.

1) Consideraciones en la implementación

Si bien es cierto que se implementó el sistema detector de placas, es preciso mencionar las consideraciones que sirvieron para la puesta en marcha en la Puerta N°5:

- **Permiso de uso del espacio:** Este permiso debe darse por el jefe de la seguridad en la UNI, dado que el ambiente ocupa el espacio donde los trabajadores de seguridad brindan su servicio.
- **Horario de pruebas:** La afluencia vehicular es mayor los días de semana en comparación que los fines de semana. Por disponibilidad, se realizaron pruebas algunos días de semana y en mayor medida más días de fin de semana.
- **Memoria externa para el almacenamiento:** Para tener los videos que graba la cámara, es necesario poseer una memoria microSD que será puesta en la ranura de la cámara. Para este caso, se usó una memoria de 8GB y por cada día de grabación de 4 horas aproximadamente se guardaban las grabaciones en la laptop y dejaba limpio el microSD, listo para otro día con 4 horas de grabación.

2) Dificultades en la implementación

Existieron dificultades durante la implementación y en la recopilación de muestras de video, las cuales se detallan a continuación:

- **Rompemuelle:** La puerta N°5 tiene un rompemuelle en la entrada, esto facilita la detección de la placa, dado que es más fácil para el detector que el objeto esté estático que se mueve constantemente. Además que todos los vehículos deben estacionarse momentáneamente antes de cruzar el rompemuelle para ser registrado él y su vehículo por el personal de seguridad. Sin embargo, existen escenarios donde autos pasan de frente, sin registro y con una velocidad no menor (esto dado que el personal ya los tiene mapeados como profesores recurrentes o autoridades de la Universidad que transitan en todo momento por el campus) y esto genera que el detector pueda tener un grado de precisión menor al promedio.
- **Afirmar el cable de red al suelo:** Luego de tener el ambiente listo, tal cual se expuso en esta sección, existe la posibilidad que en un pequeño descuido el cable de red no este sujeto al suelo y el vehículo que transite jale el cable y la cámara, pudiendo malograrlo hasta incluso romperlo si la velocidad del auto es alto. Por tal motivo es importante afianzar el cable de red al suelo o por donde vaya.

VI. Experimentación y Resultados

Aquí se presentan los resultados del entrenamiento de los modelos usados: Yolov3, Yolov3-tiny, Yolov4-tiny y LeNet-5. Cada modelo se entrenó bajo distintas variaciones y fueron escogidas las mejores personalizaciones. La primera sección muestra los resultados del entrenamiento y predicción con el modelo detector de matrícula. En la Sección C tenemos los resultados de procesamiento a la matrícula. Finalmente, en la última sección se comparan los resultados de distintos entrenamientos para hallar el mejor modelo de clasificación de caracteres.

A Detección de matrículas usando Yolov3 y YOLOv4

Para un inicio, el modelo Yolov3 fue pensado como el detector de matrículas; sin embargo, este fue dejado de lado porque demanda muchos recursos computacionales y solo es usado para fines experimentales. Por tal motivo, no se entrenó el modelo Yolov4 debido a que, luego de la interpretación previa, no nos serviría tal modelo, pero su versión pequeña sí sería útil. Además, hoy en día se usan mayormente estos modelos para fines de investigación y competir con otros modelos que obtienen altos resultados de precisión. Pero, para una aplicación de detector de objetos con bajos recursos, es difícil usar este modelo por su gran demanda computacional; en su lugar, investigadores han desarrollado las versiones pequeñas o *tiny* para sistemas embebidos y que ofrecen un rápido proceso de detección, ideal para usarlo en tiempo real con una cámara. Por tal motivo, como se verán en los gráficos de los entrenamientos, el modelo Yolov3 fue entrenado con 20 y 30

épocas, el modelo Yolov3-tiny con 100 épocas y el modelo Yolov4-tiny con 50 épocas. A continuación, se detallan las observaciones que se rescatan del entrenamiento:

- Los entrenamientos de los modelos Yolov3 y Yolov3-tiny fueron realizados con el clúster Jaguar, descrito en la Sección A, por el hecho de que YOLOv3 necesitaba al menos $5Gb$ para un procesamiento óptimo y además que el clúster tiene la facilidad de estar encendido las 24 horas del día.
- El primer entrenamiento de Yolov3 fue realizado con 20 épocas, esto para medir en primera instancia qué tan bien entrenaba el modelo con pocas épocas. Dando un *validation loss* de 2.121.
- El tiempo que tomó en entrenar para esta primera prueba fue de *425 minutos* (*7 horas y 5 minutos*) dando un promedio de *21.25 minutos por época*.
- Las pruebas realizadas eran alentadoras, sin embargo, se quiso medir qué mejora se podría alcanzar con 10 épocas más.
- El segundo entrenamiento de Yolov3 con 30 épocas tomó un tiempo aproximado de *11 horas*. La mejoría fue escasa, es decir, solamente una disminución de 0.077 en la función de coste en la validación: 2.044.
- Los logros fueron prometedores en su momento y fue determinado que sería el modelo de detección de matrículas hasta entonces.
- Luego el modelo Yolov3-tiny fue entrenado con 100 épocas y tomó aproximadamente *1 día*. El *validation loss* obtenido luego de las 100 épocas fue de 2.158
- Después de examinar los resultados de Yolov3 y Yolov3-tiny, se procedió a entrenar el modelo Yolov4-tiny con 50 épocas inicialmente para ver su comportamiento y observar si converge antes de 50 épocas o necesita más épocas para la convergencia. El entrenamiento duró *7 horas y 25 minutos*.
- Finalmente se entrenó el modelo Yolov4-tiny con 80 épocas para visualizar si tenía una mejora con respecto al modelo con 50 épocas. Sin embargo, el comportamiento

del modelo se estabilizó antes de la época 50 según las gráficas dadas. Es más, en el modelo con 80 épocas el *validation loss* fue de 1.93 y en el de 50 épocas de 1.83, siendo mejor este último. Entonces, según las pruebas empíricas el modelo Yolov4 da mejores resultados que su antecesor Yolov3.

- Un punto a tomar en cuenta fue que la conexión con el clúster Jaguar fue por medio de SSH (Secure Shell) que permite la comunicación por terminal, pero no con interfaz gráfica. Entonces fue necesario llevar los *logs* a la *Laptop Aspire* o a la *computadora de escritorio* para que los gráficos del entrenamiento sean observados con la herramienta Tensorboard.

El entrenamiento realizado de Yolov3 con 20 y 30 épocas y Yolov3-tiny con 100 épocas se muestran en la Figura XXX, Figura XXXI y Figura XXXII. Las figuras muestran la función de costo en la etapa de validación. Como se observa en los gráficos, la función de costo tiende a converger en la época 10 en el modelo Yolov3 y en la época 20 para el modelo Yolov3-tiny.

Por el lado del modelo Yolov4-tiny la Figura XXXIII muestra el entrenamiento con 50 épocas y la Figura XXXIV con 80 épocas. La convergencia se puede notar gráficamente a partir de la época 40.

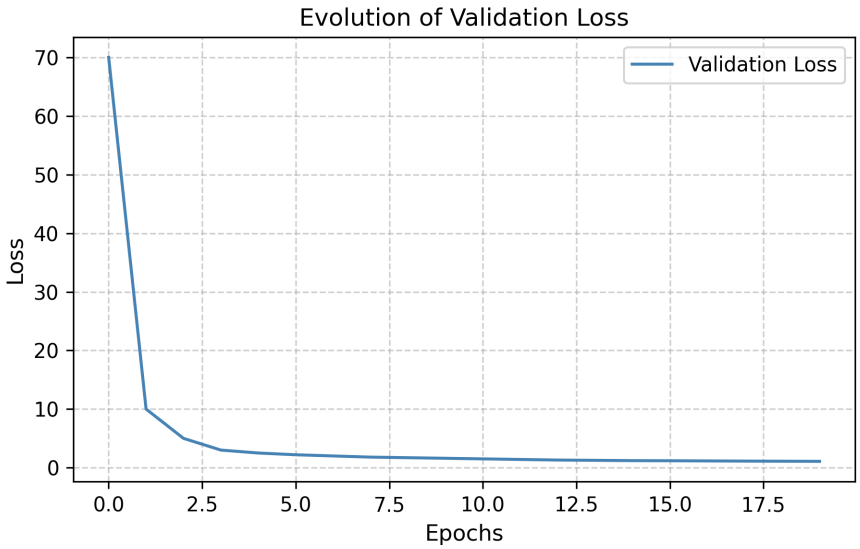


FIGURA XXX: Función de Costo del entrenamiento de Yolov3 con 20 épocas.

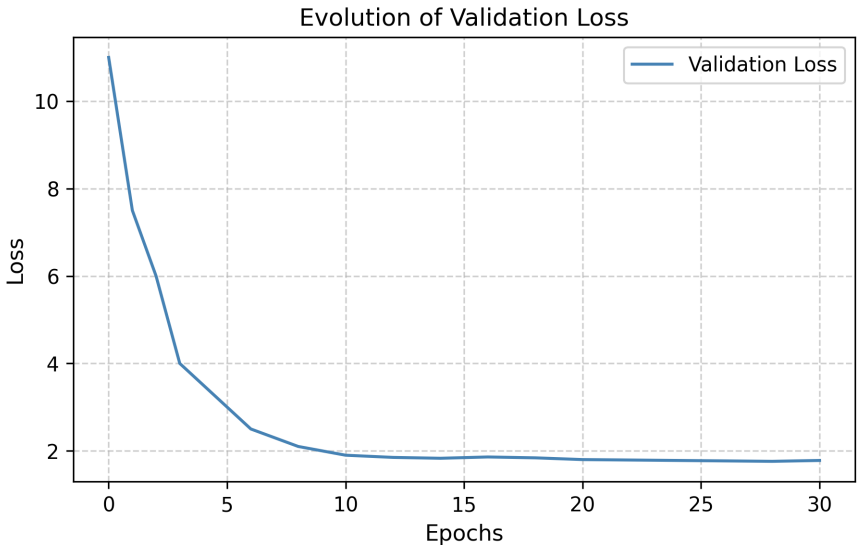


FIGURA XXXI: Función de Costo del entrenamiento de Yolov3 con 30 épocas.

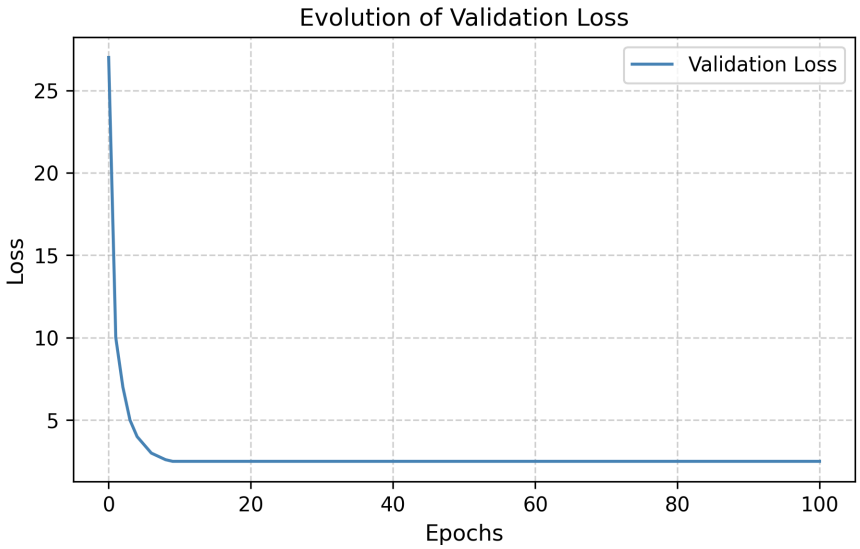


FIGURA XXXII: Función de Costo del entrenamiento de YOLOv3-tiny con 100 épocas.

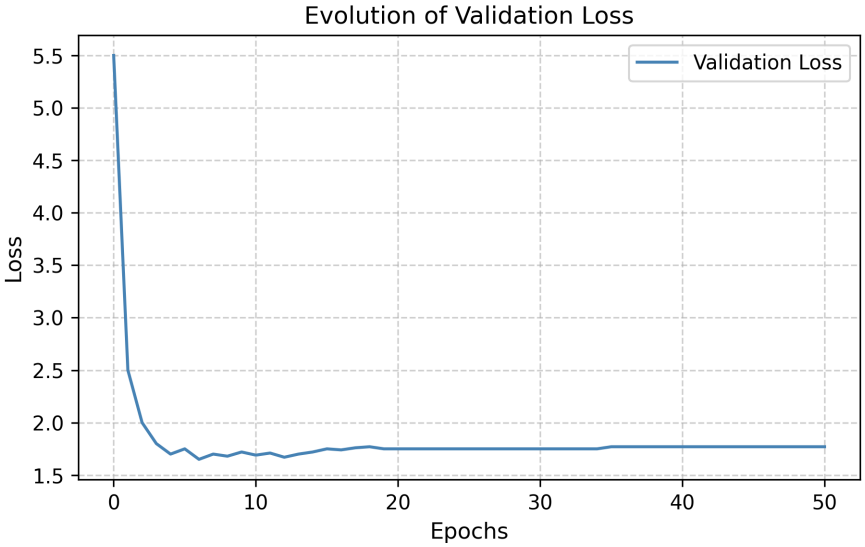


FIGURA XXXIII: Función de Costo del entrenamiento de YOLOv4-tiny con 50 épocas.

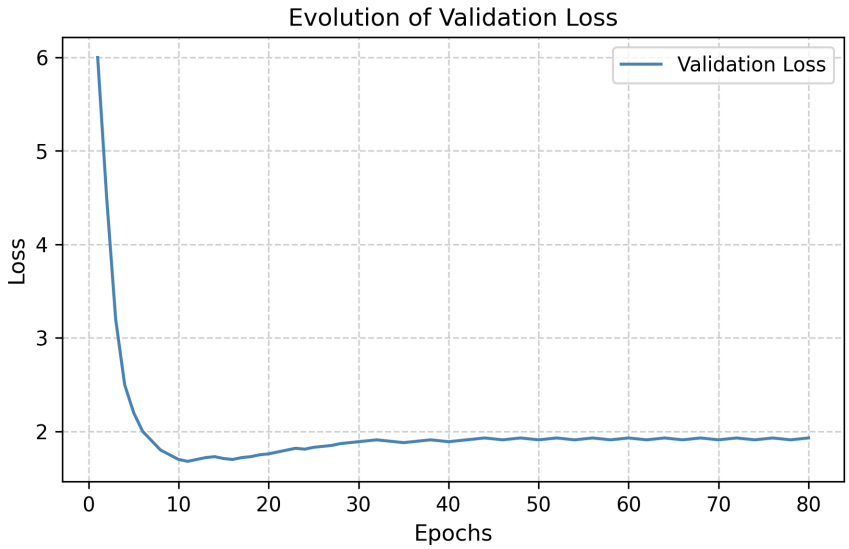


FIGURA XXXIV: Función de Costo del entrenamiento de Yolov4-tiny con 80 épocas.

B Optimización de Hiperparámetros - Fase de Entrenamiento del modelo Yolo

En esta sección, se identifican las configuraciones óptimas para el entrenamiento del modelo YOLO. La personalización del modelo es fundamental para adaptarlo al problema específico de detección de matrículas vehiculares, garantizando así la obtención de los mejores resultados posibles.

El modelo Yolo, según vimos en la Sección B, cuenta con los anchor boxes que de forma predeterminada tienen un ancho y alto fijos para diferentes escalas (esto porque se entrenó con el dataset COCO, el cual tiene una variedad de tipos de objetos como personas, animales, etc.), donde Yolov4-tiny tiene 2 escalas a optimizar. Estos valores deben ajustarse al objeto que vamos a predecir (placas).

Para identificar los anchors que mejor se ajusten a las características de las placas vehiculares, se utilizó el algoritmo **k-means++** sobre nuestro dataset de placas peruanas visto en la Sección 2). Este método permitió agrupar los tamaños (ancho y alto) de los bounding boxes y calcular las dimensiones óptimas de los anchors para nuestro conjunto

de datos. A diferencia de los anchors predeterminados de YOLOv4-tiny, diseñados para objetos genéricos como los del dataset COCO, el uso de **k-means++** asegura que los valores de los anchors estén adaptados específicamente a las matrículas, lo que mejora la precisión y eficiencia del modelo.

Adicionalmente, otra propiedad de Yolo es la cantidad máxima de bounding boxes que el modelo puede predecir. Para nuestro caso, donde las placas no están superpuestas una sobre otra, sino que siempre hay una en todo el área de la imagen (no necesariamente, pero vamos a enfocarnos en 1 sola placa de la imagen), vamos a indicar dicho parámetro en 1.

En resumen, estos parámetros se configuran con los siguientes valores:

$$YOLO\_ANCHORS = \begin{bmatrix} [12, 11], & [14, 14], & [15, 16], \\ [17, 18], & [19, 22], & [21, 27], \\ [0, 0], & [0, 0], & [0, 0] \end{bmatrix}$$

$$YOLO\_MAX\_BBOX\_PER\_SCALE = 1$$

Para la búsqueda de los otros hiperparámetros, se emplea el **Método GridSearch**. Dado que contamos con los recursos computacionales disponibles las 24 horas, hemos optado por este enfoque exhaustivo para garantizar que se prueben todas las combinaciones posibles.

Como resumen de los hiperparámetros configurados se tiene la Tabla X.

TABLA X: Resumen de hiperparámetros configurados

Hiperparámetro	Valores
YOLO IOU LOSS THRESH	[0.3, 0.5, 0.7]
YOLO INPUT SIZE	[416]
TRAIN BATCH SIZE	[4, 8, 16]
TRAIN LR INIT	[1e-4, 1e-6]
TRAIN LR END	[1e-6, 1e-8]
TRAIN WARMUP EPOCHS	[2, 4, 6]
TRAIN EPOCHS	[20, 30, 40]

El detalle de cada hiperparámetro testeado es el siguiente:

- **YOLO IOU LOSS THRESH [0.3, 0.5, 0.7]**: Este es el umbral de la pérdida de IoU (*Intersection over Union*) se refiere a qué tan bien el modelo está prediciendo la ubicación de los objetos en relación con los cuadros delimitadores reales. Valores usados: **[0.3, 0.5, 0.7]**, indicará diferentes configuraciones del umbral. Si el IoU entre el cuadro predicho y el cuadro real es menor que este valor, el modelo considerará que la predicción es incorrecta.
- **YOLO INPUT SIZE [416]**: Este tamaño de entrada son las dimensiones (ancho x alto) de las imágenes que alimentan al modelo. En YOLO, las imágenes se redimensionan a tamaños cuadrados, y el tamaño establecido según los pesos pre-entrenados es de 416x416. Valores usados: **[416]**, indica que probaremos solo un valor posible dado que la red neuronal fue previamente entrenada con este *input size*. Las imágenes de mayor resolución son aceptadas, pero luego son redimensionadas a este valor.
- **TRAIN BATCH SIZE [4, 8, 16]**: El tamaño del batch (lote) indica cuántas imágenes se procesan juntas antes de actualizar los pesos del modelo. Valores usados: **[4, 8, 16]** para determinar el más adecuado. Un tamaño de batch más grande puede estabilizar el proceso de entrenamiento, pero requiere mayor memoria. Dado que las imágenes usadas para entrenar tienen una resolución alta (*3360x2100*), es importante tener en cuenta la capacidad de memoria total del GPU (VRAM) disponible. Un batch size

demasiado grande podría exceder los recursos de hardware, limitando la viabilidad del entrenamiento en GPUs con menor capacidad.

- **TRAIN LR INIT [1e-4, 1e-6]**: Esta es la tasa de aprendizaje inicial (Learning Rate, LR) que controla qué tan rápido el modelo ajusta sus pesos en cada iteración al principio del entrenamiento. Valores usados: **[1e-4, 1e-6]**, representa dos configuraciones de tasa de aprendizaje inicial, donde un valor más alto significa cambios más rápidos, pero puede resultar en inestabilidad, mientras que un valor más bajo ofrece un entrenamiento más estable y gradual.
- **TRAIN LR END [1e-6, 1e-8]**: Esta tasa de aprendizaje final es la que se alcanzará al final del entrenamiento. A medida que el entrenamiento avanza, el valor de la tasa de aprendizaje va disminuyendo para afinar los ajustes finales del modelo. Valores usados: **[1e-6, 1e-8]**, son dos configuraciones para el LR final del entrenamiento. Y además esta relacionado con el *TRAIN LR INIT* en el sentido que se forman dos combinaciones: *LR INIT de* $1e-4$ *con LR END de* $1e-6$. Y *LR INIT 1e-6 con LR END de 1e-8*.
- **TRAIN WARMUP EPOCHS [2, 4, 6]**: El 'warmup' se refiere a un período inicial durante el cual la tasa de aprendizaje aumenta gradualmente desde un valor muy pequeño hasta el valor especificado en *TRAIN LR INIT*. Esto ayuda a evitar saltos bruscos en el ajuste de los pesos al inicio del entrenamiento. Valores usados: **[2, 4, 6]**, donde generalmente un valor bajo es suficiente.
- **TRAIN EPOCHS [20, 30, 40]**: Los 'epochs' son el número de veces que el modelo verá todo el conjunto de datos durante el entrenamiento. Un mayor número de epochs permite que el modelo entrene más tiempo, pero también existe el riesgo de sobreajuste si entrenas durante demasiados epochs. Valores usados: **[20, 30, 40]**.

1) Configuraciones probadas que no funcionaron

De la mejor configuración para el entrenamiento de la sección anterior, tomamos ese modelo como base. Fue el punto de partida para aplicar **Transfer Learning** y no empezar de 0 sino que se toman los pesos de dicho modelo para afinarlos más a nuestro segundo conjunto de datos - dataset2 (descrito en la Sección 2)).

Para aplicar *Grid Search* se particiona el dataset2 de la siguiente manera:

- Entrenamiento: 302 (75 %)
- Test: 50 (25 %)

De todos los entrenamientos realizados (162 en total, con una duración de 2 horas por entrenamiento, dando un total de 13 días de duración aproximadamente) en base al método GridSearch (ejecutado sobre el servidor FC-UNI descrito en la Sección A) se recolectan las mejores combinaciones con menos error total (*total loss, descrito en la Sección* II.8):

TABLA XI: Entrenamiento usando Grid Search

Combinación	IOU_THRESH	INPUT_SIZE	BATCH_SIZE	LR_INIT	LR_END	WARMUP	EPOCHS	TOTAL LOSS
1 Comb.	0.5	416	4	0.0001	0.000001	2	40	0.27
2 Comb.	0.3	416	4	0.0001	0.000001	4	30	0.3
3 Comb.	0.3	416	4	0.0001	0.000001	6	20	0.31
4 Comb.(kmeans)	0.5	416	4	0.0001	0.000001	6	30	0.64
5 Comb.(kmeans)	0.5	416	8	0.0001	0.000001	6	40	0.64
Modelo Base	0.5	416	8	0.0001	0.000001	2	50	1.83

Por cada combinación encontrada se prueba con el dataset de Test:

TABLA XII: Pruebas sobre el dataset de Testeo

Combinación	Placas Totales	Placas Predichas	Porcentaje de Aciertos
1 Comb.	50	14	28 %
2 Comb.	50	20	40 %
3 Comb.	50	19	38 %
4 Comb.(kmeans)	50	16	32 %
5 Comb.(kmeans)	50	16	32 %
Modelo Base	50	27	**54 %**

Se observa claramente que el *Modelo Base* tiene el "Porcentaje de Aciertos" más alto entre los otros con Transfer Learning. Estos resultados tienen sentido dado que el procedimiento fue realizar un ajuste para que el modelo detecte mejor las placas peruanas; sin embargo, los datos recogidos y etiquetados no son suficientes para un entrenamiento que requiere una gran cantidad de imágenes. Actualmente no se tiene un dataset público de placas vehiculares peruanas, lo cual dificulta mucho realizar un ajuste sobre pocas imágenes. Dado que las versiones Tiny de Yolo son más rápidas pero pequeñas, entonces tienen menos capacidad para aprender detalles complejos y requieren de más datos para aprender más patrones y generalizar mejor. Aproximadamente se requiere un mínimo de 6000 muestras, análogo al dataset de imágenes de la primera iteración obtenida para el modelo base.

Por lo tanto, para las pruebas de las siguientes secciones se usará el **Modelo Base** (hiperparámetros de defecto) como modelo del primer módulo (detección de matrículas).

C Procesamiento de matrículas

El procesamiento de las matrículas fue elaborado con placas del Perú, obtenidas desde Internet a través del buscador web Google. En este procesamiento de matrículas, se muestran unos resultados bastante alentadores en el sentido de que nuestro sistema procesa correctamente la mayoría de las matrículas evaluadas.

Este módulo da unos buenos resultados relativos al proceso de segmentación de los caracteres en las matrículas analizadas. Los resultados se visualizan en las siguientes figuras.

FIGURA XXXV: Procesamiento de placa 1.

FIGURA XXXVI: Procesamiento de placa 2.

FIGURA XXXVII: Procesamiento de placa 3.

FIGURA XXXVIII: Procesamiento de placa 4.

FIGURA XXXIX: Procesamiento de placa 5.

FIGURA XL: Procesamiento de placa 6.

D Reconocimiento de caracteres

Esta parte muestra las pruebas realizadas a modelos para obtener el mejor de ellos en reconocimiento de letras mayúsculas y números. Como se mencionó en la Sección F fueron 3 modelos a evaluar; luego de comprobar la mayor precisión con el modelo *LeNet-5*, se evaluaron distintas variaciones para obtener la mejor configuración y modelo posible.

1) Data Augmentation

La técnica del Data Augmentation se usó porque si bien el detector puede capturar las placas de los coches, no está exento de tener distorsiones o ruidos en la placa como por ejemplo: detectar la placa girada, ocurrencia de manchas dentro de la placa, detectar una placa antigua o desgastada donde se percibe con dificultad los caracteres, entre otros. Para poder predecir correctamente los caracteres a pesar de las distorsiones mencionadas, una solución sería adaptar dichas distorsiones al entrenamiento de la NN y que aprenda

de ellas. Esta técnica permite incrementar la data con datos distorsionados para que la red se adapte a ellas.

Las distorsiones fueron hechas en el conjunto de entrenamiento y fueron las siguientes: rotaciones aleatorias de 20 grados como máximo (rotación horaria o antihoraria), desplazamiento de la imagen horizontal y vertical del 10 % del total de la imagen. Inicialmente se tenía 36787 imágenes, luego de aplicar la técnica Data Augmentation, fueron agregadas 16517 imágenes más las cuales representan el ruido generado, quedando finalmente 53304 imágenes. Del total de imágenes se agruparon el 80 % para el entrenamiento y el 20 % para el testeo. La Figura XLI muestra algunas imágenes que representan la imagen original y la distorsión generada.

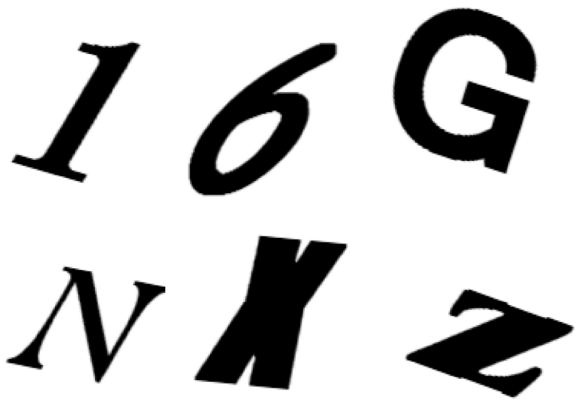


FIGURA XLI: Conjunto de imágenes producidas con la técnica Data Augmentation (rotaciones y desplazamientos).

2) Elección del mejor modelo

Los modelos escogidos para su evaluación fueron: (i) Un modelo creado desde cero; (ii) Un modelo basado en AlexNet; y (iii) El modelo Lenet-5. A continuación se detallan las arquitecturas de cada uno de ellos:

Modelo creado desde cero

Este modelo (a partir de ahora llamado **0NN**) tiene 19 capas en total, compuestas por 4 capas convolucionales, 2 capas MaxPooling, 5 capas BatchNormalization, 2 capas Dropout, 1 capa Flatten, 3 capas Dense y una capa de salida. Este modelo fue elaborado con la idea de crear varias capas regularizadoras que previnieran el overfitting. Sin embargo, un modelo complejo no siempre resulta en el mejor para esta tarea clasificadora de letras y números, como se demostrará en las gráficas. La arquitectura completa y los parámetros en total se muestran en la siguiente Figura XLII.

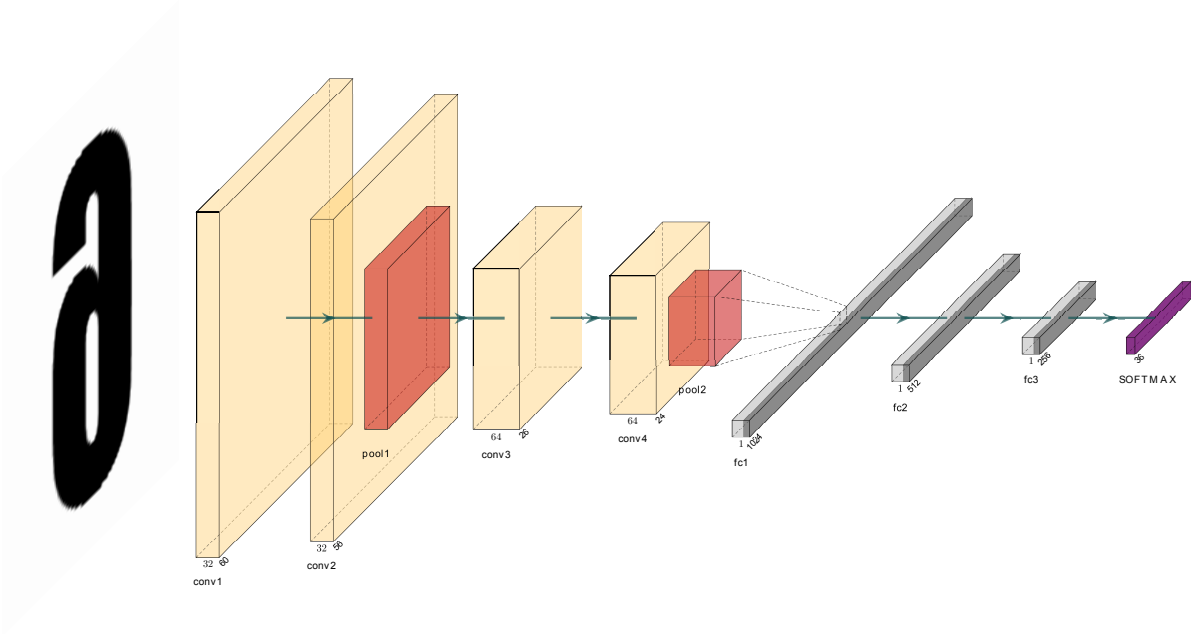
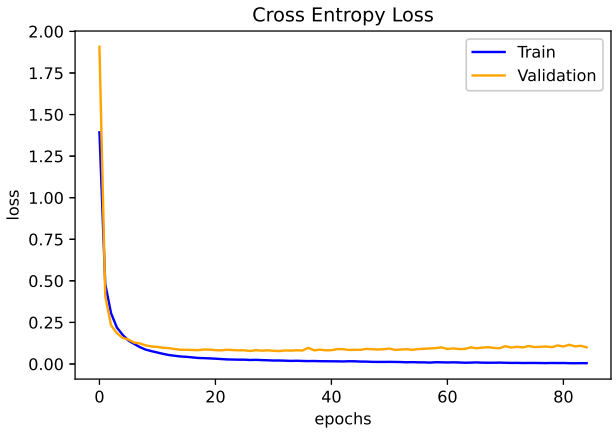
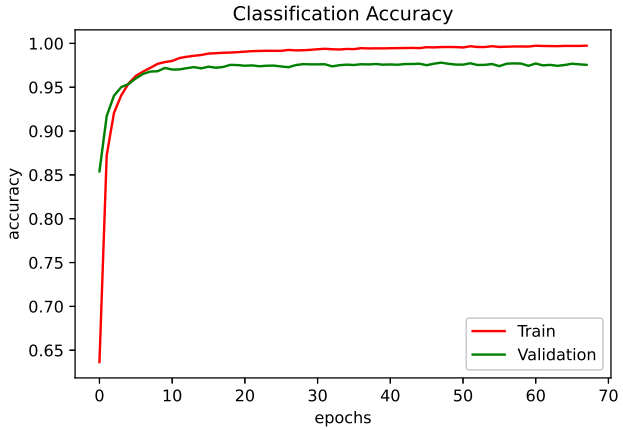


FIGURA XLII: Arquitectura del modelo 0NN.

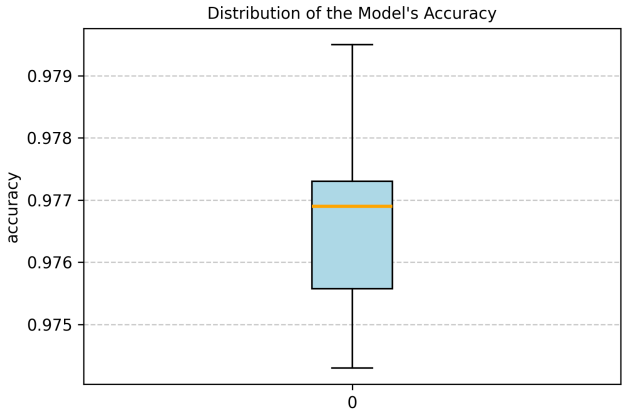
El entrenamiento de esta red neuronal mejora en cada época, posee una buena curva de aprendizaje, lo que se visualiza en Figura XLIIIa y Figura XLIIIb, las cuales muestran la función de costo y precisión, respectivamente. Y los resultados alcanzados (*precisiones*) en cada entrenamiento se visualizan y promedian en la Figura XLIIIc en donde el valor promedio es de 0.977 aproximadamente.



(A) Evaluación de Función de Costo. Azul: Entrenamiento. Amarillo: Validación.



(B) Evaluación de la Precisión. Rojo: Entrenamiento. Verde: Validación.



(C) Rango y media aritmética de las precisiones obtenidas.

FIGURA XLIII: Entrenamiento del modelo 0NN.

Modelo pequeño de AlexNet

Este modelo (a partir de ahora llamado **MinAlexNet**) tiene como base al modelo AlexNet, pero se omitieron algunas capas y se redujeron algunos parámetros. Estas reducciones fueron debido a que en las primeras semanas no se tenía el clúster para el entrenamiento, sino solamente la Laptop *Aspire 3*, es decir, fue arreglada para que pudiera ser entrenada con una VRAM de $2Gb$. El modelo consta de 3 capas convolucionales, 2 capas MaxPooling, 1 capa Flatten, 1 capa Dense y una capa *output* (salida). La arquitectura se visualiza en la Figura XLIV. Las gráficas muestran que tanto el **cost function** como **accuracy** dan excelentes resultados, bien en la etapa de entrenamiento y de validación. No se considera que incurra en *overfiting* dado que la fase de validación no varía mucho o se aleja del resultado del entrenamiento.

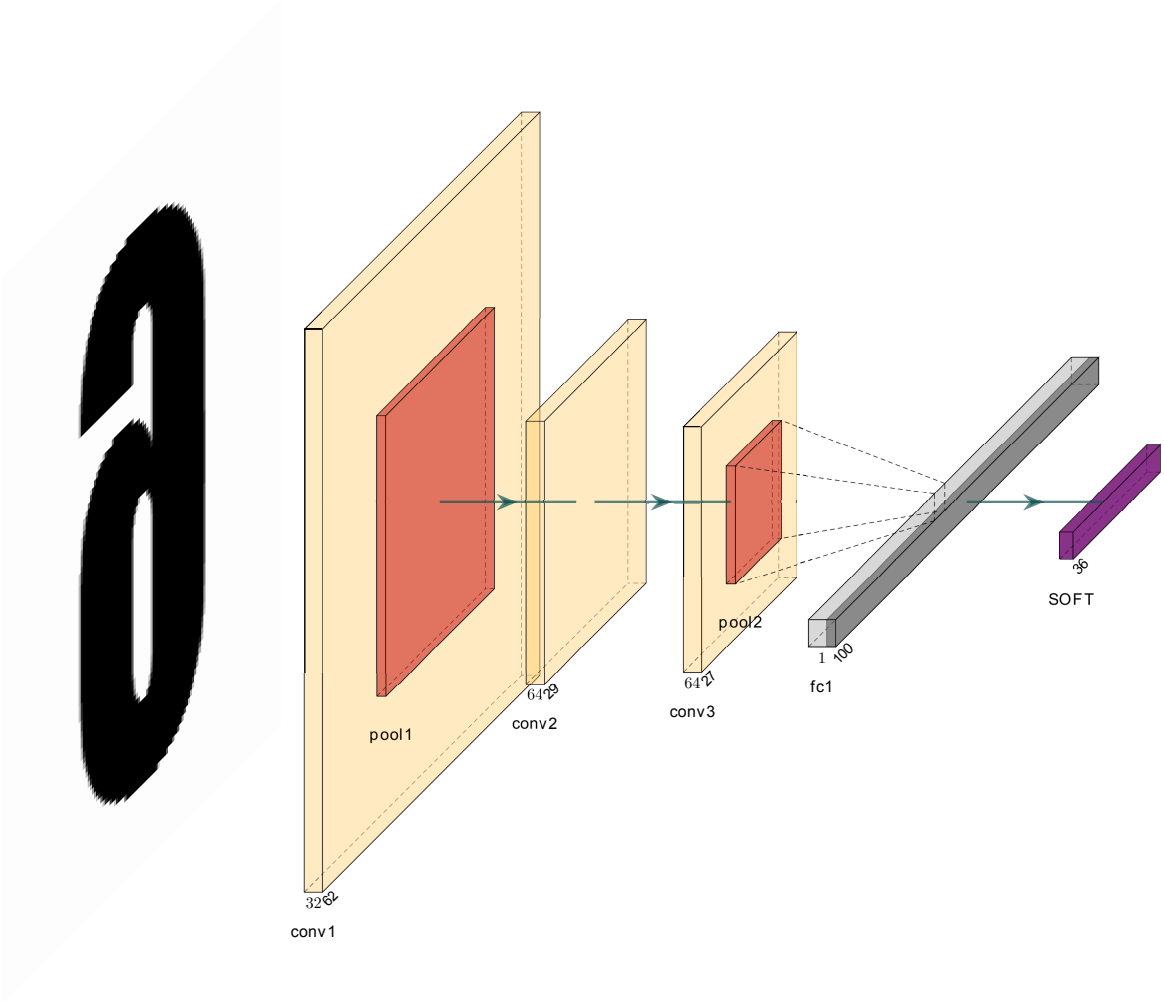
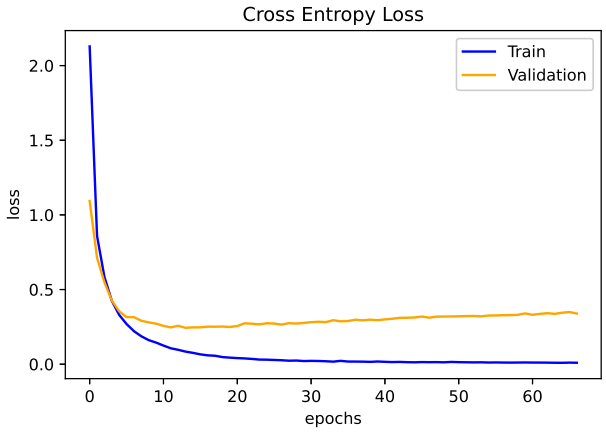
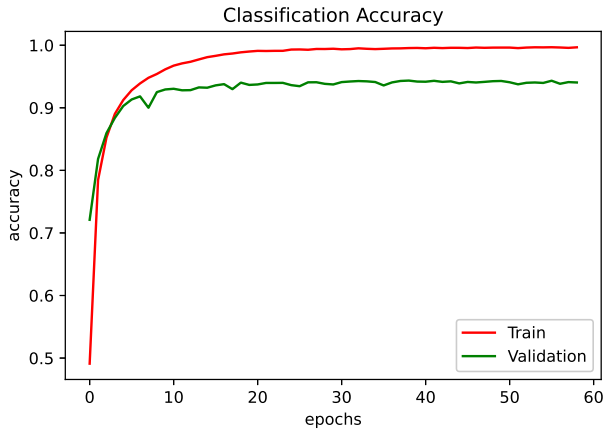


FIGURA XLIV: Arquitectura del modelo MinAlexNet.

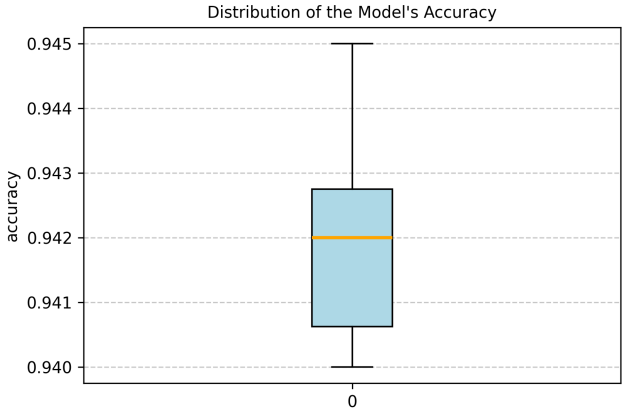
Este modelo tiene una buena curva de aprendizaje, sin embargo, en la parte de validación muestra peores resultados por una mínima diferencia la cual se visualiza en la Figura XLVa y Figura XLVb. Y los resultados obtenidos(precisiones) después de 5 entrenamientos dan un promedio aproximado de 0.942 tal como lo muestra la Figura XLVc.



(A) Evaluación de Función de Costo. Azul: Entrenamiento. Amarillo: Validación.



(B) Evaluación de la Precisión. Rojo: Entrenamiento. Verde: Validación.



(C) Rango y media aritmética de las precisiones obtenidas.

FIGURA XLV: Entrenamiento del modelo MinAlexNet.

Modelo LeNet-5

Este modelo (a partir de ahora llamado **AugLeNet**) fue escogido debido a que, en sus inicios, fue usado para el reconocimiento de dígitos. Para el trabajo desarrollado se pretende incrementar algunos parámetros para que pueda reconocer 36 clases (*26 letras mayúsculas y 10 dígitos*). El modelo tiene 11 capas en total: 3 capas convolucionales, 3 capas MaxPooling, 1 capa Flatten, 1 capa Dropout, 1 capa BatchNormalization, 1 capa Dense y una capa output. La arquitectura se visualiza en la Figura XLVI.

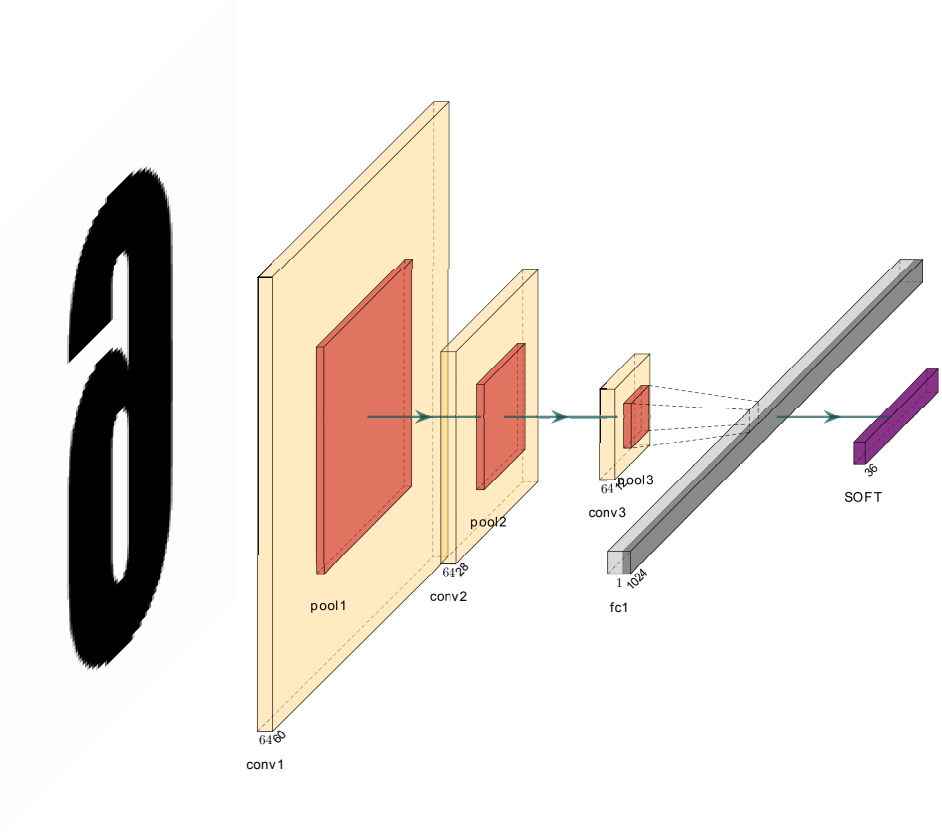
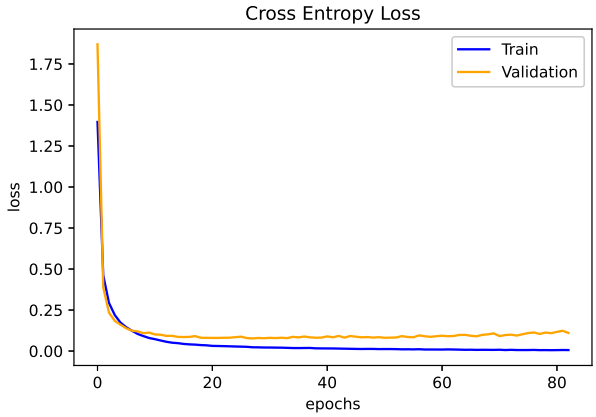


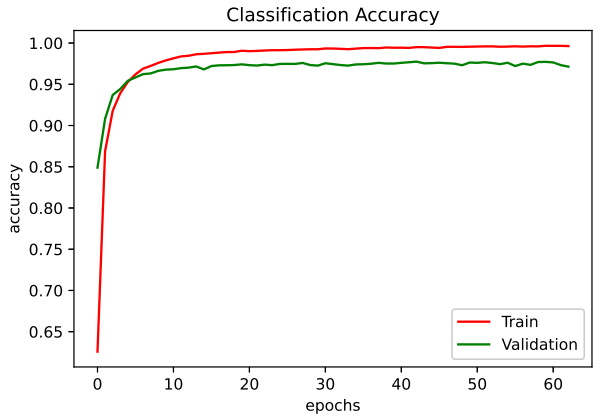
FIGURA XLVI: Arquitectura del modelo AugLeNet.

Este modelo es quien tiene muy buenos resultados tanto en la etapa de entrenamiento como en la validación. Posee una buena curva de aprendizaje que se puede visualizar en la Figura XLVIIa, y la función de precisión en la Figura XLVIIb. Y el resultado obtenido en el promedio de los 5 entrenamientos da una precisión aproximada de 0.977,

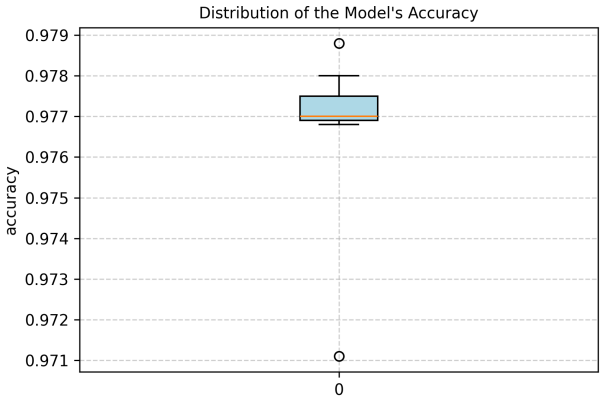
que se visualiza en la Figura XLVIIc.



(A) Evaluación de Función de Costo. Azul: Entrenamiento. Amarillo: Validación.



(B) Evaluación de la Precisión. Rojo: Entrenamiento. Verde: Validación.



(C) Rango y media aritmética de las precisiones obtenidas.

FIGURA XLVII: Entrenamiento del modelo AugLeNet.

Se concluye que los resultados obtenidos de los 3 modelos mencionados son similares, se distinguen por unas cuantas unidades en su precisión y función de costo. La Figura XLIII, Figura XLV y la Figura XLVII muestran la evaluación en cada modelo (0NN, MinAlexet y AugLeNet, respectivamente) por medio del *cross validation* con 5-*folds*. El término *cross validation* no es más que evaluar un modelo n veces (llamado *folds*) con distintas particiones del dataset y de sus resultados, hallar su media aritmética. Los gráficos de sección **A** y **B** muestran la función de coste (*loss*) y la precisión (*Accuracy*) a lo largo del entrenamiento y los gráficos de la sección **C** el rango de las precisiones obtenidas en cada fold.

3) Comparativa de la complejidad de los modelos evaluados

La Tabla XIII presenta una comparación de la complejidad de los modelos evaluados, medida en términos del número de parámetros y la cantidad de FLOPs. Se puede observar que un mayor número de parámetros o una mayor cantidad de operaciones no garantiza un mejor $accuracy$ en nuestro caso específico de clasificar caracteres.

TABLA XIII: Comparación de complejidad de modelos

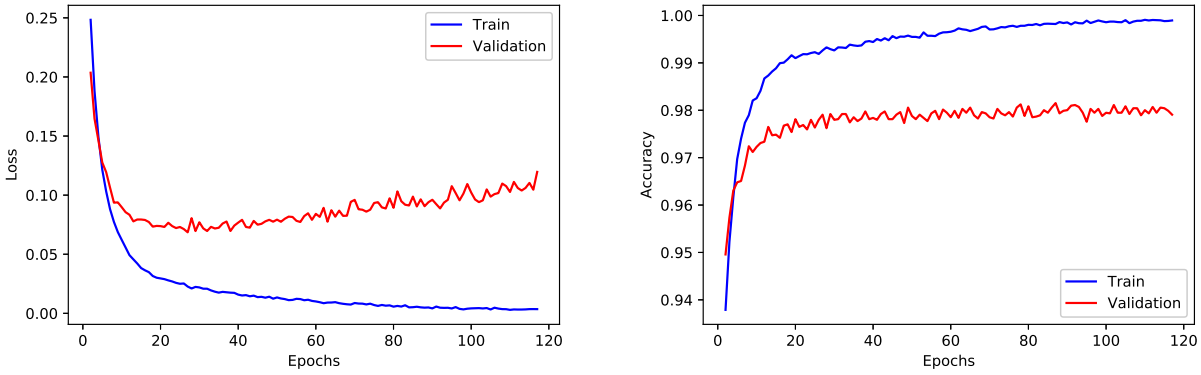
Modelo	N° params	FLOPs
0NN	$10,191,140$	$254.5M$
MinAlexnet	$1,141,080$	$89.5M$
AugLenet	$2,475,492$	$80.1M$

4) Variaciones al mejor modelo

Los resultados muestran que el mejor modelo es *AugLeNet*. Por tanto, este trabajo se analizará en base a este modelo con diversas modificaciones para optimizar un poco más la métrica ya obtenida. Algunas de estas modificaciones fueron:

- La capa de entrada (*Input Layer*) fue analizada en 2 entrenamientos, la primera recibió imágenes de `64x64` y la segunda imágenes de `128x128`. La función de costo y la precisión fueron registradas en fase de entrenamiento y se muestran en

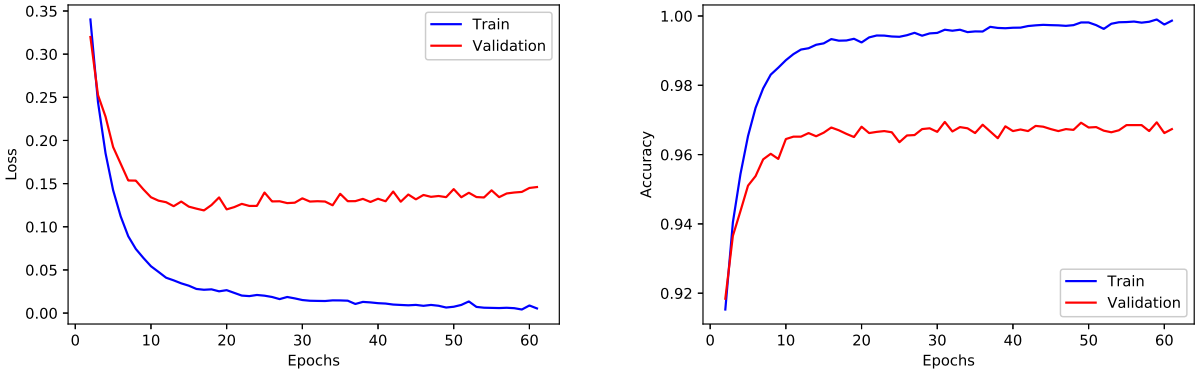
la Figura XLVIII y Figura XLIX. Se visualiza en las entradas de 64x64 se obtiene una mejor presición ($98\,\%$) que en las entradas de 128x128 en donde se obtuvo un $97\,\%$. Si bien las curvas de entrenamiento y validación son parecidas y estan cercas, las gráficas dan señales de un posible **overfitting** debido a que hay una variación constante entre las 2 curvas. El caso más notorio es en la imagen (A) de la Figura XLVIII donde la curvatura de validación en la función de costo esta en un ascenso constante dando señal de **overfitting**, por este motivo el modelo mencionado no resulta mejorar sino empeorar los resultados. Por lo tanto se descarta su uso en el sistema propuesto.



(A) Función de Costo (*Loss function*).

(B) Precisión obtenida (*Accuracy*).

FIGURA XLVIII: Entrenamiento del modelo AugLeNet con imágenes de 64x64 de entrada.

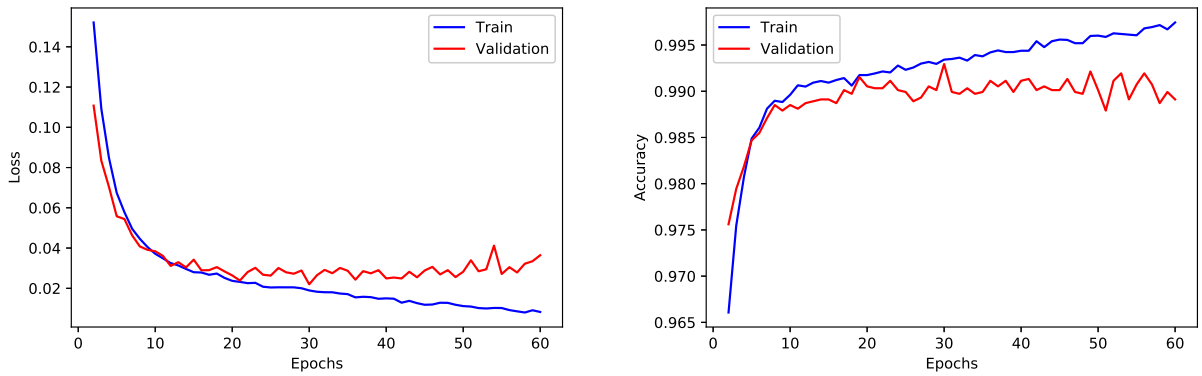


(A) Función de Costo (*Loss Function*).

(B) Precisión obtenida (*Accuracy*).

FIGURA XLIX: Entrenamiento del modelo AugLeNet con imágenes de 128x128 de entrada.

- Además, se ha realizado una segunda limpieza de los datos con el fin de asemejar más las imágenes del dataset de entrenamiento con los caracteres que aparecen en las matrículas. Si bien la precisión es sorprendente dado que alcanza un $99\,\%$, no es precisa ante los datos de testeo, es decir, es bien preciso para los datos parecidos al del entrenamiento y no ante caracteres con pequeños obstáculos. Esto muestra la importancia de los datos que se van a entrenar, éstos tienen que cubrir un contexto amplio y no uno corto porque tendríamos estos resultados engañosos que cubren un pequeño contexto. La Figura L enseña la función de costo y la precisión alcanzada con la segunda limpieza de los datos e imágenes de entrada de 64x64.



(A) Función de Costo (*Loss Function*).

(B) Precisión obtenida (*Accuracy*).

FIGURA L: Entrenamiento del modelo AugLeNet con imágenes de $64\text{x}64$ y segunda limpieza de los datos.

- Por otro lado, se realizaron distintas variaciones a las capas convolucionales. Las tres capas convolucionales originales tienes $64, 64, 64$ filtros y las variaciones van en torno a modificaciones a éstas 3 o eliminando 1 capa. Se entrenaron con capas convolucionales con $(32, 64, 128)$; $(16, 32, 64)$; $(32, 64, 64)$; $(32, 32, 64)$; $(64, 128, 256)$; $(64, 128)$; $(32, 64)$ y $(64, 64)$ filtros. Los resultados en cada modelo no superan la precisión del modelo original por lo cual se descartan, pero se recogen observaciones como las siguientes: **(i)** Si se elimina una capa convolucional, la precisión baja; y **(ii)** Si se empieza con una capa convolucional con 32 filtros, la precisión baja, con lo cual es mejor empezar con 64 filtros.

E Predicciones sobre imágenes y videos reales

Las pruebas reales del sistema propuesto recaen sobre imágenes, videos y grabaciones en vivo. Los resultados demostrarán entonces la eficacia y desempeño alcanzados del trabajo realizado. Para poner a prueba y ver las fallas y mejoras que necesita el sistema, es necesaria una variedad de imágenes y grabaciones de vehículos sobre las calles, jirones o avenidas de Lima, con la finalidad de observar hasta qué punto el sistema tiene una exactitud aceptable.

1) Predicciones sobre imágenes - Primera Prueba

En esta sección se predicen imágenes de automóviles con el sistema propuesto sobre vehículos con placas vehiculares diferentes a la placa del Perú (a excepción de una placa). Es decir, analizamos qué tan bien es nuestro sistema con placas de diferentes países. Estas pruebas entonces pasan por las 3 etapas mencionadas: en la primera se detecta la matrícula, en la segunda se procesa dicha matrícula capturada y, por último, se reconocen cada carácter de la matrícula. Las imágenes analizadas para esta prueba se encuentran descritas en la Figura LI y han sido obtenidas de la internet o tomadas con un celular con el debido consentimiento del dueño del auto. La condición de las imágenes es fundamental para así tener la mayor nitidez de los caracteres de la matrícula. A mayor nitidez, obviamente, mejores predicciones. Por tal motivo, la importancia de tener un recurso como una cámara de alta definición para el mejor provecho del sistema propuesto. La Tabla XIV muestra las matrículas predichas por el sistema en cada una de las imágenes.

TABLA XIV: Resultados sobre las imágenes de prueba.

Imagen	**Placa Real**	**Placa Predicha**	**Resultado**
Imagen1	4770LCJ	L770LCJ	*FP*
Imagen2	AA990GA	AA990GA	*TP*
Imagen3	C3H173	C3H173	*TP*
Imagen4	7ATS062	7AJSD62	*FP*
Imagen5	A5U018	R9018	*Negative*
Imagen6	BBU114	BU114	*Negative*

Los resultados no son alentadores, pero se entiende dado que el sistema no está pensado a predecir sobre placas que no sean del Perú. Según los datos recogidos y aplicando el criterio de evaluación de la fórmula IV.2 y IV.1 se tiene un **accuracy de** $33\,\%$ y **precision de** $50\,\%$.

FIGURA LI: Imágenes de prueba utilizadas para las predicciones.

2) Predicciones sobre imágenes - Segunda Prueba

En esta segunda parte se tiene un dataset más apropiado para medir la exactitud del sistema propuesto. Las imágenes fueron tomadas a los vehículos dentro de un taller de mecánica que estaban siendo trabajados y por medio de un celular de baja resolución fueron capturadas. Con una mayor cantidad de datos de prueba, se puede obtener la eficacia del sistema. Hay imágenes en donde se observan autos desgastados y, por ende, las placas no son tan nítidas o presentan distorsión y evitan un adecuado reconocimiento de éstos. Algunas de estas imágenes se encuentran en la Figura LII y la Tabla XV indica las predicciones y las placas verdaderas de cada automóvil fotografiado.

FIGURA LII: Imágenes de prueba para las predicciones del sistema.

TABLA XV: Resultados sobre las imágenes de prueba - Segunda prueba.

Placa Real	Placa Predicha	Resultado	Placa Real	Placa Predicha	Resultado
D90016	No Placa	*Negative*	D90016(t)	D90016	*TP*
F5P608	No Placa	*Negative*	F2C261	F2C261	*TP*
P2D074(t)	No Placa	*Negative*	COZ608	No Placa	*Negative*
A7Q866	A7Q866	*TP*	AFV265	AFV265	*TP*
AFV265(t)	No Placa	*Negative*	F9G343	F9G343	*TP*
F2N441	F2N441	*TP*	C1L078	No Placa	*Negative*
APY301	APV301	*FP*	APY301(t)	APV3D1	*FP*
D5K394	No Placa	*Negative*	B4S414	B4S414	*TP*
DOW353	No Placa	*Negative*	D5Z111	D5Z111	*TP*
ARL673	WARL673	*FP*	BGJ241	No Placa	*Negative*
F4D005(t)	F4D005	*TP*	D1N611	No Placa	*Negative*
C1T324	C1T324	*TP*	F8X243	F8X243	*TP*
D6B584	D6B584	*TP*	AYF704	No Placa	*Negative*
A9K492	No Placa	*Negative*	D9E444	Z9E444	*FP*
C6U222	C6U222	*TP*	D7N520	D7N520	*TP*
D2A065	D2AD65	*FP*	B6D657	B6DB57	*TP*
C5R175	U5R175	*FP*	BER687	BER687	*TP*
AFR589	AFR589	*TP*	F5R564(t)	No Placa	*Negative*
BNI677	BNI677	*TP*	BNI677(t)	BNI677	*TP*
ACO185	No Placa	*Negative*	BOL595	No Placa	*Negative*
ANW537	ANH537	*FP*	ARO367	ARO367	*TP*
AYK270	AVK27D	*FP*	F5P608	F5P608	*TP*
F4Q132	F4Q132	*TP*	BCP647	BCP647	*TP*
D2P549	D2P549	*TP*	D7E598	No Placa	*Negative*
A2Q341	A2Q341	*TP*	F8H305	F8H305	*TP*
F0O078	No Placa	*Negative*	AJJ666	WJJ666	*FP*
D7C368	No Placa	*Negative*	F4E042	No Placa	*Negative*
A0T114	A0T114	*TP*			

Los resultados de esta prueba y con referencia a la Tabla XV muestran que 36 de 55 imágenes fueron detectadas, esto nos da un $65.5\,\%$ de recall del sistema detectando las placas (muy baja debido a la calidad de imágenes). Dentro de las 36 placas detectadas, existen 9 placas que no fueron correctas, esto debido a una predicción errónea en algún carácter. **La exactitud fue de** $49\,\%$ **y la precisión de** $75\,\%$. Si las imágenes hubieran sido tomadas desde otro ángulo o no tan cerca, ni tan lejos, o con una cámara de mejor calidad

a la usada, se hubieran tenido otros resultados más favorables. Esto refleja la complejidad de las pruebas que requieren de dispositivos adecuados (como se verá en las siguientes pruebas con una cámara 4K especializada).

3) Predicciones sobre videos públicos (YouTube) y privados

Las pruebas sobre videos muestran un **valor máximo de** 40 **FPS y un mínimo de** 30 de FPS en cuanto a rapidez. Fueron evaluados varios videos entre los cuales hay grabaciones que son en calles de Estados Unidos, las cuales fueron útiles debido a las cámaras 4K utilizadas para observar el tráfico por diferentes calles, pero con la consideración de que son placas extranjeras. Estos videos fueron recogidos del canal de *YouTube* de *J Utah* en la ciudad de Beverly Hills [1]. Otro video fue una grabación con la cámara Amcrest referida en la Sección A en el interior de la puerta N° 5 de la UNI (se dará más detalle en la siguiente sección). Estos videos están disponibles en el siguiente link VIDEOS DE PRUEBA. Las grabaciones muestran el rendimiento del sistema en diferentes tipos de escenarios, las cuales son o bien sacadas de la internet o por cuenta propia. Debido a la procedencia extranjera de las grabaciones recogidas en estas primeras pruebas con videos, no se evaluarán según las reglas de evaluación expuestas en el Capítulo 4. Para esto, nos centraremos en las siguientes secciones.

F Predicciones en el campus de la UNI (Puerta 5)

El escenario para realizar las pruebas del sistema propuesto en grabaciones y en tiempo real fue la Puerta 5 de la UNI, dado que es el único lugar de entrada al campus y necesariamente todos los vehículos pasan por allí. Las siguientes secciones muestran los datos de varias pruebas ejecutadas.

1) Predicciones de grabaciones

En pruebas iniciales se dejó que la cámara grabara en el ambiente propuesto. Estas grabaciones estaban siendo almacenadas en la memoria externa de la cámara. Luego de terminar las grabaciones, se procedía a procesar el video con el sistema. Tener en cuenta lo siguiente:

- Cada componente es independiente uno del otro, a excepción del componente **Sistema integral** que constituye todo el sistema y calcula su valor en base al criterio mencionado anteriormente (fórmula IV.1).
- El *Detector de objetos* es el módulo en el cual el sistema es apto para detectar correctamente todas las placas de las grabaciones. Se calcula en base a la métrica IV.3.
- El valor de precisión para el *Clasificador de caracteres* es qué tan preciso fue el sistema en clasificar un carácter dado que el caracter fue capturado correctamente y enviado para su clasificación.
- El módulo de *Preprocesamiento de imagen* es el que detectado la placa, segmenta correctamente los caracteres para que sea enviado al clasificador de caracteres. Los parámetros *SCORE THRESHOLD* y *IOU THRESHOLD* son relevantes para determinar qué tan preciso es el *bounding box* al enmarcar la placa predicha por el detector. El *SCORE THRESHOLD* determina qué tan preciso debe estar el detector de que en cierta región hay una placa. Y el *IOU THRESHOLD* es un umbral en el cual se acepta que la región predicha esté próxima a la placa real. La precisión de este componente recae en si logra segmentar todos los caracteres de la placa.

El siguiente cuadro muestra el resumen de los resultados obtenidos:

TABLA RESUMEN					
ID	**Componente**	**Modelo o Recurso**	**Modelo de Base de Datos**	**Descripción**	**Precisión**
1	Detector de objetos	yolov4 tiny	Pesos entrenados con COCO Dataset + OpenImage[6531 imágenes](Incluido placas de Puerta N°5)	Se realizó transfer learning para la predicción de placas	$100\,\%$
	Clasificador de caracteres	AugLeNet	chars74k[38758 imágenes] + caracteres Puerta N° 5	Pesos entrenados desde 0 para los caracteres	$94,73\,\%$
	Preprocesamiento de imagen	OpenCV		Hiperparámetros (SCORE THRESHOLD:$0,5$ y IOU THRESHOLD:$0,2$)	$65,4\,\%$
	Sistema Integral				$71,88\,\%$
2	Detector de objetos	yolov3 tiny	Pesos entrenados con COCO Dataset + OpenImage[6531 imágenes](Incluido placas de Puerta N°5)	Se realizó transfer learning para la predicción de placas	$97,96\,\%$
	Clasificador de caracteres	AugLeNet	chars74k[38758 imágenes] + Caracteres Puerta N° 5	Pesos entrenados desde 0 para los caracteres	$93,61\,\%$
	Preprocesamiento de imagen	OpenCV		Hiperparámetros (SCORE THRESHOLD:$0,6$ y IOU THRESHOLD:$0,2$)	$58,2\,\%$
	Sistema Integral				$72,41\,\%$
3	Detector de objetos	yolov4 tiny	Pesos entrenados con COCO Dataset + OpenImage[6481 imágenes](Placas)	Se realizó transfer learning para la predicción de placas	$100\,\%$
	Clasificador de caracteres	AugLeNet	chars74k[38758 imágenes]	Pesos entrenados desde 0 para los caracteres	$92,58\,\%$
	Preprocesamiento de imagen	OpenCV		Hiperparámetros (SCORE THRESHOLD:$0,6$ y IOU THRESHOLD:$0,2$)	$68,43\,\%$
	Sistema Integral				$70,59\,\%$

4	Detector de objetos	yolov3 tiny	Pesos entrenados con COCO Dataset + OpenImage[6481 imágenes](Placas)	Se realizó transfer learning para la predicción de placas	$97,96\,\%$
	Clasificador de caracteres	AugLeNet	chars74k[38758 imágenes]	Pesos entrenados desde 0 para los caracteres	$93,61\,\%$
	Preprocesamiento de imagen	OpenCV		Hiperparámetros (SCORE THRESHOLD:$0,6$ y IOU THRESHOLD:$0,2$)	$58,17\,\%$
	Sistema Integral				$72,41\,\%$
5	Detector de objetos	yolov4 tiny	Pesos entrenados con COCO Dataset + OpenImage[6481 imágenes](Placas)	Se realizó transfer learning para la predicción de placas	$100\,\%$
	Clasificador de caracteres	AugLeNet	chars74k[38758 imágenes] + Caracteres Puerta N° 5	Pesos entrenados desde 0 para los caracteres	$90,67\,\%$
	Preprocesamiento de imagen	OpenCV		Hiperparámetros (SCORE THRESHOLD:$0,6$ y IOU THRESHOLD:$0,2$)	$56,07\,\%$
	Sistema Integral				$59,30\,\%$
6	Detector de objetos	yolov4 tiny	Pesos entrenados con COCO Dataset + OpenImage[6481 imágenes](Placas)	Se realizó transfer learning para la predicción de placas	$100\,\%$
	Clasificador de caracteres	AugLeNet	chars74k[38758 imágenes]	Pesos entrenados desde 0 para los caracteres	$93,08\,\%$
	Preprocesamiento de imagen	OpenCV		Hiperparámetros (SCORE THRESHOLD:$0,6$ y IOU THRESHOLD:$0,3$)	$68,43\,\%$
	Sistema Integral				$73,53\,\%$

Asimismo, por cada Sistema se tienen las siguientes tablas que fueron las placas capturadas por la cámara con su respectiva precisión (fueron omitidas las placas no detectadas o que no fueron segmentadas correctamente en la mayoría de los caracteres para obtener la métrica requerida):

Sistema ID - 1:

TABLA XVI: Resultados de pruebas hechas en la puerta N°5 de la UNI . ID-1

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
BAN382	BN382	83 %	0 %
C9B132	C9B13	83 %	0 %
A1G433	A1G433	100 %	100 %
AXF501	AXF501	100 %	100 %
C3P097	C3P097	100 %	100 %
BPQ361	BPQ361	100 %	100 %
BNZ426	BNZ426	100 %	100 %
B9C617	B9C617	100 %	100 %
BBR583	BBR583	100 %	100 %
AXX850	AXX850	100 %	100 %
B3S343	B3S343	100 %	100 %
BND247	BND247	100 %	100 %
BZN277	BZN277	100 %	100 %
D1X352	D9X35	67 %	0 %
AWH723	AHH723	83 %	0 %
APN340	APN340	100 %	100 %
AAV044	AAV044	100 %	100 %
D9X600	D9X600	100 %	100 %
BYW117	BYH117	83 %	0 %
D7H425	D7H425	100 %	100 %
D9Q573	D9Q573	100 %	100 %
A7U451	A7U451	100 %	100 %
B3S343	B3S34	83 %	0 %
BHG424	BHG424	100 %	100 %
BME689	BME689	100 %	100 %
AXM683	AXM683	100 %	100 %
CCH149	CCH149	100 %	100 %
BIO209	BID2D9	67 %	0 %
AFZ912	AFZ912	100 %	100 %
CBF408	CBF408	100 %	100 %
ATV488	ATV488	100 %	100 %
AWX178	AHX178	83 %	0 %
Promedio Total		94, 73 %	71, 88 %

Sistema ID - 2:

TABLA XVII: Resultados de pruebas hechas en la puerta N°5 de la UNI . ID-2

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
C9B132	C9B13	83 %	0 %
AXF501	AXF501	100 %	100 %
C3P097	C3PD97	83 %	0 %
BPQ361	BPQ361	100 %	100 %
BNZ426	BNZ426	100 %	100 %
D1S043	91SN43	67 %	0 %
B9C617	B9C617	100 %	100 %
BBR583	BBR583	100 %	100 %
AXX850	AXX850	100 %	100 %
B3S343	B3S343	100 %	100 %
BND247	BND247	100 %	100 %
BZN277	BZN277	100 %	100 %
D1X352	IJX35	50 %	0 %
AWH723	AWH723	100 %	100 %
APN340	APN340	100 %	100 %
AAV044	AAV044	100 %	100 %
D9X600	D9X600	100 %	100 %
BYW117	BYH117	83 %	0 %
D7H425	D7H425	100 %	100 %
D9Q573	D9Q573	100 %	100 %
A7U451	A7U451	100 %	100 %
B3S343	B3S34	83 %	0 %
BHG424	BHG424	100 %	100 %
BME689	BME689	100 %	100 %
AXM683	AXM683	100 %	100 %
CCH149	CCH149	100 %	100 %
CBF408	CBF4D8	83 %	0 %
ATV488	ATV488	100 %	100 %
AWX178	AHX178	83 %	0 %
Promedio Total		93, 61 %	72, 41 %

Sistema ID - 3:

TABLA XVIII: Resultados de pruebas hechas en la puerta N°5 de la UNI . ID-3

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
BAN382	BN382	83 %	0 %
C9B132	C9B13	83 %	0 %
A1G433	A1G433	100 %	100 %
AXF501	AXF501	100 %	100 %
C3P097	C3P097	100 %	100 %
BPQ361	BPQ361	100 %	100 %
BNZ426	BNZ426	100 %	100 %
B9C617	B9C617	100 %	100 %
BBR583	BBR583	100 %	100 %
AXX850	AXX850	100 %	100 %
B3S343	B3S343	100 %	100 %
BND247	BND247	100 %	100 %
BZN277	BZN277	100 %	100 %
D1X352	D9X35	67 %	0 %
AWH723	AHH723	83 %	0 %
APN340	APN340	100 %	100 %
AAV044	AAV044	100 %	100 %
D9X600	D9X600	100 %	100 %
BYW117	BYH117	83 %	0 %
BOG708	RQG7N	33 %	0 %
D7H425	D7H425	100 %	100 %
D9Q573	D9Q573	100 %	100 %
D5I612	DSI612	83 %	0 %
A7U451	A7U451	100 %	100 %
B3S343	B3S34	83 %	0 %
BHG424	BHG424	100 %	100 %
BME689	BME689	100 %	100 %
AXM683	AXM683	100 %	100 %
CCH149	CCH149	100 %	100 %
BIO209	BID2D9	67 %	0 %
AFZ912	AFZ912	100 %	100 %
CBF408	CBF408	100 %	100 %
ATV488	ATV488	100 %	100 %
AWX178	AHX178	83 %	0 %
Promedio Total		92, 58 %	70, 59 %

Sistema ID - 4:

TABLA XIX: Resultados de pruebas hechas en la puerta N°5 de la UNI . ID-4

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
C9B132	C9B13	83 %	0 %
AXF501	AXF501	100 %	100 %
C3P097	C3PD97	83 %	0 %
BPQ361	BPQ361	100 %	100 %
BNZ426	BNZ426	100 %	100 %
D1S043	91SN43	67 %	0 %
B9C617	B9C617	100 %	100 %
BBR583	BBR583	100 %	100 %
AXX850	AXX850	100 %	100 %
B3S343	B3S343	100 %	100 %
BND247	BND247	100 %	100 %
BZN277	BZN277	100 %	100 %
D1X352	IJX35	50 %	0 %
AWH723	AHH723	83 %	0 %
APN340	APN340	100 %	100 %
AAV044	AAV044	100 %	100 %
D9X600	D9X600	100 %	100 %
BYW117	BYH117	83 %	0 %
D7H425	D7H425	100 %	100 %
D9Q573	D9Q573	100 %	100 %
A7U451	A7U451	100 %	100 %
B3S343	B3S34	83 %	0 %
BHG424	BHG424	100 %	100 %
BME689	BME689	100 %	100 %
AXM683	AXM683	100 %	100 %
CCH149	CCH149	100 %	100 %
CBF408	CBF408	100 %	100 %
ATV488	ATV488	100 %	100 %
AWX178	AHX178	83 %	0 %
Promedio Total		93, 61 %	72, 41 %

Sistema ID - 5:

TABLA XX: Resultados de pruebas hechas en la puerta N°5 de la UNI. ID-5

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
BAN382	BN382	83 %	0 %
C9B132	C9B13	83 %	0 %
A1G433	1G433	83 %	0 %
AXF501	AXF511	83 %	0 %
C3P097	C3P097	100 %	100 %
BPQ361	BPQ361	100 %	100 %
BNZ426	BNZ426	100 %	100 %
B9C617	B9C617	100 %	100 %
BBR583	BBR583	100 %	100 %
B3S343	B3S343	100 %	100 %
BND247	BND247	100 %	100 %
BZN277	BZN277	100 %	100 %
D1X352	D9X35	67 %	0 %
AWH723	IIH723	67 %	0 %
APN340	1PN340	83 %	0 %
D9X600	D9X6D0	83 %	0 %
BYW117	BYV117	83 %	0 %
D7H425	D7H425	100 %	100 %
D9Q573	D9Q573	100 %	100 %
A7U451	A7U451	100 %	100 %
B3S343	B3S34	83 %	0 %
BHG424	BHG424	100 %	100 %
BME689	BME689	100 %	100 %
AXM683	AXM683	100 %	100 %
CCH149	CCH149	100 %	100 %
CBF408	CBF408	100 %	100 %
AWX178	IVX17B	50 %	0 %
Promedio Total		90, 67 %	59, 26 %

Sistema ID - 6:

TABLA XXI: Resultados de pruebas hechas en la puerta N°5 de la UNI . ID-6

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
BAN382	BAN38	83 %	0 %
C9B132	C9B13	83 %	0 %
A1G433	A1G433	100 %	100 %
AXF501	AXF501	100 %	100 %
C3P097	C3P097	100 %	100 %
BPQ361	BPQ361	100 %	100 %
BNZ426	BNZ426	100 %	100 %
B9C617	B9C617	100 %	100 %
BBR583	BBR583	100 %	100 %
AXX850	AXX850	100 %	100 %
B3S343	B3S343	100 %	100 %
BND247	BND247	100 %	100 %
BZN277	BZN277	100 %	100 %
D1X352	D9X35	67 %	0 %
AWH723	AHH723	83 %	0 %
APN340	APN340	100 %	100 %
AAV044	AAV044	100 %	100 %
D9X600	D9X600	100 %	100 %
BYW117	BYH117	83 %	0 %
BOG708	RQG7N	33 %	0 %
D7H425	D7H425	100 %	100 %
D9Q573	D9Q573	100 %	100 %
D5I612	DSI612	83 %	0 %
A7U451	A7U451	100 %	100 %
B3S343	B3S34	83 %	0 %
BHG424	BHG424	100 %	100 %
BME689	BME689	100 %	100 %
AXM683	AXM683	100 %	100 %
CCH149	CCH149	100 %	100 %
BIO209	BID2D9	67 %	0 %
AFZ912	AFZ912	100 %	100 %
CBF408	CBF408	100 %	100 %
ATV488	ATV488	100 %	100 %
AWX178	AWX178	100 %	100 %
Promedio Total		93, 08 %	73, 53 %

A partir de estas pruebas, se tienen los siguientes resultados generales:

- La **Precisión del sistema** *ID6* (el mayor de todas las opciones) es de $73.53\,\%$. Y el sistema con menor precisión fue el *ID5* con $59,30\,\%$.
- $Recall_{dp}$ (Recall de detección de placas) da un valor promedio de $99.32\,\%$
- *Precisión por caracter* (medimos la precisión del clasificador de caracteres) obtiene el valor promedio de $93,05\,\%$
- El factor determinante y el cual hace decaer ligeramente la *Precisión por caracter* es el **Preprocesamiento de imagen**. Es el módulo que no implica un modelo Deep Learning, pero es el intermediario entre los 2 modelos presentes. Justamente el realizar pruebas reales, implica tener imágenes de todo tipo, algunas que sufren mucha distorsión, orientación inadecuada, desenfoque debido al movimiento de los automóviles (tal como se explicó en la Sección 2)), entre otros. Todo esto debido a la naturaleza del sistema de realizar las pruebas más reales posibles en un entorno real como el campus de la UNI.

2) Predicciones en tiempo real

Luego de las pruebas con grabaciones y obtenido la mejor versión del sistema de acuerdo a resultados de las tablas vistas (**ID 6**), se procedió a realizar pruebas en tiempo real, es decir, el sistema predecía en vivo los vehículos que entraban a la Puerta N°5. Para este propósito, se grabó la pantalla de la laptop porque es ahí donde sale el video con la placa predicha en tiempo real. Los siguientes datos muestran los resultados obtenidos de las pruebas.

Pruebas - 1:

TABLA XXII: Resultados de pruebas realizadas en la puerta N°5 de la UNI .
ID 1 - Predicción en vivo

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
AHU315	4HU315	83 %	0 %
F9X659	F9X659	100 %	100 %
BF7090	BF7090	100 %	100 %
ABC175	ABC175	100 %	100 %
D3D341	D3D341	100 %	100 %
COP773	COP773	100 %	100 %
C2C086	C2C086	100 %	100 %
CBX689	CBX689	100 %	100 %
B7S439	B7S430	83 %	0 %
AMJ142	4MJ142	83 %	0 %
BYH411	BYH411	100 %	100 %
ATF266	ATF266	100 %	100 %
V3M825	V3M825	100 %	100 %
ADZ596	——	0 %	0 %
BSA214	BSI214	83 %	0 %
B9C265	B9C265	100 %	100 %
ANN875	4NN875	83 %	0 %
Z6Z268	Z6Z268	100 %	100 %
C3I147	C3I147	100 %	100 %
D6V755	——	0 %	0 %
B9C617	B9C617	100 %	100 %
D5I612	——	100 %	0 %
BBR583	BBR583	100 %	100 %
AXX850	4XX850	83 %	0 %
BAK174	——	0 %	0 %

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
BZN277	BZN277	100 %	100 %
D1X352	1X352	83 %	0 %
AWH723	I1H723	67 %	0 %
APN340	4PN340	83 %	0 %
D7H245	D7H245	100 %	100 %
D9Q573	D9Q573	100 %	100 %
A7U451	I7U451	83 %	0 %
B3S343	B3S34	83 %	0 %
BHG424	BHG424	100 %	100 %
BME689	BME689	100 %	100 %
AXM683	AXM683	100 %	100 %
CCH149	CCH149	100 %	100 %
BIO209	——	0 %	0 %
AFZ912	——	0 %	0 %
CBF408	CBF408	100 %	100 %
AWX148	14X148	67 %	0 %
D7T355	J71375	50 %	0 %
AVQ625	IVQ625	83 %	0 %
BAR393	BIR393	83 %	0 %
B5E613	85E613	83 %	0 %
Promedio Total		91, 34 %	58, 97 %

Pruebas - 2:

TABLA XXIII: Resultados de pruebas realizadas en la puerta N°5 de la UNI . ID 2 - Predicción en vivo

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
B5E613	B5E613	100 %	100 %
BVN437	BVN437	100 %	100 %
BRJ158	BRJ158	100 %	100 %
AXI389	AXI389	100 %	100 %
BEO173	8EO173	83 %	0 %
8PK066	8PK066	100 %	100 %
A3Z697	A3Z697	100 %	100 %
BAL624	BAL624	100 %	100 %
T4D532	14D532	83 %	0 %
CAT590	UT590	67 %	0 %
BKJ674	BKJ674	100 %	100 %
ANE130	ANE130	100 %	100 %

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
BEZ682	BEZ682	100 %	100 %
ACR312	ACR312	100 %	100 %
W5V846	——	0 %	0 %
AVF354	AVF35H	83 %	0 %
BSY275	SY275	83 %	0 %
AXU609	AXU609	100 %	100 %
D6U379	DOU379	83 %	0 %
D8J144	D8J144	100 %	100 %
BST383	BST383	100 %	100 %
BAR172	BAR172	100 %	100 %
F4Z570	F4Z570	100 %	100 %
A1E027	1E027	83 %	0 %
ATW117	——	0 %	0 %
W2S359	W2S359	100 %	100 %
BNE574	BNE574	100 %	100 %
BKJ418	BKJ418	100 %	100 %
F8S829	F8S829	100 %	100 %
BHW405	BHW405	100 %	100 %
C6W153	C6H153	83 %	0 %
BBJ353	88J353	67 %	0 %
F3B854	F3B854	100 %	100 %
BRG554	BRG554	100 %	100 %
A4M633	A4M633	100 %	100 %
BXM230	BXM230	100 %	100 %
BRY185	3RY185	83 %	0 %
A2A369	A2A369	100 %	100 %
AAS014	AAS01H	83 %	0 %
BPQ146	BPQ146	100 %	100 %
ARZ355	ARZ355	100 %	100 %
B5R521	B5R52	83 %	0 %
C7R533	C7R533	100 %	100 %
ATW117	ATN177	83 %	0 %
D9P252	D9P25	83 %	0 %
AWJ249	AVJ249	83 %	0 %
BMG163	BMG163	100 %	100 %
AEC468	AEC4K8	83 %	0 %
AZZ313	AZZ313	100 %	100 %
AZX119	A2X119	83 %	0 %
APV168	APU168	83 %	0 %
C4K524	C4K524	100 %	100 %
CCS169	CCS169	100 %	100 %
AXZ623	AXZ623	100 %	100 %
BPP937	BPP937	100 %	100 %
BRJ158	BRJ158	100 %	100 %

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
D3D358	D3D358	$100\,\%$	$100\,\%$
BFY013	BFY013	$100\,\%$	$100\,\%$
D7K141	D7K141	$100\,\%$	$100\,\%$
BHY555	BHY555	$100\,\%$	$100\,\%$
ATW872	ATV872	$83\,\%$	0 %
F2D711	F29711	$83\,\%$	0 %
Promedio Total		$93,80\,\%$	$66,67\,\%$

De todas las pruebas realizadas en vivo, se obtiene una exactitud del $93\,\%$ en la detección de las placas, un promedio general de $92,57$ **%** para la precisión por carácter (modelo clasificador de caracteres) y de $62,82$ **%** para la precisión general del sistema.

Este mismo comportamiento se observa que ocurre en estos resultados y los previos (de las grabaciones), una *Precisión de caracter* alto, pero una *Precisión Sistema* no tan alta como la anterior. Esto muestra entonces que el factor **Preprocesamiento de imagen** es el factor principal del deterioro de la métrica. Como no procesa bien la imagen del carácter, el clasificador no es capaz de clasificar correctamente en algunos casos.

3) Predicciones con la cámara girada 45°

Otra manera de realizar las pruebas fue girar la cámara 45° grados desde el centro de la pista hacia la izquierda, de tal forma que vea directamente a los autos que ingresan a la universidad. De este modo se tiene la Tabla XXIV con los resultados de esta prueba.

TABLA XXIV: Resultados de pruebas realizadas en la puerta N°5 de la UNI . Predicción en vivo con 45° de inclinación

Placa Real	Placa Predicha	Precisión por caracter	Precisión Sistema
F8S829	F8S829	$100\,\%$	$100\,\%$
BHW405	BHW405	$100\,\%$	$100\,\%$
C6W153	C6H153	$83\,\%$	$0\,\%$
BBJ353	88J353	$67\,\%$	$0\,\%$
F3B854	F3B854	$100\,\%$	$100\,\%$
BRG554	BRG554	$100\,\%$	$100\,\%$
A4M633	A4M633	$100\,\%$	$100\,\%$
BXM230	BXM230	$100\,\%$	$100\,\%$
BRY185	3RY185	$83\,\%$	$0\,\%$
A2A369	A2A369	$100\,\%$	$100\,\%$
AAS014	AAS01H	$83\,\%$	$0\,\%$
BPQ146	BPQ146	$100\,\%$	$100\,\%$
ARZ355	ARZ355	$100\,\%$	$100\,\%$
B5R521	B5R52	$83\,\%$	$0\,\%$
C7R533	C7R533	$100\,\%$	$100\,\%$
ATW117	ATN177	$83\,\%$	$0\,\%$
D9P252	D9P25	$83\,\%$	$0\,\%$
AWJ249	AVJ249	$83\,\%$	$0\,\%$
BMG163	BMG163	$100\,\%$	$100\,\%$
AEC468	AEC4K8	$83\,\%$	$0\,\%$
AZZ313	AZZ313	$100\,\%$	$100\,\%$
AZX119	A2X119	$83\,\%$	$0\,\%$
APV168	APU168	$83\,\%$	$0\,\%$
C4K524	C4K524	$100\,\%$	$100\,\%$
CCS169	CCS169	$100\,\%$	$100\,\%$
AXZ623	AXZ623	$100\,\%$	$100\,\%$
BPP937	BPP937	$100\,\%$	$100\,\%$
BRJ158	BRJ158	$100\,\%$	$100\,\%$
Promedio Total		$92,75\,\%$	$60,71\,\%$

Los resultados obtenidos de estas pruebas son parecidos a los anteriores. En la precisión por carácter se obtiene un $92.75\,\%$ y en la precisión del sistema un valor de $60.71\,\%$. Entonces, las pruebas demuestran que un giro de 45 no afecta sustancialmente la precisión obtenida por el sistema.

VII. Conclusiones y Trabajo Futuro

Aquí se detallan las conclusiones recogidas de los resultados de las experimentaciones y pruebas con el sistema propuesto. Por último, se describen las posibles modificaciones, experimentaciones y desarrollos que podrían mejorar el sistema propuesto por algún lector interesado en el trabajo realizado.

A Conclusiones

El desarrollo de un sistema de reconocimiento de placas vehiculares mediante técnicas de Deep Learning, inicialmente orientado a su aplicación en la medida *Pico y Placa* de Lima, fue adaptado para el control vehicular en el campus de la UNI. Se concluye, **en primer lugar**, que las integraciones y optimizaciones realizadas con los recursos disponibles permitieron obtener un sistema robusto. Las pruebas con los modelos *Yolov4-tiny* y una variación al modelo *LeNet-5* arrojaron una precisión del **62,82 %** en la detección de matrículas usando cámaras de alta precisión. Esta precisión se vio limitada por la falta de un dataset público de imágenes de matrículas vehiculares peruanas, lo que dificultó optimizar completamente el modelo para este tipo de problema aplicado al Perú. Aun así, se valida el objetivo de reconocimiento en tiempo real.

El uso de una *GPU* fue clave, acelerando el entrenamiento hasta 10 veces. Se emplearon una GPU Tesla K80 en el clúster Jaguar y una RTX 2060 Super en una PC de escritorio.

Segundo, se destaca la importancia de elegir y evaluar arquitecturas adecuadas (en base a la literatura), ajustarlas al dataset y optimizar hiperparámetros del modelo,

el cual debe representar fielmente el problema para garantizar que el modelo aprenda y generalice correctamente. Estas consideraciones fueron críticas en el desarrollo del sistema.

Con base a los resultados y los objetivos específicos, se presentan las siguientes conclusiones:

- Los modelos tienen que ser personalizados y configurados acorde a la tarea a resolver, para este trabajo se evaluaron los modelos YOLOv3, Yolov3-tiny, Yolov4 y Yolov4-tiny para el propósito de reconocer matrículas peruanas con datasets públicos de placas vehiculares y caracteres del vocabulario español, llegando a obtener un $93\,\%$ en exactitud en la detección de placas peruanas en tiempo real.
- Se alcanzó una precisión de $92,57\,\%$ en la clasificación de caracteres presentes en las placas peruanas en las pruebas en tiempo real, resultado que demuestra las capacidades de las CNN pequeñas, pero robustas para cierto tipo de tareas. El modelo variado de Lenet-5 (AugLenet) demostró tener mejor precisión entre los otros modelos en las métricas de validación con el dataset escogido.
- Se implementó el sistema propuesto ALPR en el campus de la UNI para las pruebas sobre escenarios reales y se realizaron diferentes pruebas para medir su rendimiento: 1) Sobre videos: Se alcanzó un $73,53\,\%$ de precisión general del sistema y 2) Un $62,82\,\%$ en pruebas en tiempo real.

B Trabajo Futuro

Los resultados muestran un rendimiento y precisión aceptables, lo que se comprueba con las pruebas realizadas, pero no es perfecto ni el mejor de todos los sistemas similares a nivel mundial. Por tal motivo, la dirección de las investigaciones futuras debe ir en incrementar la precisión y rapidez de detección y ser adaptable a distintos países. En tal sentido, los lectores con mejores recursos podrían experimentar con modelos más robustos. Por otro lado, cada país tiene sus particularidades en cuanto al diseño de las

matrículas, lo cual es un punto importante en cuanto al procesamiento que requiere el sistema para su correcto reconocimiento. A continuación, se detallan 4 aspectos en los cuales el trabajo puede seguir desarrollándose.

Disponibilidad de Datos de entrenamiento de placas vehiculares peruanas

Una de las principales limitaciones encontradas para el trabajo fue la falta de un dataset público y extenso de matrículas vehiculares peruanas. Las particularidades de las placas en Perú, como su formato, diseño y caracteres específicos, vistas en este trabajo, afectan directamente a la capacidad de los modelos de Deep Learning para reconocerlas de forma correcta.

Esto condujo a usar datasets de repositorios públicos que no son peruanos, sino de Estados Unidos. Si bien el alfabeto contiene las mismas letras en ambos idiomas (omitiendo la letra ñ), un entrenamiento con un dataset grande de placas peruanas hubiera sido la mejor manera de entrenar.

Por ello, es crucial que futuras investigaciones se enfoquen en la creación de un dataset amplio y diverso que incluya imágenes de matrículas peruanas en diferentes condiciones, como iluminación, ángulo y distancias variables. Este dataset permitiría mejorar significativamente el rendimiento del sistema propuesto, al posibilitar un fine-tuning más específico y adecuado. Además, permitiría abordar casos especiales, como las placas deterioradas o sucias, que podrían afectar el reconocimiento en situaciones reales.

Pruebas en videos y en tiempo real en otros escenarios

Las pruebas de videos son el último paso y en el cual se mide la eficacia y utilidad del sistema propuesto. En tal sentido se requieren más pruebas de videos, tanto grabaciones en las calles de Lima así como grabaciones en vivo por medio de una cámara adecuada para este propósito. La conexión a la nube (*cloud computing*) representa una dirección prometedora para este trabajo, ya que permitiría el monitoreo de las calles de Lima desde una central de control.

Se menciona esto dado que los escenarios en las pruebas de este trabajo no fueron intensivos, por ejemplo, en tráfico denso, condiciones meteorológicas adversas (neblina o lluvia), en horas de la noche o con diferentes iluminaciones.

Datos de entrenamiento y testeo de otros países

En relación a los datos de entrenamiento especificados en este trabajo para la detección de matrículas de autos, se mencionó que es un dataset público y no está hecho con placas peruanas, pero sí son parecidas. En tal sentido, sería importante realizar el entrenamiento en base a un dataset de placas peruanas, que hasta hoy en día no se tiene. Este problema del dataset específico de placas peruanas sería un trabajo futuro útil a realizar.

Por el lado de la clasificación de caracteres, el dataset *Chars74k* se adecuó a nuestro modelo de reconocimiento de caracteres, existen otros datasets para otros tipos de caracteres de otros países. En tal sentido, los datasets se escogen de acuerdo a los países. El sistema presentado puede ser testeado ante matrículas de otros países, por lo cual es una evaluación deseable en un futuro. Se desconoce el nivel de precisión que podría alcanzar en otro contexto. En esta dirección, el sistema puede ser entrenado con otro dataset y con el procesamiento de la matrícula adecuada a cada país para que pueda adaptarse a él.

Evaluar y comparar resultados con otros modelos

Existen otros modelos que realizan las detecciones de objetos como se mencionó anteriormente, entre ellos las versiones de Yolo más recientes (v5-v9) y los de tipo Two-Stage. Estas nuevas versiones de Yolo podrían mejorar los resultados; sin embargo, se debe tener en cuenta el tamaño del modelo, el cual afecta el tiempo de inferencia (y por ende la particularidad de detección en tiempo real del trabajo). También está Fast R-CNN, que es uno de los modelos Two-Stage populares de detección de objetos, el cual compite con el modelo Yolov4 presentado acá y que podría presentar mejores resultados. Pero se

debe tener en cuenta la velocidad de procesamiento del modelo, donde un valor aceptable y mínimo requerido sería de 30 FPS. En modelos de clasificación de imágenes, existen muchos más; sin embargo, estos no deberían ser tan complejos para detectar caracteres, puesto que es deseable un modelo efectivo, pequeño y rápido.

VIII. Referencias bibliográficas

[1] Géron Aurélien. Hands-on machine learning with scikit-learn & tensorflow. *O'Reilly Media, Sebastopol, CA, USA*, page 255, 2017.

[2] Blogthinkbig. Historia de la ia: Frank rosenblatt y el mark i perceptrón, el primer ordenador fabricado específicamente para crear redes neuronales en 1957. `https://empresas.blogthinkbig.com/historia-de-la-ia-frank-rosenblatt-y-e/`. Online; accedido 1-Febrero-2020.

[3] Hogne Jørgensen. Automatic license plate recognition using deep learning techniques. Master's thesis, NTNU, 2017.

[4] Diego Calvo. Red neuronal convolucional cnn. `https://www.diegocalvo.es/red-neuronal-convolucional/`. Online; accedido 1-Febrero-2020.

[5] Satya Mallick. Code opencv in visual studio. `https:www.learnopencv.com`. Online; accedido 1-Febrero-2020.

[6] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[9] Huansheng Song, Haoxiang Liang, Huaiyu Li, Zhe Dai, and Xu Yun. Vision-based vehicle detection and counting system using deep learning in highway scenes. *European Transport Research Review*, 11(1):1–16, 2019.

[10] Jesus Enrique Piscoya Ferreñan. Sistema de visión artificial para apoyar en la identificación de plagas y enfermedades del cultivo de sandía en el distrito de ferreñafe. 2019.

[11] María Victoria Palacín Silva. Visión artificial aplicada al monitoreo automatizado del proceso de cloración para mejorar la calidad del agua. 2011.

[12] Francisco José Núñez Sánchez-Agustino. Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional. 2016.

[13] Hamed Saghaei. Proposal for automatic license and number plate recognition system for vehicle identification. *arXiv preprint arXiv:1610.03341*, 2016.

[14] Nuzulha Khilwani Ibrahim, Emaliana Kasmuri, Norazira A Jalil, Mohd Adili Norasikin, Sazilah Salam, and Mohamad Riduwan Md Nawawi. License plate recognition (lpr): a review with experiments for malaysia case study. *arXiv preprint arXiv:1401.5559*, 2014.

[15] Zied Selmi, Mohamed Ben Halima, and Adel M Alimi. Deep learning system for automatic license plate detection and recognition. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 1, pages 1132–1138. IEEE, 2017.

[16] Brayan De Jesús Ramirez Mejía and Mack Rolly Tito Apaza. Reconocimiento automático de placas de rodaje utilizando una red neuronal convolucional para el ingreso de vehículos en la universidad ricardo palma. 2020.

[17] MMLima. Pico y placa. `https://aplicativos.munlima.gob.pe/pico-y-placa`. Online; accedido 26-Febrero-2024.

[18] MMLima. Pico y placa. https://aplicativos.munlima.gob.pe/pico-y-placa/camion. Online; accedido 26-Febrero-2024.

[19] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[25] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[26] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[27] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[29] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[31] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[32] David H Hubel. Single unit activity in striate cortex of unrestrained cats. *The Journal of physiology*, 147(2):226, 1959.

[33] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

[34] Stevo Bozinovski. Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica*, 44(3), 2020.

[35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[36] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3. Edinburgh, 2003.

[37] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[38] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[39] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104:154–171, 2013.

[40] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[42] Juan Terven and Diana Cordova-Esparza. A comprehensive review of yolo: From yolov1 to yolov8 and beyond. *arXiv preprint arXiv:2304.00501*, 2023.

[43] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[44] Shay Aharon, Louis-Dupont, Ofri Masad, Kate Yurkova, Lotem Fridman, Lkdci, Eugene Khvedchenya, Ran Rubin, Natan Bagrov, Borys Tymchenko, Tomer Keren, Alexander Zhilko, and Eran-Deci. Super-gradients, 2021.

[45] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. Yolov6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022.

[46] Sayani Sarkar and Nathan Johnson. A deep-learning, vision-based framework for testing swarm algorithms using inexpensive mini drones. page 12, 05 2022.

[47] George Antonio Mundaca Vidarte. Detección de caracteres de placas de automóviles mediante técnicas de visión artificial. 2016.

[48] Gerardo Alfonso Joel Espinoza Vásquez. Sistema de reconocimiento de patrones en placas vehiculares para el acceso automático de visitas a un edificio. 2014.

[49] Utsha Saha, Imam Uddin Ahamed, and M Iffat Hossain. Yolov8 for bangla license plate recognition: Advancing real-time object detection in localized contexts. In *2024 7th International Conference on Informatics and Computational Sciences (ICICoS)*, pages 413–418. IEEE, 2024.

[50] Tarqui Pisaya and Jessica Mishell. Sistema para reconocimiento automático de placas vehiculares mediante una biblioteca openalpr. 2024.

[51] Shijuan Chen, Zongmei Li, Xiaofeng Du, and Qin Nie. Eand-lprm: Enhanced attention network and decoding for efficient license plate recognition under complex conditions. *Algorithms*, 17(6):262, 2024.

[52] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[53] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[54] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. *Advances in neural information processing systems*, 31, 2018.

[55] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020.

[56] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *CoRR*, abs/1908.08681, 2019.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[58] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[59] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[60] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

[61] Samsung. Dispositivo móvil. `https://www.samsung.com/latin/smartphones/galaxy-a30-a305/SM-A305GZBKTPA/`. Online; accedido 1-Septiembre-2020.

[62] Huawei. Router inalámbrico. `https://consumer.huawei.com/pe/routers/ws318n/specs/`. Online; accedido 3-Abril-2023.

[63] tp link. Inyector poe tl-poe150s. `https://www.tp-link.com/pe/business-networking/accessory/tl-poe150s/#specifications`. Online; accedido 3-Abril-2023.

[64] Angelo Vittorio. Toolkit to download and visualize single or multiple classes from the huge open images v4 dataset. `https://github.com/EscVM/OIDv4_ToolKit`, 2018.

[65] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[66] Jiaqi Wang, Kai Chen, Shuo Yang, Chen Change Loy, and Dahua Lin. Region proposal by guided anchoring. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2965–2974, 2019.

[67] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[68] Tiancai Wang, Rao Muhammad Anwer, Hisham Cholakkal, Fahad Shahbaz Khan, Yanwei Pang, and Ling Shao. Learning rich features at high-speed for single-shot object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1971–1980, 2019.