Universidad Autónoma de Baja California

Tesis de Maestria

Implementación de un sistema de detección vehicular mediante el uso de Inteligencia Artificial en plataformas embebidas

---

Autor:
Jonathan Jesus Sanchez Castro

Director de Tesis:
Dr. Julio Cesar Rodríguez Quiñonez

Co-Director de Tesis:
Dr. Guillermo Galaviz Yáñez

Una tesis presentada en cumplimiento de los requisitos para el grado de Maestro en Ciencias.

en la

Facultad de Ingeniería
Campus Mexicali

1 de junio de 2021

UNIVERSIDAD AUTÓNOMA
DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA
MEXICALI

Declaración de Autoría

Yo, Jonathan Jesus Sanchez Castro, declaro que esta tesis titulada, "Implementación de un sistema de detección vehicular mediante el uso de Inteligencia Artificial en plataformas embebidas " y el trabajo presentado en ella es de mi autoría. Confirmo que este trabajo enviado para evaluación ha sido elaborado por mí y se expresa en mis propias palabras. Cualquier uso que se haga en él de las obras de otros autores en cualquier forma (por ejemplo, ideas, ecuaciones, figuras, texto, tablas, programas) se reconoce adecuadamente al momento de su uso. Se incluye una lista de las referencias empleadas.

Firma: Sanchez Jonathan

Fecha: 4/6/2021

"Lo importante es no dejar de cuestionar. La curiosidad tiene su propia razón de existir."

Albert Einstein

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Resumen

Facultad de Ingeniería
Campus Mexicali

Maestro en Ciencias.

Implementación de un sistema de detección vehicular mediante el uso de Inteligencia Artificial en plataformas embebidas

por Jonathan Jesus Sanchez Castro

En este trabajo de tesis se tiene como objetivo el realizar un sistema de reconocimiento vehicular mediante la aplicación de algoritmos de inteligencia artificial, específicamente las redes neuronales convolucionales de aprendizaje profundo, debido a sus buenos resultados en el área de visión de máquina reportados en los últimos años. Estos algoritmos tuvieron mayor desarrollo e implementación durante el transcurso de la última década y han desplazado en tareas de manejo de imágenes a otros como las máquinas de soporte vectorial o vecinos cercanos, por lo que la presente investigación implementará estas redes convolucionales para realizar la clasificación y detección de vehículos. El propósito de aplicación es desarrollar un sistema visual de detección de vehículos para apoyo de las vialidades de la ciudad de Mexicali, ya que su uso se exiende a temas de seguridad, mantenimiento y estadística vehicular. En la presente investigación se realizan seis propuestas de arquitecturas esbeltas de redes neuronales convolucionales que permiten la implementación de estas redes para las aplicaciones de clasificación y detección de vehículos.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACYT), por el apoyo brindado económicamente y materialmente para el desarrollo de la investigación durante estos dos años.

A la Universidad Autónoma de Baja California (UABC), al permitir hacer uso de sus instalaciones para desarrollo de investigación y por la oportunidad brindada para poder realizar mis estudios de posgrado y permitirme obtener el grado de Maestro en Ciencias.

A la Universidad Autónoma de Sinaloa (UAS), por brindarme la oportunidad de continuar con mis estudios de posgrado al otorgarme apoyo económico durante estos dos años de estudio.

A la Facultad de Ingeniería de la UABC Campus Mexicali, por permitirme hacer uso de sus excelentes instalaciones y además de proveer del equipo necesario para el desarrollo de la investigación.

A mi director de tesis, el Dr. Julio Cesar Rodríguez Quiñonez al haber brindado la oportunidad de realizar mis estudios de posgrado bajo su tutela, por su tiempo y dedicación en el apoyo brindado para realización de esta investigación.

A mi Co-director de tesis, el Dr. Guillermo Galaviz Yáñez por su apoyo brindado con sugerencias y retroalimentación para desarrollo y mejora de la investigación durante el transcurso de estos dos años.

A mi comité de tesis, el Dr. Oleg Sergiyenko, Dra. Wendy Flores Fuentes, Dr. Lars Lindner por su apoyo brindado cada semestre con retroalimentación en cada presentación de avances de tesis.

Índice General

Índice de Figuras

Índice de Tablas

Abreviaturas

IA	**I**nteligencia **A**rtificial
DNN	**D**eep **N**eural **N**etworks
CNN	**C**onvolutional **N**eural **N**etworks
SBC	**S**ingle **B**oard **C**omputer
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
ML	**M**achine **L**earning
DL	**D**eep **L**earning
SVM	**S**upport **V**ector **M**achine
ANN	**A**rtificial **N**eural **N**etwork
MSE	**M**ean **S**quare **E**rror
CPU	**C**entral **P**rocessing **U**nit
RAM	**R**andom **A**cces **M**emory
GPU	**G**raphic **P**rocessing **U**nit
SSD	**S**olid **S**tate **D**rive
HDD	**H**ard **D**rive **D**isk
ReLU	**Re**ctifier **L**inear **U**nit
YOLO	**Y**ou **O**nly **L**ook **O**nce
RCNN	**R**egions **C**onvolutional **N**eural **N**etworks
FPS	**F**rames **P**er **S**econd
DCP	**D**ataset **C**reation **P**rogram
SVR	**S**istema de **R**econocimiento **V**ehicular
DA	**D**ata **A**ugmentation
KNN	**K**-**N**earest **N**eighbor
ACC	**Acc**uracy

UAV **U**nmanned **A**erial **V**ehicle

Símbolos

Símbolo	***Nombre***
X_n	**Entradas**
Y_n	**Salidas**
W_n	**Pesos**
H_n	**Neurona**
$F(x)$	**Función de Activación**
e	**Error**
η	**Tasa de Aprendizaje**
z	**Señal de entrada**
g	**Función de transferencia**
K	**Banco de filtros**
I	**Filtro**
b_j	**Bias**
P_j	**Mapa de Características**
C_i	**Ventana de pooling**
S	**Cantidad de celdas en YOLO**
B	**Bounding Boxes**
C	**Clases**
p	**Nivel de confianza**
x	**Coordenada central del eje X del Bounding Box**
y	**Coordenada central del eje Y del Bounding Box**
w	**Ancho del Bounding Box**
h	**Alto del Bounding Box**
E_c	**Error de coordenadas centrales**

E_d	**Error de dimensiones de Bounding Box**
E_{cf}	**Error de nivel de confianza**
E_{cl}	**Error de clases**
λ_{co}	**Parámetro de fortalecimiento al valor de error**
λ_{nbj}	**Parámetro para evitar que el calor de una celda llegue a cero**
$¶_{ij}^{obj}$	**Parámetro que denota cuando existe un objeto dentro de la celda**
$¶_{ij}^{nbj}$	**Parámetro que denota cuando no existe objeto dentro de la celda**
$p_i(C)$	**Probabilidad de clase en la celda i**
PC_i	**Valor de confianza en la celda i**
$\tilde{P}_1$	**Punto inicial del Bounding Box**
$\tilde{P}_2$	**Punto final del Bounding Box**
$C(x, y)$	**Centro del Bounding Box**
$C_{norm}(x, y)$	**Centro normalizado del Bounding Box a la imagen**
$D(h, w)$	**Dimensiones del Bounding Box**

Esta tesis se la dedico principalmente a mis padres por sus sacrificios, cariño, amor y consejos brindados durante el transcurso de mi vida, los cuales me han permitido llegar a ser la persona que soy actualmente y poder haber llegado hasta este momento de mi vida.

A mis hermanos, primos y amigos por siempre apoyarme durante el transcurso de mis estudios y estar presentes en los momentos difíciles a lo largo de mi vida.

Capítulo 1

Introducción

1.1. Antecedentes

La inteligencia artificial (IA) ha evolucionado en gran medida en los últimos años, tanto que se ha logrado su implementación en distintos temas de investigación para resolución de problemáticas importantes como lo es la navegación automática, reconocimiento facial, análisis de señales, interpretación de caracteres, reconocimiento de escenas, monitoreo de tráfico, etc. Tomando este último ejemplo, podemos dar una idea del tipo de aplicaciones que tiene la IA, por ejemplo se podría implementar un sistema automatizado basado en la detección de vehículos para apoyar al área de infraestructura civil de una ciudad en donde podría brindar apoyo en la selección del tipo de material utilizado para el pavimento, ya que este sistema puede otorgarnos datos más concretos sobre el tipo de vehículo que viaja por la vialidad y de esa manera planificar el tipo de material y gastos que se requieran. La IA tiene una innumerable cantidad de aplicaciones que las colocan como una de las tecnologías de mucho interés para la investigación, pero ha sido hasta en años recientes cuando este interés se ha incrementado considerablemente debido a que se han logrado avances importantes, como la creación de algoritmos de aprendizaje profundo (Deep Learning) que permiten la implementación de los algoritmos en aplicaciones como en la de los ejemplos antes mencionados. Como se ha mencionado el interés por utilizar la IA para distintas aplicaciones y nuevas investigaciones ha crecido considerablemente gracias a los avances computacionales y a la llegada de las Deep Neural Networks (DNN) Yao et al. [2019], Traore

et al. [2018], Shi et al. [2016]. Una de las áreas más beneficiadas es la de clasificación de imágenes, ya que es necesario el manejo de una gran base de datos para crear algoritmos de clasificación, en donde las DNN tienen un buen desempeño Yao et al. [2019]. Con ayuda de equipo computacional y de estos algoritmos de aprendizaje profundo se es capaz de implementar una red de IA para la clasificación de imágenes en tiempo real Yao et al. [2019], ya sea para la identificación de personas, autos o motocicletas en la vía pública, seguridad en centros comerciales, disponibilidad de lugar en estacionamientos, etc. Existen diversos algoritmos de clasificación, sin embargo, para el manejo de imágenes se prefiere utilizar las Deep Neural Networks (DNN), donde se encuentran distintos métodos de clasificación utilizados en investigaciones recientes. Uno de los métodos más utilizados en clasificación de imágenes es el de Convolutional Neural Networks (CNN), esto debido a su buen rendimiento, manejo de datos y mejor exactitud en la clasificación Traore et al. [2018], Seo and Shin [2019], Mao et al. [2016a]. La CNN es una evolución del perceptrón multicapa Yao et al. [2019], el cual es uno de las primeras redes neuronales creadas para la clasificación de patrones. La CNN cuenta con una estructura similar al perceptrón multicapa, ya que cuenta con una capa de entrada, de salida y varias capas internas, la diferencia nace en las capas internas ya que estas realizan diferentes tareas. Generalmente las CNN cuentan con las siguientes capas internas en su arquitectura: capa de convolución, pooling, activación y al final una capa completamente conectada (Fully connected layer) Traore et al. [2018], Jain et al. [2019]. Algunas de las ventajas que se presentan en las CNN se enlistan a continuación:

- Mayor velocidad de entrenamiento.
- Extraen automáticamente las cualidades buscadas en cada imagen sin necesidad de hacerlo manualmente.
- Capacidad de trabajar con grandes cantidades de información.

Las aplicaciones de estos algoritmos de clasificación pueden llegar a abarcar diversos campos como: seguridad, medicina, investigación, márketing, agricultura, reconocimiento facial, reconocimiento de accidentes, etc. Existen un gran número de aplicaciones para algoritmos de clasificación y en algunos existe la necesidad de tener mayor portabilidad para su uso, en estos casos entran los sistemas embebidos, en los cuales es posible descargar el algoritmo de clasificación para usarlo a

distancia de una computadora o servidor Gong et al. [2017]. Para ejemplificar este caso podemos tomar la aplicación de uso de UAV para la detección de grietas en edificios, la identificación de número de matrícula de un automóvil o reconocimiento de espacio para la navegación de un robot Pereira and Pereira [2015], Lee et al. [2017a], Park et al. [2018]. Los sistemas embebidos, son dispositivos electrónicos de bajo costo y consumo de energía, que han demostrado ser de gran utilidad para aplicaciones donde no es posible el utilizar una computadora. En el área de los sistemas embebidos podemos encontrar a los FPGA's y Single Board Computer (SBC). Los SBC como su nombre indica son computadoras de una sola placa, se destacan por su tamaño reducido, eficiencia de energía, variedad de puertos y versatilidad con diferentes lenguajes de programación, lo cual las coloca como una buena alternativa a usar como sistema embebido, para ejemplo de esto se tienen aplicaciones como el control inteligente del clima en edificios, detección de objetos o seguimiento de objetos Aftab et al. [2017], Yang et al. [2018], Mao et al. [2016b]. Los sistemas embebidos son usados para suplir una tarea en específico debido a su reducida capacidad de procesamiento y memoria comparado con un ordenador normal.

1.2. Planteamiento del Problema

Los algoritmos de IA se utilizan para resolver distintas tareas donde dependiendo del tipo de aplicación se decide el tipo de trabajo que estos van a realiza. Como se ha mencionado podemos tener algoritmos de IA para clasificación y detección de objetos. Con estos algoritmos se pueden crear sistemas de vigilancia, monitoreo, conteo, entre otros. Basándonos en estas posibles aplicaciones se plantea crear dos algoritmos que puedan apoyar con la tarea de identificar los vehículos que transitan por una vialidad específica, esto para de recolectar información más precisa y específica. Se busca que el método desarrollado pueda servir de apoyo en diversas tareas como la planeación de pavimentación de una vialidad para evitar gastos excesivos en reparaciones. También se podría utilizar la información para análisis estadísticos de una ciudad, esto porque se busca que el proceso de conteo proporcione mayor información en comparación con los métodos utilizados actualmente como las mangueras neumaticas SCT [2016]. Se consideran dos algoritmos de apoyo para esta problemática, uno para la clasificación directa de los vehículos y otro para la detección de los vehículos que circulen por la vialidad. Para el desarrollo de

estos algoritmos se planea utilizar a las CNN ya que se ha comprobado su utilidad y eficacia en este tipo de tareas.

Preguntas de Investigación:

¿Qué estructura de red neuronal, considerando el número de capas, función de optimización, método de entrenamiento y neuronas de salida, permitirá la detección de vehículos sobre carreteras, a la vez que puede ser implementada dentro de un sistema embebido?

¿En qué medida es necesario profundizar una red neuronal convolucional para realizar la tarea de clasificación de una cantidad reducida de objetos (vehículos: 5 categorías), para utilizarse en un sistema embebido?

1.3. Justificación y Uso de los Resultados

La integración de un algoritmo de clasificación o de detección en un sistema embebido es una tarea que ha recibido gran atención en años recientes debido a que en ciertas aplicaciones se necesita la portabilidad, por lo que el espacio, el consumo de energía, el peso, entre otros factores limita el uso de equipo con mayores recursos computacionales y es donde entran los sistemas embebidos. Se ha comprobado que se puede lograr la conjunción de un algoritmo de Deep Learning y un sistema embebido, sin embargo, la clasificación y detección de vehículos en sistemas embebidos sigue estando en desarrollo debido a que este tipo de clasificación necesita de una mayor capacidad de procesamiento de la que ofrecen los sistemas embebidos, por lo que los algoritmos de clasificación de vehículos siguen recibiendo nuevos métodos de optimización y nuevos acercamientos utilizando redes neuronales para mejorar su rendimiento.

La implementación de un sistema de reconocimiento vehicular es uno de los retos que tiene gran número de investigaciones debido a sus aplicaciones en el sector automotriz. Con la llegada de las CNNs se han tomado nuevos acercamientos para la implementación de sistemas de clasificación de vehículos, sin embargo, las investigaciones recabadas indican que aún se siguen desarrollando nuevas técnicas para hacer algoritmos más precisos y disminuir la carga de procesamiento para su implementación en sistemas embebidos. Por lo tanto, esta investigación busca

aportar nuevos conocimientos para la clasificación y detección de vehículos y su implementación en tiempo real utilizando sistemas embebidos.

1.4. Objetivos de la Investigación

1.4.1. Objetivo General

Crear e implementar dos algoritmos de IA para las tareas de detección y clasificación de vehículos respectivamente, donde estos algoritmos estén basados en redes neuronales y se logre su implementación en un sistema embebido.

1.4.2. Objetivos Específicos

- Crear una base de datos específica para la clasificación y detección.
- Desarrollar un algoritmo de clasificación basado en redes neuronales y que su precisión sea igual o mayor que un 80 % en prueba.
- Crear un algoritmo de detección basado en redes neuronales y que su precisión sea igual o mayor que 80 % en prueba.
- Desarrollar e implementar un algoritmo capaz de superar los 20 cuadros por segundo para percibir fluidez ne su la aplicacion en video.
- Desarrollar un algoritmo de clasificación que permita su implementación en un SBC (Jetson Nano).

1.5. Hipótesis

Las redes neuronales convolucionales son algoritmos de Deep Learning muy poderosos que han permitido el desarrollo de la IA en distintas áreas de investigación, pero un punto importante a considerar es el reto que significa poder implementar estos algoritmos dentro de un sistema embebido, esto debido a los recursos de CPU, RAM, SSD o HDD reducidos con los que estos cuentan. Por lo que se ha planteado desarrollar arquitecturas de CNN esbeltas, con las que se pueda lograr

implementar los algoritmos de detección y de clasificación dentro de un sistema embebido, también el utilizar las librerías de software disponibles para segmentación de tareas entre el CPU y GPU.

Capítulo 2

Marco Teórico

2.1. Inteligencia Artificial

2.1.1. Definición

La inteligencia artificial ha sido un campo muy estudiado durante las últimas décadas debido a la gran cantidad de aplicaciones que esta logra tener, tales como: reconocimiento facial, reconstrucción facial, navegación automática, predicción de mercado, detección y segmentación de objetos, entre otros Alzubaidi et al. [2021]. La IA se ha ido desarrollando durante el transcurso de los años y se han escrito distintas definiciones como "Estudio de la computación que observa que una máquina sea capaz de percibir, razonar y actuar" de Winston [1992], esta definición se menciona en el libro de Russell and Norvig [2004], donde se explica que se pueden agrupar las diferentes definiciones existentes en cuatro divisiones: sistemas que piensan como humanos, que piensan racionalmente, que actúan como humanos y que actúan racionalmente. Si bien la mayoría de las definiciones mencionan a la IA como un cúmulo de pensamientos auto capaces de comprensión, razonamiento y análisis, pero esto no puede estar alejado de la realidad actual, debido a que aún no se ha acercado a lo que estas definiciones indican sobre la IA. Para establecer una definición actual de IA más general podría ser la siguiente:

- El esfuerzo de automatizar tareas intelectuales realizadas normalmente por seres humanos Chollet et al. [2018]

2.1.2. Inteligencia Artificial: Antecedentes

La Inteligencia Artificial tiene sus inicios a partir la mitad del siglo XX cuando McCullouch y Pitts diseñaron la teoría sobre la primera neurona artificial, la cual era capaz de resolver funciones aritméticas y lógicas básicas. El perceptrón como es llamada esta neurona artificial es el modelo matemático basado en la fisiología de una neurona la cual se observa en la figura 2.1, a grandes rasgos una neurona cuenta con un núcleo, dendritas y axón los cuales al momento de realizar el proceso de sinapsis permiten el intercambio de corrientes eléctricas entre varias neuronas y así crear lo que se conoce como una red neuronal.

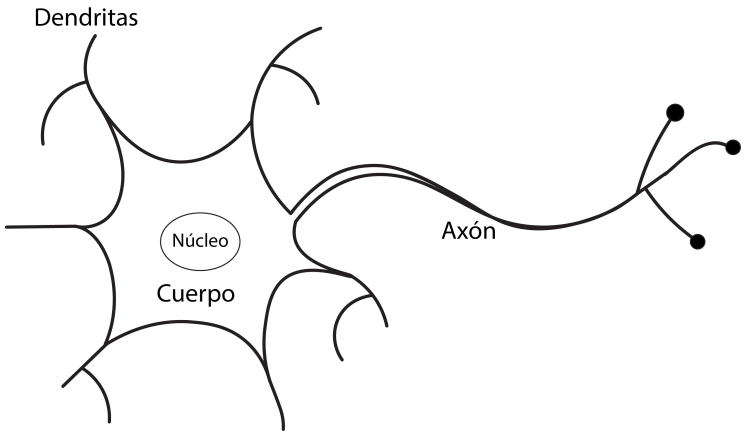


FIGURA 2.1: Modelo de neurona biológica

El perceptrón está basado en el procedimiento natural de las neuronas para transportar información, la figura 2.2 muestra un esquema de cómo está constituido una neurona artificial, donde observamos que tenemos entradas (dendritas), un cuerpo y una salida (Axón), este modelo de neurona artificial es el que abre paso al nacimiento de la inteligencia artificial como la conocemos hoy.

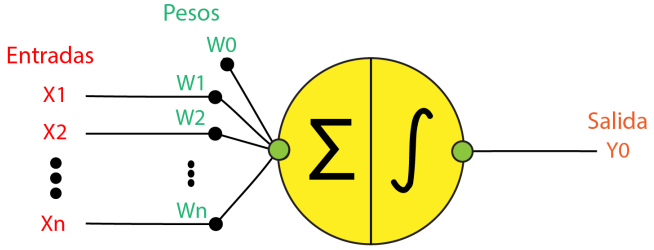


FIGURA 2.2: Modelo de neurona artificial

El funcionamiento de un perceptrón es explicado a continuación:

a) Las entradas x (Dendritas) son multiplicadas por los pesos w.

b) Después de realizar la multiplicación se continúa con la sumatoria de las multiplicaciones.

c) El resultado de la sumatoria es enviado a una función de activación.

d) Como salida y (Axón) obtenemos el valor generado por la función de activación.

El procedimiento para realizar el "Aprendizaje" se realiza mediante el reajuste de las variables w (pesos), se utiliza la retroalimentación como si fuera un sistema. La retroalimentación se logra utilizando una función de transferencia, a este procedimiento se le conoce como "Backpropagation" y será retomado poco más adelante en la sección de Redes Neuronales 2.3.1.

La IA es un amplio tema de investigación el que cuenta con grandes avances y aportaciones durante su desarrollo, lo que ha dado como origen a subdivisiones de la IA. Estas son el Aprendizaje de Máquina (Machine Learning) y Aprendizaje Profundo (DL), siendo DL una subdivisión del mismo ML pero que en la actualidad DL es un campo de gran importancia debido a su impacto en el campo de IA, gracias a que ha llegado a revolucionar la manera de aplicar los algirtmos de IA en tareas de vision de máquina. En esta tesis nos enfocaremos en estas dos areas de IA, pero principalmente en DL y sus Redes Neuronales Convolucionales (CNN). En la figura 2.3 se muestra un simple diagrama de como esta constituida el área de IA.

2.2. Machine Learning

Durante el transcurso de las siguientes décadas después de la aparición del perceptrón se desarrollaron los algoritmos de Machine Learning (ML), los cuales nacen de la premisa de hacer que una computadora "aprenda" información para realizar la toma de decisión en tareas como regresión, clasificación o predicción. Los algoritmos de ML se basan en métodos estadísticos para lograr un mapeo de los datos (como se muestra en la figura 2.4) y representarlos en diferentes espacios

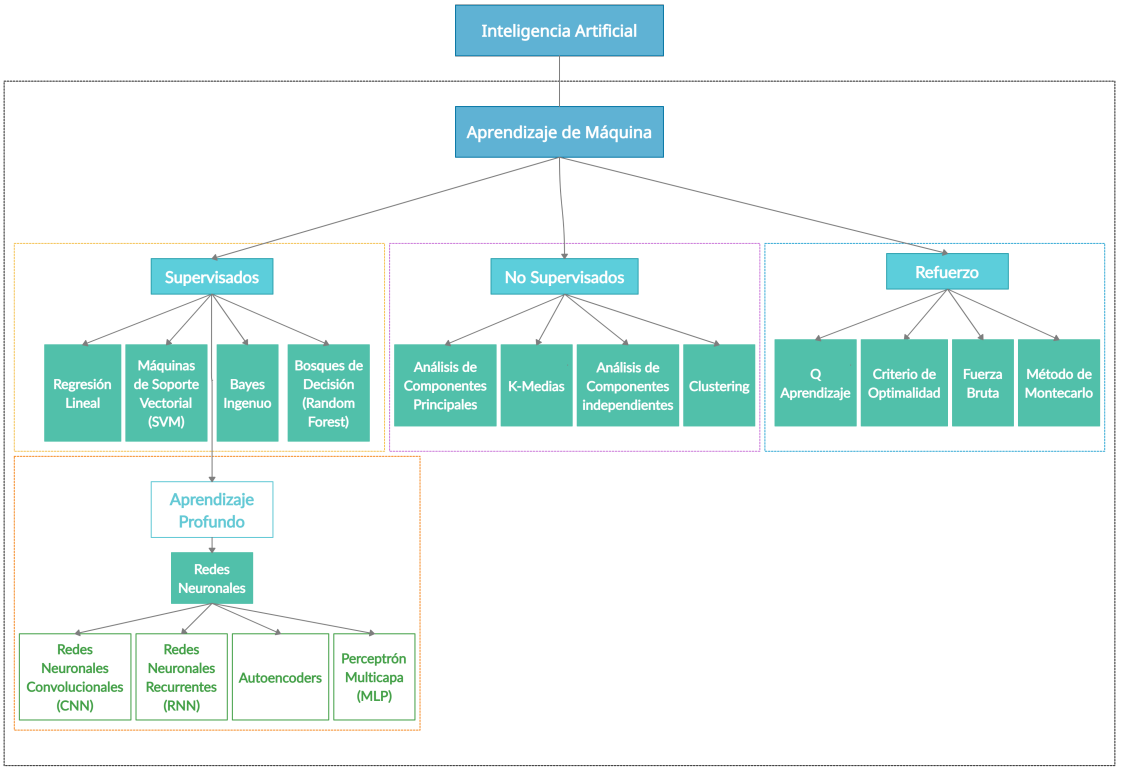
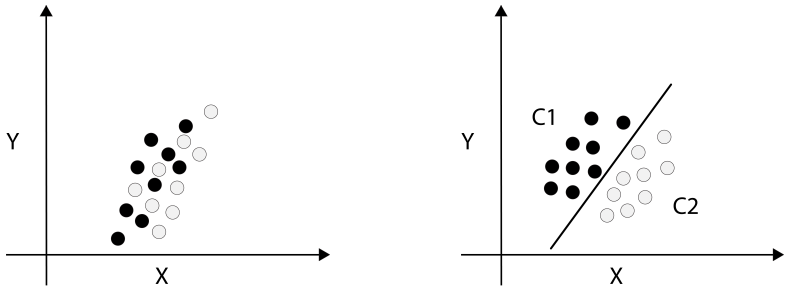


FIGURA 2.3: Inteligencia Artificial

que faciliten su interpretación para las tareas de regresión y clasificación, como se puede observar en la figura 2.4 (b) se ejemplifica el caso de lograr la separación de dos tipos de clases de datos Goodfellow et al. [2016].



(A) Datos sin acondicionar (B) Datos acondicionados

FIGURA 2.4: Representación en espacio

Los algoritmos de ML a su vez se encuentran divididos en dos categorías, en supervisados y no supervisados, a continuación se explican más a detalle las diferencias entre cada uno.

2.2.1. Algoritmos Supervisados

Los algoritmos de ML supervisados, son aquellos que buscan generar una función que represente a una base de datos, mediante la estimacion de los datos de salida $\boldsymbol{y}$ con respecto a los datos de entrada $\boldsymbol{x}$, esto mediante uso de reglas estadísticas. Las bases de datos se encuentran "etiquetadas", es decir ya cuentan con un valor o categoría definida para el proceso de entrenamiento de los algoritmos, las tareas que estos pueden realizar son de regresión y clasificación. Los algoritmos supervisados comúnmente utilizados hacen uso de la función de distribución de probabilidad $\mathrm{p}(y|\boldsymbol{x})$ en sus operaciones para los entrenamientos, algunos de estos algoritmos supervisados son:

- Regresión Lineal.
- Máquinas de Soporte Vectorial (SVM).
- Bayes Ingenuo.
- Arboles de decisión.
- Redes Neuronales.

2.2.2. Algoritmos No Supervisados

Los algoritmos de ML no supervisados son aquellos que su trabajo no es directamente el generar una respuesta específica de salida, sino el generar representaciones más detalladas, extraer características, realizar separaciones de grupos de clases y reducir la dimensionalidad de los datos. Todas estas tareas no necesitan base de datos con "etiquetas" ya que estos algoritmos no necesitan producir un resultado conciso, solamente hacer la interpretación de los datos. Los algoritmos no supervisados son utilizados para hacer agrupaciones (Clustering), reducción de dimensionalidad, estimación de densidad, entre otros.

En esta rama del ML podemos encontrar a distintos algoritmos no supervisados como, por ejemplo:

- Análisis de componentes principales.

- Agrupamiento de K-medias.
- Análisis de componentes independientes.

Muchos de estos algoritmos no supervisados son utilizados en la minería de datos debido a que se encuentran enfocados al manejo de datos, principalmente para realizar separaciones, agrupaciones, extracciones de los datos para lograr comprender la información obtenida de la medición de algún fenómeno o proceso.

2.2.3. Aplicaciones

La IA ha incrementado su popularidad por parte de la comunidad de investigadores en los últimos años y ha generado una diversidad de aportaciones para multiples aplicaciones en distintas áreas de oportunidad. Se han llegado a utilizar en las áreas de seguridad, higiene, automovilismo, militar, biometría, comercio, entre otras. Algunos ejemplos de aplicación de la IA son: sistema de reconocimiento facial el cual usa como clasificador las redes neuronales Hu et al. [2017], clasificador de vehículos utilizando vecinos cercanos, clasificación de vehículos mediante máquinas de soporte vectorial utilizando historial de gradientes orientados (HOG) Lee et al. [2015], reconocimiento de imágenes médicas de la zona abdominal Kondo et al. [2014], reconocimiento de la calidad del aire Chakma et al. [2017], análisis de señales.

2.3. Aprendizaje Profundo

El aprendizaje profundo (AP) es una subdivisión de la IA y del mismo ML, el cual nace gracias al desarrollo de las redes neuronales artificiales Goodfellow et al. [2016]. Las redes neuronales artificiales (ANN) son un conjunto de neuronas que forman una red de comunicación y permite realizar tareas más complejas de análisis e interpretación de información Estas redes neuronales están conformadas por distintas capas: capa de entrada, capas ocultas y una capa de salida. El Aprendizaje profundo como su nombre indica hace referencia a la "profundidad" de una red, debido a la cantidad de capas ocultas con que una red puede estar conformada.

En el AP existen distintos algoritmos de ANN como las CNN, en la figura 2.5 observamos otros tres tipos de algoritmos

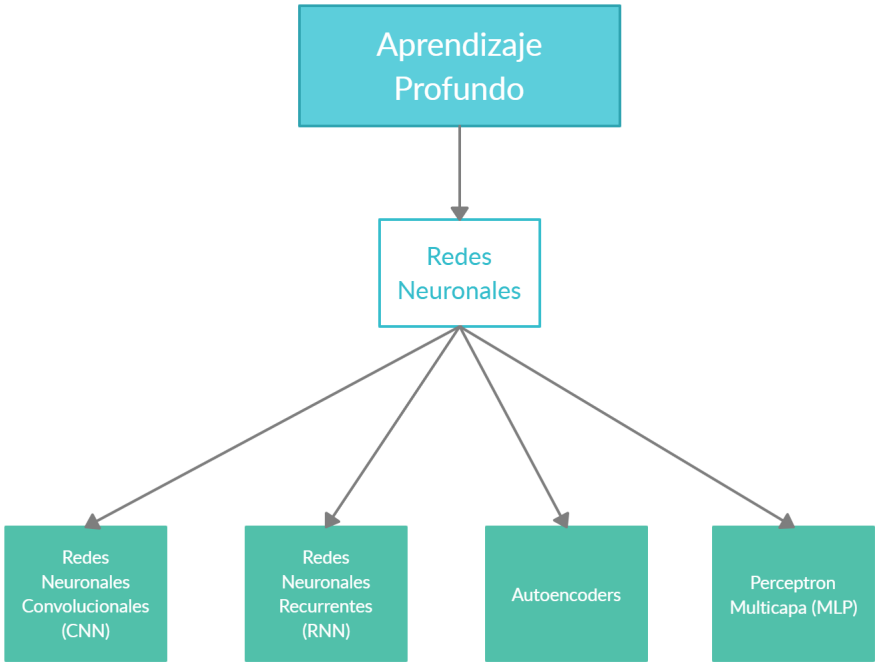


FIGURA 2.5: Algoritmos de Aprendizaje profundo

Para el caso de esta tesis de investigación nos enfocaremos en las CNN.

2.3.1. Redes Neuronales Artificiales

Como es mencionado, ANN son un conjunto de neuronas conectadas entre sí que forman una interconexión capaz de realizar operaciones más complejas que un perceptrón. También es mencionado que las redes se encuentran divididas en capa de entrada, capa oculta y capa de salida, su ejemplificación gráfica se muestra en la figura 2.6. Si tomamos que la función de activación es $f(x) = y$, la ecuación de salida de nuestra ANN de una capa oculta queda como la ecuación 2.1

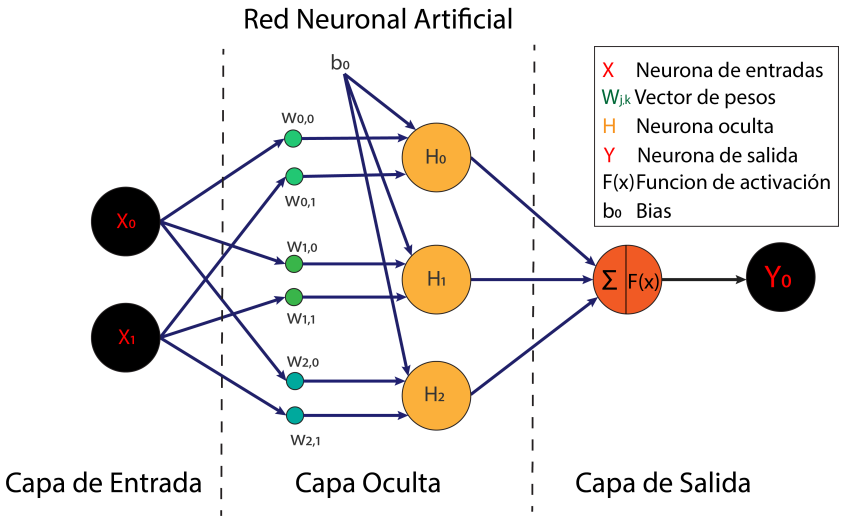


FIGURA 2.6: Red Neurona Artificial

$$Y = \sum (W_{j,k}\, X) + b_0 \tag{2.1}$$

Como se mencionó en una sección anterior, el proceso de aprendizaje de una neurona es debido a la propagación hacia adelante (Feedforward) y hacia atrás (Backpropagation). Tomaremos como ejemplo el diagrama de la figura 2.7 para realizar la explicación de cómo se lleva a cabo el aprendizaje de una red neuronal. Iniciamos con la propagación hacia adelante, este proceso inicia con la multiplicación de los pesos W(inicializados aleatoriamente) por las entradas X más un valor de bias b_0, este último valor tiene como propósito proporcionar mayor libertad de movimiento a la función de la red resultante para adecuarse debidamente a los datos de la base de datos.

$$H_0 = (w_0 \cdot x_0) + (w_1 \cdot x_1) + b_0 \tag{2.2}$$

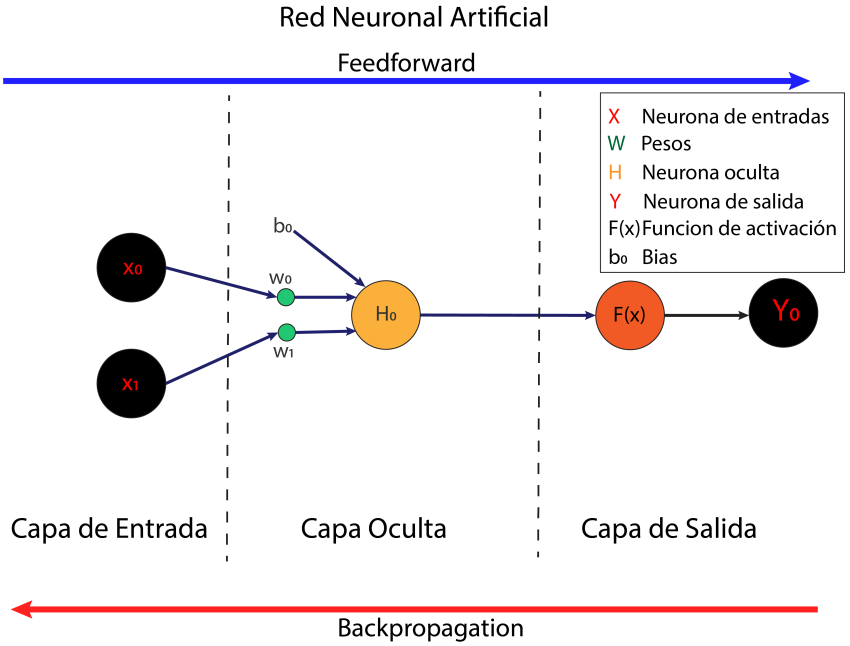


FIGURA 2.7: Red Neurona Artificial

Tomando el resultado de la suma se procede a aplicar la función de activación *F(x)*, el resultado pasará a la salida y_0, como se muestra en ecuación 2.3

$$y_0 = F(H_0) \tag{2.3}$$

Por otro lado, la propagación hacia atrás se realiza mediante la aplicación de una función de transferencia asemejando al de un sistema de comunicaciones, el objetivo principal de Backpropagation es disminuir lo más posible o hasta un nivel definido el error de la red. Existen distintas ecuaciones para encontrar el error como por ejemplo el Mean Square Error (MSE), pero para ejemplificación se utilizará una resta simple del valor predicho Y^p menos el valor deseado Y^t como se muestra en la ecuación 2.4:

$$e = Y^P - Y^T \tag{2.4}$$

Una vez encontrado el o los valores de error, el procedimiento de Backpropagation inicia con la actualización de los pesos de la red, como contamos con solo un par de pesos en el ejemplo solo será necesario una sola ecuación para cada peso, la expresión general para actualizar los pesos se muestra en la ecuación 2.5 :

$$w_i^n = w_i - \eta \cdot \frac{\partial e}{\partial w_i}, \qquad para \quad i = 0, 1 \tag{2.5}$$

Donde η representa la taza de aprendizaje de la ANN y w^n los nuevos pesos.

La ecuación de actualización de los pesos se obtienen de realizar el procedimiento de Backpropagation, donde se busca encontrar la razón de cambio entre el error de salida con los pesos para disminuir este mismo, es por eso que la fracción $\frac{\partial e}{\partial w_i}$ de la ecuación 2.5 es una derivada parcial. A continuación demostramos como se obtienen $\frac{\partial e}{\partial w_i}$ para el caso de w_0

$$\frac{\partial e}{\partial w_0} = \frac{\partial e}{\partial y_0} \cdot \frac{\partial y_0}{\partial H_0} \cdot \frac{\partial H_0}{\partial w_0} \tag{2.6}$$

De la ecuación 2.6 podemos notar como se realiza el Backpropagation, ya que buscamos el cambio con respecto el error entre variables.

2.3.2. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNN) son algoritmos de aprendizaje profundo que aparecieron en la década de los 90's, pero sus inicios empiezan con las investigaciones "Generalization and Network Design Strategies" y "Handwritten Digit Recognition with a Back-Propagation Network " LeCun et al. [1989], LeCun et al. [1990]. En los trabajos iniciales de LeCun et al. antes de que estas redes recibieran su nombre, planteaba el utilizar las ANN para reconocimiento de dígitos, pero sin haber utilizado métodos de extracción de características como los mencionados en los algoritmos no supervisados de ML, sino que el procedimiento se realiza directamente con las neuronas de la red, donde cada neurona se encargaría de celdas de nxn de la imagen de entrada, las neuronas son agrupadas en dimensiones de mxm y las salidas de estas reciben el nombre de mapas de características, los cuales pueden tener compartidos sus pesos, esto se realizaría en capas iniciales para permitir que los mapas de características partan desde las mismas características generales de la imagen, ya que mientras más profunda sea la red se van a extraer características más específicas de la imagen.

La primera aplicación comercial exitosa de las CNN fue la de un sistema de lecturas de cheques desarrollado en AT&T al principio de los años 90's y para finales de la década ya se utilizaba para leer el 10 % de los cheques en Estados Unidos LeCun et al. [2010]. Esta aplicación abrió el interés de muchas instituciones tanto gubernamentales como privadas, tal como es el caso de Microsoft que las

implementó para el reconocimiento de escritura. En la actualidad las CNN se han implementado en una gran cantidad de sistemas y han sido participé en un gran número de investigaciones, que no solo abarcan reconocimiento de letras y números, sino que logran correctamente detectar y clasificar objetos Redmon and Farhadi [2017].

Las CNN han recibido algunos cambios durante el transcurso de los primeros años de su desarrollo hasta el día de hoy, son uno de los algoritmos de IA de mucho auge e interés en la comunidad de investigación debido a la relevancia que estos aún tienen en la actualidad. Las CNN son algoritmos de arquitectura modular ya que cuentan con la posibilidad de poder modificar la cantidad de capas y sus parámetros. La arquitectura general de una CNN la podemos observar en la figura 2.8.

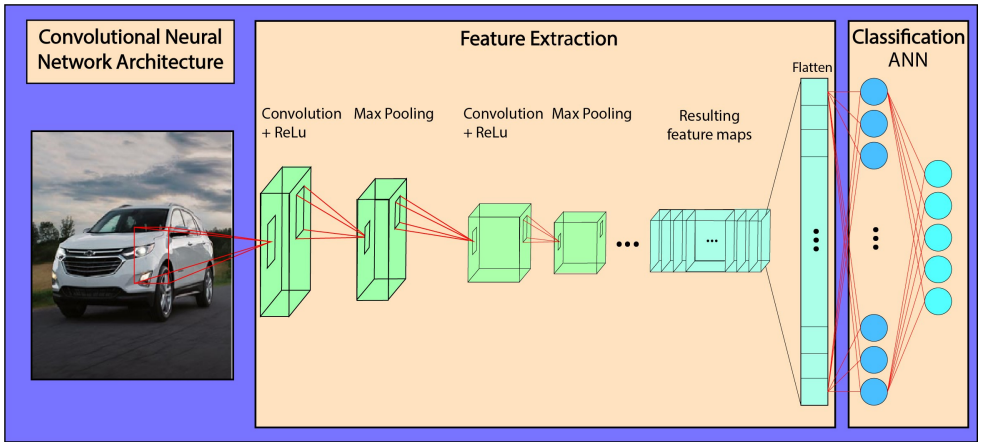


FIGURA 2.8: Red Neuronal Convolucional

La entrada de una CNN pueden ser vectores de 1D para estudios de audio como por ejemplo reconocimiento de voz, vectores de 3D para los tres canales RGB de una imagen en tareas de clasificación de objetos Abdel-Hamid et al. [2014], Sardogan et al. [2018]. La cantidad de dimensiones son definidas por el tipo de trabajo que se querrá realizar. Las capas ocultas de una CNN están compuestas de una capa de filtrado, capa de activación y capa de reducción de dimensionalidad, estas tres etapas en una capa de convolución serían equivalentes a una neurona de la antigua forma en que fueron utilizadas las ANN en LeCun et al. [1990], y por último se tiene una algoritmo de clasificación el cual puede ser una ANN, SVM o Vecinos Cercanos (KNN), aunque comúnmente son utilizadas las ANN.

2.3.2.1. Convolución

La convolución es un método matemático utilizado con gran frecuencia en el área de señales y sistemas, ya que la combinación de dos señales en el dominio del tiempo se realiza mediante el proceso de convolución el cual es denotado por la ecuación 2.7.

$$z[m] \star g[m] = \sum_{n} z[n]g[m-n] \tag{2.7}$$

Donde $z[m]$ es la señal de entrada discreta, $g[m]$ es la función de transferencia del sistema.

La ecuación de convolución 2.7 es específica para señales y sistemas, pero no hay que confundirla con la de IA, esto debido a que en ML se denomina como convolución el realizar una correlación cruzada de los datos en cuestión. La convolución en IA no realiza el paso de inversión de la señal, sino que solamente se realiza la multiplicación de término por término de la ecuación, por lo que para las CNN por ejemplo, la ecuación de convolución correspondería a la ecuación 2.8 la cual contiene doble sumatoria debido a que estas denotan las dimensiones de la entrada X.

$$(I \star X)(i,j) = \sum_{m} \sum_{n} X(m,n)K(i+m, j+n) \tag{2.8}$$

Donde $K_{i,j}$ es el banco de filtros.

2.3.2.2. Filtrado

La etapa de filtrado es donde se realiza la operación de convolución denotada por la ecuación 2.9 de la entrada X con el banco de filtros. Los filtros o también llamados kernels son matrices de tamaño (l_1, l_2) entrenables, las cuales deben embonar correctamente con la entrada X para el procedimiento de la convolución, como se muestra en la figura 2.7. El resultado de la convolución será un vector p_j de dimensiones (m_1, m_2, m_3), donde m_1 es la cantidad de mapas de características resultante y (m_2, m_3) son las dimensiones de largo y ancho de dichos mapas y por último tenemos b_j es el parámetro conocido como bias.

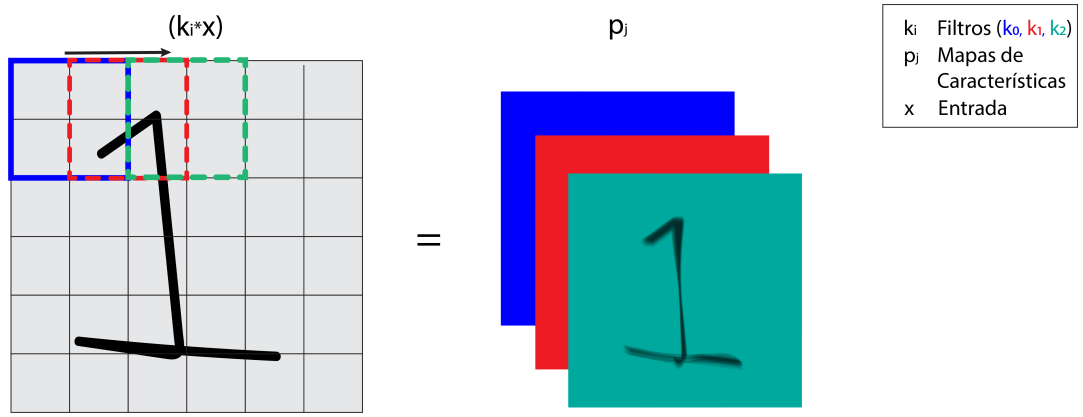
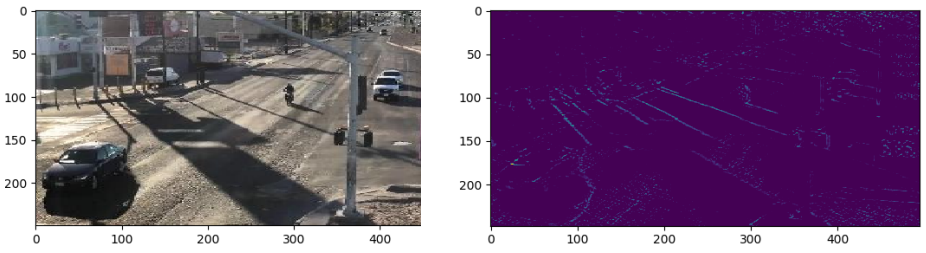


FIGURA 2.9: Procedimiento de convolución

$$p_j = b_j + \sum_i k_{i,j} \star x_i \tag{2.9}$$

La función de los filtros (kernels) es de detectar características particulares de los datos (imagen) y se generaran la misma cantidad de mapas de características que filtros utilizados. Los filtros están compuestos por unos y ceros en sus celdas, pero dependiendo de su acomodo son las características que estos van a encontrar en la imagen, pueden generarse filtros como el operador de Sobel el cual es utilizado para detectar bordes Jeong et al. [2021]. En la figura 2.10 tenemos un ejemplo de mapa de características resultante del proceso de filtrado.



(A) Imagen de entrada (B) Mapa de características

FIGURA 2.10: Demostración del resultado cuando se aplica una capa de convolución

2.3.2.3. Activación

La función de activación indica el comportamiento de salida de la neurona con respecto al resultado obtenido del proceso de filtrado y existen diferentes funciones de activación como por ejemplo la Sigmoide, SoftMax, Tanh, Leaky ReLu, Maxout, entre otras. Cada una de estas funciones tiene sus propias características y son utilizadas para diferentes aplicaciones, en las CNN la función de activación es utilizada para disminuir la no-linealidad.

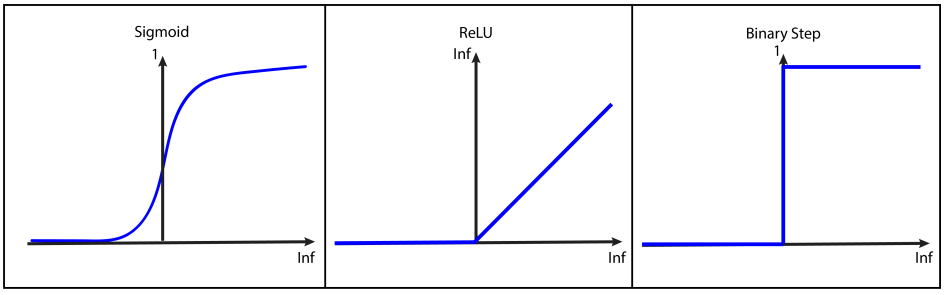


FIGURA 2.11: Funciones de Activación

En la figura 2.11 se pueden observar las funciones de activación Sigmoide, ReLU y Escalon Unitario. En las ecuaciones 2.10, 2.11, 2.12 se muestra la funcion matematica de cada una respectivamente.

$$f(n) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

$$f(n) = max(0, x) \tag{2.11}$$

$$f(n) = \begin{cases} 1, & \text{para } x \geq 0 \\ 0, & \text{otro} \end{cases} \tag{2.12}$$

2.3.2.4. Reducción de Dimensionalidad

La reducción de dimensionalidad (Pooling) es el procedimiento de reducir el tamaño de los mapas de características resultantes. En la mayoría de los casos el pooling ayuda a reducir el impacto de las variaciones de pequeñas traslaciones

dentro de la imagen. Los tipos de pooling más utilizados son el Pooling Máximo (MaxPooling) y Pooling Promediado (AvPooling). En la figura 2.12 podemos observar cómo es el procedimiento del MaxPooling.

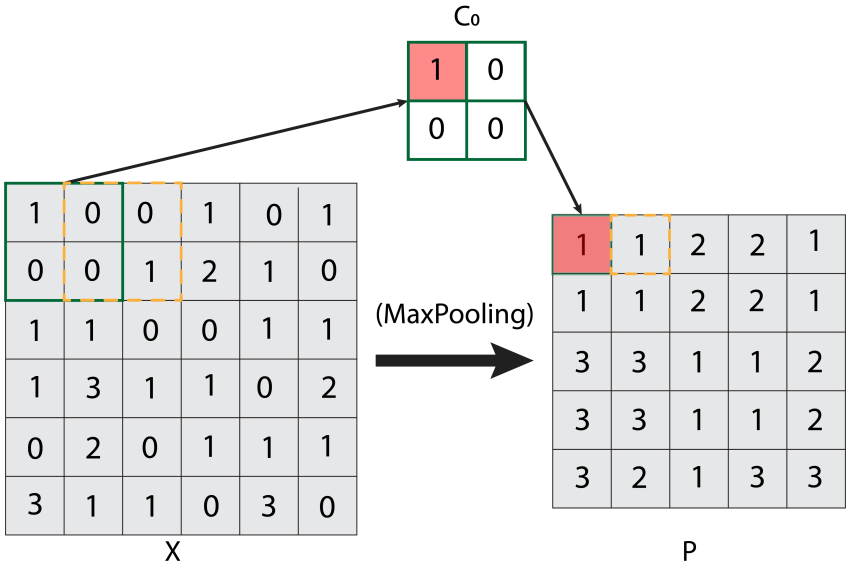


FIGURA 2.12: Ejemplo de MaxPooling

El procedimiento de Pooling puede realizarse de la siguiente manera:

- Establecer el tamaño de la ventana $C_i(m_1, m_2)$, la cual debe tener un número de pasos en que la ventana se mantenga dentro de la imagen de entrada X.
- Establecer los pasos en $(X = s_1, Y = s_2)$ de desplazamiento de la ventana C_i, el desplazamiento se realiza primero por el número de pasos en horizontal hasta llegar al límite permitido y después los pasos en vertical, repitiendo el proceso hasta haber cubierto la imagen por completo.
- Comenzar con el procedimiento de Pooling ya sea Max o AV, iniciando desde la esquina superior izquierda y desplazándose el número de pasos establecidos.
- Terminado las traslaciones de la ventana obtendremos de salida un nuevo mapa de características P reducido en dimensiones.

2.4. Sistemas Embebidos

Durante muchos años el interés de lograr tener una computadora portátil, la cual pudiera ser llevada a todos lados o introducida en dispositivos de reducidas dimensiones han sido de gran interés debido a sus aplicaciones en el campo, por ejemplo el tener un sistema de reconocimiento de placas de vehículos Lee et al. [2017b].

Muy conocidos son los Single Board Computer (SBC), que son computadoras de una sola placa, las cuales ya cuentan con todos los componentes de una computadora normal, pero estos están dedicados a realizar una o algunas tareas específicas. Los SBC están también enfocados para uso tiempo real, por lo que los convierte en dispositivos de interés para tareas de visión de máquina. En años recientes el avance de la tecnología ha permitido que la aplicación de algoritmos de IA para reconocimiento de imágenes sea una realidad, debido a que los SBC de última generación ya cuentan con unidades de procesamiento gráfico (mejor conocidos como GPU's) integradas junto al CPU, RAM y SSD, lo que permite el poder realizar tareas más complicadas de visión ya que un GPU aligeraría la carga al CPU. Existen distintas plataformas de sistemas embebidos como por ejemplo Raspberry Pi, Arduino, Matriz de Compuertas Lógicas programables en campo (FPGA) y Nvidia Jetson, siendo los primeros dos mayormente enfocados para estudiantes dado el enfoque que se le ha dado a los dos ecosistemas y también que son de las más débiles en recursos (memoria interna, RAM, CPU, GPU). Las tarjetas de Nvidia tienen un enfoque un poco más elevado cómo para la investigación, debido a que cuentan con mayor cantidad de recursos (memoria interna, RAM, CPU, GPU) internos. Los algoritmos como CNN se han logrado implementar satisfactoriamente en sistemas embebidos, pero se hace uso de metodologías de optimización o algún procedimiento que agilice los tiempos de ejecución de los algoritmos, esto para lograr implementarlos en tiempo real Alippi et al. [2018].

Por otro lado, de los sistemas embebidos tenemos a los FPGA's los cuales no son una computadora como tal, pero tienen una ventaja considerable sobre los SBC, la cual es la velocidad de ejecución de instrucciones, los FPGA's son extremadamente rápidos y son ampliamente usados donde se requiera adquirir información a alta frecuencia, como: captura de imágenes, reconocimiento de objetos, reconocimiento de voz, entre otras Osio [2019], Hsiao et al. [2015]. Los FPGA's son sistemas embebidos que reciben mucho interés debido a su velocidad de reacción,

pero estos aun no logran una buena relación con algoritmos como las CNN debido a la complejidad de estos, por lo que el uso de SBC sigue teniendo un margen mayor. Aun así, los FPGA's han recibido soluciones para lograr implementar los algoritmos de DL como crear una metodología para aplicar una CNN o implementar un coprocesador para acelerar las Redes Neuronales y manejar varias líneas de transmisión de información Dundar et al. [2016] y Gokhale et al. [2014].

FIGURA 2.13: Ejemplo de SBC: Jetson Nano Developer Kit

Capítulo 3

Procedimiento de Investigación

3.1. Fabricación de la Base de Datos para Clasificación

Una base de datos es lo más importante al momento de entrenar un algoritmo de aprendizaje profundo como lo son las CNN, por lo que se debe de buscar las imágenes adecuadas para llevar a cabo dicho proceso, así que en la presente investigación, para clasificación se realizó un compendio de bases de datos de imágenes con los vehículos que se definieron para clasificar (sedán, motocicleta, camioneta, camión, pickup). En la figura 3.1 se muestra el tipo de imágenes con las que está compuesta la base de datos.

FIGURA 3.1: Ejemplos de vehículos para clasificación

3.1.1. Fabricación de la CNN

Para la fabricación de la CNN, se utilizó el lenguaje de programación ‘’Python”, esto debido a las librerías disponibles y experiencia que ya se contaba con este tipo de lenguaje. Implementar el lenguaje de programación Python para la fabricación de modelos de IA como ANN, CNN, entre otros se realiza de forma que permita una relativa facilidad en la reproducibilidad de los resultados debido a la existencia de librerías como ‘’Tensor Flow” y ‘’Keras”, ya que cuentan con los modelos y procedimientos ya integrados en funciones y que permiten tener una integración más estable entre sí. En la hoja A de anexos se podrá encontrar el código completo creado para la aplicación de clasificación.

La sección de código que nos denota la arquitectura de la CNN desarrollada se muestra en el código 3.1, donde podemos observar la versatilidad con que podemos modificar los parámetros de las capas de nuestra red.

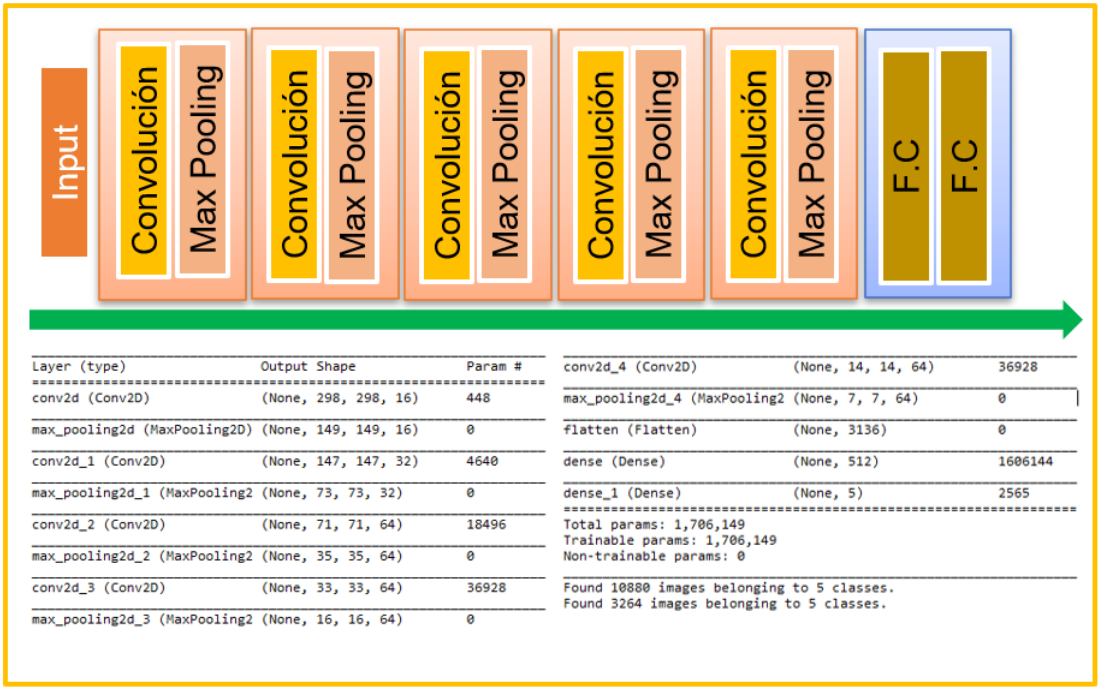
```
model = tf.keras.models.Sequential([
# primera convolucion
tf.keras.layers.Conv2D(16, (3, 3), activation='relu', ...
input_shape=(300, 300, 3)),
tf.keras.layers.MaxPooling2D(2, 2),
# Segunda Convolucion
tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
tf.keras.layers.MaxPooling2D(2, 2),
# Tercera Convolucion
tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
tf.keras.layers.MaxPooling2D(2, 2),
# Cuarta Convolucion
tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
tf.keras.layers.MaxPooling2D(2, 2),
# Quinta convolucion
tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
tf.keras.layers.MaxPooling2D(2, 2),
# aplicamos flat para alimentar la CNN
tf.keras.layers.Flatten(),
# Tenemos 512 neuronas en la primera capa de ANN
tf.keras.layers.Dense(512, activation='relu'),
#  Hay 5 neuronas en la capa de salida, por los 5 tipos de vehiculos
tf.keras.layers.Dense(5, activation='softmax')
])
```

Listing 3.1: Código de la arquitectura de la CNN para clasificación

Analizando el código 3.1, se puede observar que la red cuenta con 5 capas de convolución con ventanas de tamaño de (3x3), la primera capa de convolución

inicia con 16 filtros (kernels), para la capa dos se incrementa al doble y para las siguientes tres capas de convolución aumentamos a 64 la cantidad de filtros existentes. Como se mencionó en la sección 2.3.2.2, los filtros en el proceso de convolución son utilizados para extraer las características (información) de las imágenes creando lo que conocemos como "mapas de características", pero hay que tener en consideración la cantidad de filtros que podremos utilizar dadas las especificaciones de memoria con las que cuenta la computadora en uso. Por esta razón se puede observar que se utilizó una red más esbelta que una ''VGG16" o ''Alexnet" para implementar el clasificador en un sistema embebido. Las funciones de activación de cada capa convolucional son "Rectifier Linear unit" (ReLu), esto porque son las de mayor uso y porque no se necesita tener activaciones tangenciales o absolutas, sino, que el resultado sea lineal y exista solo desde cero hasta $y = x$, esta función se puede observar en la figura 2.11.

En la figura 3.2 podemos observar de manera gráfica como está compuesta la red CNN y también la cantidad de parámetros entrenables con los que esta cuenta.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 298, 298, 16)	448
max_pooling2d (MaxPooling2D)	(None, 149, 149, 16)	0
conv2d_1 (Conv2D)	(None, 147, 147, 32)	4640
max_pooling2d_1 (MaxPooling2	(None, 73, 73, 32)	0
conv2d_2 (Conv2D)	(None, 71, 71, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 35, 35, 64)	0
conv2d_3 (Conv2D)	(None, 33, 33, 64)	36928
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 64)	0
conv2d_4 (Conv2D)	(None, 14, 14, 64)	36928
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 512)	1606144
dense_1 (Dense)	(None, 5)	2565

Total params: 1,706,149
Trainable params: 1,706,149
Non-trainable params: 0

Found 10880 images belonging to 5 classes.
Found 3264 images belonging to 5 classes.

FIGURA 3.2: CNN Desarrollada

Por último en la fabricación de la red se definió utilizar el porcentaje de relación 80/20 de la base de datos para el proceso de entrenamiento de la red, 80 % para entrenamiento y 20 % para validación. La validación es enseñarle a la red entrenada después de cada época de entrenamiento nuevas imágenes para medir

su porcentaje de exactitud ante nuevas imágenes que no forman parte del entrenamiento. También se estableció un fin de entrenamiento cuando el porcentaje de exactitud alcance el 95 %, además de tener un límite de épocas, el cual es de 200.

3.1.2. Metodología para Detección de Objetos

Como se ha mencionado en la sección 2.3.2, las CNN son algoritmos de IA de mucho poder y con una gran cantidad de aplicaciones, donde usualmente son utilizadas en tareas de clasificación de objetos, pero en años recientes hemos visto el desarrollo de distintas arquitecturas no solo para clasificar, sino también para realizar el proceso de detección de objetos, como por ejemplo están: You Only Look Once (YOLO) y las Redes Neuronales Convolucionales Regionales (RCNN). Estos algoritmos de detección de objetos se han implementado satisfactoriamente y varias investigaciones los han tomado para distintas aplicaciones Redmon et al. [2016], Girshick et al. [2014]. Para esta investigación se decidió tratar de replicar la metodología de YOLO para la detección de vehículos, debido a las ventajas que esta tiene en comparación con RCNN, a continuación, se describira un poco más a detalle cómo es que funcionan estos dos algoritmos.

3.1.2.1. RCNN

Las Redes Neuronales Convolucionales Regionales (RCNN), son algoritmos creados para realizar las funciones de clasificación y detección. Si bien la clasificación ya se ha analizado como se realiza con las CNN, la detección es un procedimiento distinto debido a que se utilizan regiones de la imagen para detectar objetos. Este proceso es realizado en tres partes, la primera es la selección de regiones mediante un algoritmo de selección, la segunda es pasar todas las regiones propuestas por capas de convolución para extraer los mapas de características, por último, se implementa un SVM para la clasificación de las características recabadas por las capas de convolución. La figura 3.3 muestra gráficamente este procedimiento.

Si bien cuando las RCNN surgieron lograron resultados importantes en el campo de detección, ya que lograban acercar la brecha entre la detección y clasificación, aún estaban lejos para un uso práctico por la lentitud que estas tienen debido a que se analiza una gran cantidad de regiones con posibles objetos, aunado a esto la cantidad de información requerida ya que se debe tener un algoritmo que aprenda

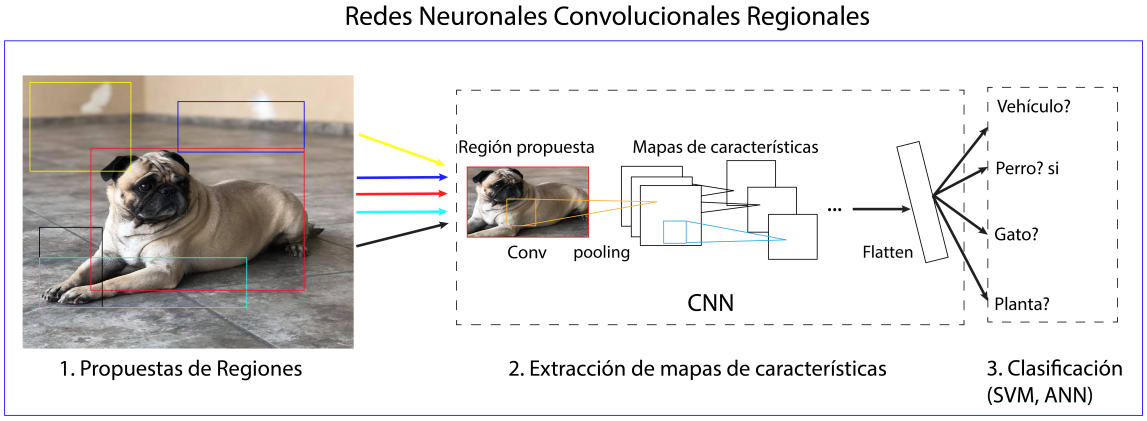


FIGURA 3.3: RCNN

a proponer regiones con un nivel de probabilidad aceptable y el entrenar el algoritmo de clasificación (SVM, ANN, etc.), debido a estas limitantes se desarrollaron versiones que tienen un tiempo de operación más corto, las cuales son Fast-RCNN y Faster-RCNN, que operan bajo el mismo principio de clasificar y detectar por regiones de la imagen Girshick [2015], Ren et al. [2015].

3.1.2.2. You Only Look Once

El algoritmo de "You Only Look Once" (YOLO), es un tipo de arquitectura que basa su metodología de detección de una manera diferente a la de RCNN y desde su primera versión obtiene buenos resultados y en tiempos de ejecución mucho menores en comparación con otras redes, esto porque YOLO es una CNN que su diferencia más notable es la de utilizar su propia función de error para el "Back-propagation" de la red. Otra característica de YOLO es que analiza las imágenes completas sin recortar o seleccionar alguna región. YOLO divide la imagen en una cuadrícula de (S, S) celdas, y esta división es mantenida hasta llegar a la ANN de la CNN. La ANN analiza todas las celdas en paralelo y a la salida entrega un volumen de dimensiones $(S, S, (B * 5 + C))$, donde S es la cantidad de celdas por lado, B la cantidad de "Bounding Boxes" por cada celda y C la cantidad de clases existentes para clasificación.

El proceso de YOLO se observa en la figura 3.4, donde encontramos que se realiza todo el proceso normal de una CNN pero a la salida obtenemos una matriz de resultados. YOLO usa como premisa que cada celda es "responsable " de detectar a los objetos dentro de ella, por ello estas se tratan como un vector de tamaño $(B * 5 + C)$, donde ya conocemos B y C pero el número 5 denota los parámetros

de un Bounding Box (centro (x,y), dimensiones (h,w) del Bounding Box y el nivel confianza). Para ejemplificar el caso de solo tener un Bounding Box, definimos $B = 1$, lo que indica que solo tendremos un Bounding Box por celda, lo que nos permitirá representar cada una como el siguiente vector mostrado en la figura 3.5.

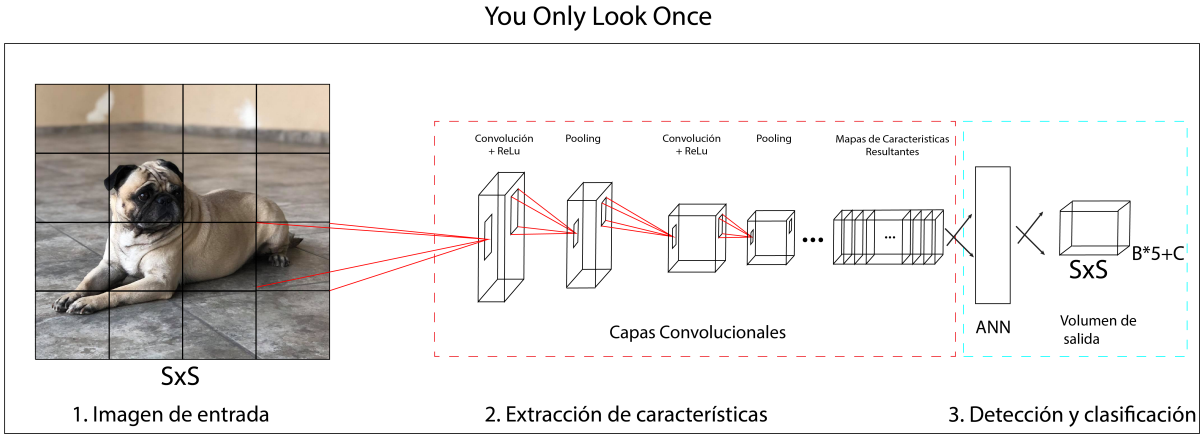


FIGURA 3.4: Arquitectura general para el algoritmo de YOLO

$$S_i = \begin{bmatrix} p \\ x \\ y \\ h \\ w \\ [C] \end{bmatrix}$$

FIGURA 3.5: Formato de visualización de una celda S_i

Donde:

- p Es el nivel de confianza que tiene la celda en si existe o no un objeto en ella.
- x Coordenada del centro del Bounding Box en el eje X.
- y Coordenada del centro del Bounding Box en el eje Y.
- w Ancho del Bounding Box.
- h Alto del Bounding Box.
- $[C]$ Vector que contiene las distintas clases existentes.

Como se ha mencionado, la forma en que YOLO realiza el proceso de detección es por la división de celdas y hacer cada una de estas "responsables" de su detección, pero para ello se utiliza una función de error específica, pero sencilla. Esta función de error está compuesta por cuatro secciones, cada una denota un tipo de parámetro a calcular. La ecuación 3.1 denota el cálculo del error del centro del objeto (E_c), la ecuación 3.2 denota el error en cuanto a las dimensiones de la caja (Bounding Box) del objeto (E_d), la ecuación 3.3 denota el error de confianza que existirá al definir si existe o no un objeto presente en la celda (E_{cf}) y por último se tiene la ecuacion 3.4 la cual calcula el error para el tipo de clase (E_{cl}) del objeto detectado.

$$E_c = \lambda_{co} \sum_{i=0}^{s^2} \sum_{j=0}^{B} ¶_{ij}^{obj} [(x_i - \tilde{x_i})^2 + (y_i - \tilde{y_i})^2] \tag{3.1}$$

$$E_d = \lambda_{co} \sum_{i=0}^{s^2} \sum_{j=0}^{B} ¶_{ij}^{obj} [(\sqrt{x_i} - \sqrt{\tilde{x_i}})^2 + (\sqrt{y_i} - \sqrt{\tilde{y_i}})^2] \tag{3.2}$$

$$E_{cf} = \sum_{i=0}^{s^2} \sum_{j=0}^{B} ¶_{ij}^{obj} [(PC_i - \tilde{PC_i})^2] + \lambda_{nbj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} ¶_{ij}^{nbj} [(PC_i - \tilde{PC_i})^2] \tag{3.3}$$

$$E_{cl} = \sum_{i=0}^{s^2} \sum_{c \epsilon clases} ¶_{ij}^{obj} [(p_i(c) - \tilde{p_i}(c))^2] \tag{3.4}$$

Donde:

- λ_{co} Es un parámetro de ajuste para evitar fortalecer al gradiente de entrenamiento.
- λ_{nbj} Es un parámetro utilizado para evitar que los valores de las celdas no sean cero cuando no exista objeto.
- $¶_{ij}^{obj}$ Parámetro que denota un uno cuando exista objeto en el Bounding Box j de la celda i.
- $¶_{ij}^{nbj}$ Parámetro que denota un uno cuando no existe objeto en el Bounding Box j de la celda i.

- $p_i(c)$ Probabilidad de clase en la celda i.
- PC_i Valor de confianza en la celda i.

Los valores con tilde son los datos predichos por la red y los que no cuentan con tilde son los valores reales.

YOLO ha demostrado ser un algoritmo poderoso a la hora de aplicarse en tiempo real, ya que se ha comprobado que puede trabajar arriba de los 30 cuadros por segundo (FPS) como lo indica Redmon et al. [2016], es por eso que se ha decidido hacer uso de esta metodología de detección en una red con características similares a la de clasificación para aplicarla en sistemas de bajos recursos, porque en donde YOLO ha sido implementado han sido en sistemas con equipo de cómputo de alto rendimiento, con recursos suficientes para redes de gran profundidad, en donde YOLO logra obtener alrededor de 45 FPS.

3.2. Fabricación de la Base de Datos para Detección

Como fue mencionado se implementará la metodología de YOLO para la red de detección, pero para ello se debe generar una nueva base de datos, proceso que requiere una gran cantidad de tiempo y para ello se ha decidido fabricar distintos programas auxiliares en el lenguaje de programación gráfico de ‘’LabVIEW”. En las siguientes secciones mencionaremos cada uno y su funcionamiento.

Previo a la fabricación de la base de datos se analizó que para obtener suficientes imágenes se debería realizar grabaciones a una altura pertinente y a un grado de inclinación que permitiera capturar la mayor cantidad de objetos, que para el caso de esta investigación son vehículos, por lo que se decidió realizar las grabaciones desde un puente peatonal en la ciudad de Mexicali. Posteriormente, se tiene que llevar a cabo un proceso de edición de los videos para delimitar el área correcta a analizar, esto debido al ángulo de visión que la cámara tiene.

Una vez realizado este proceso para obtener los videos para uso de los programas en LabVIEW se da comienzo con la creación de la base de datos. El equipo utilizado para realizar esta investigación es el siguiente: una cámara Fujifilm FinePix HS20EXR, el programa de edición de video Sony Vegas Pro 2015 y por

último la computadora utilizada para el desarrollo y experimentación de todos los programas es una Laptop Dell con un Intel Core i5, 16gb de RAM y 1 Terabyte de memoria interna.

3.2.1. Programa para creación de la Base de Datos

El "Dataset Creation Program" (DCP) es un programa desarrollado para extraer cuadros (imagen) de un video donde se cumpla con los criterios siguientes:

- Vehículo visible y claro.
- No existe solapamiento entre vehículos.
- Debe de encontrarse al menos la mitad del área vehículo dentro de una celda de la cuadrícula establecida.
- Evitar que existan partes de vehículos visibles por las orillas de la imagen.

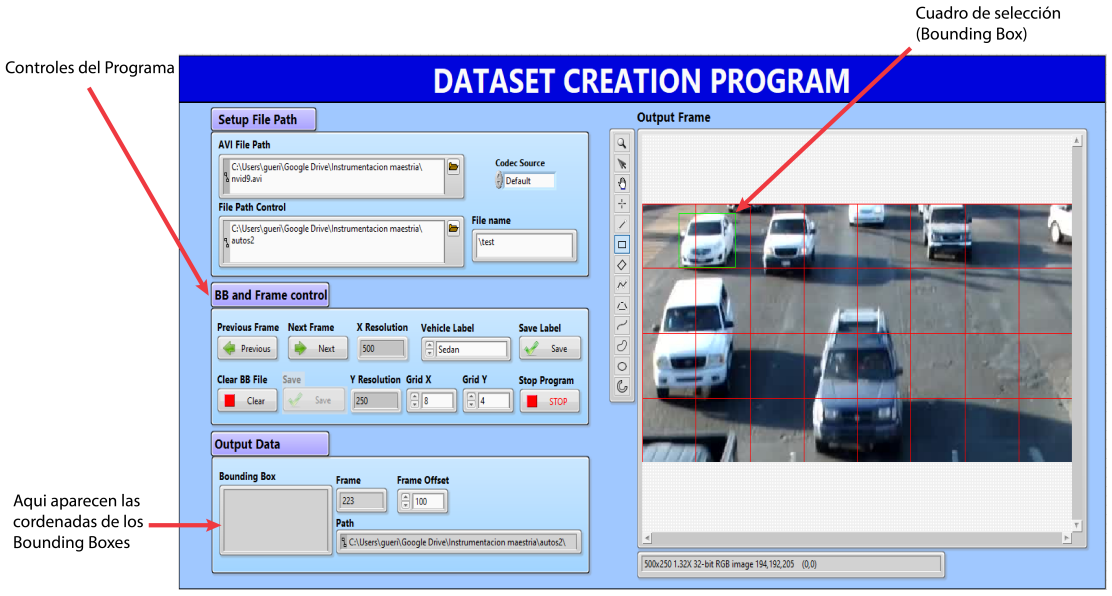


FIGURA 3.6: DCP, programa para la creación de la base de datos donde se observa tener una sección de inicialización, control y datos de salida.

Los criterios establecidos son para garantizar que exista un vehículo por celda o al menos el centro del Bounding Box de ese vehículo se encuentre dentro de dicha celda, esto debido a la complejidad que se podría tener al momento de realizar el entrenamiento. De igual forma, se ha decidido solamente utilizar un Bounding

Box por celda, lo que significa que solo es posible detectar un vehículo a la vez por celda, esto porque el tener más de un Bounding Box por celda implica aumentar sustancialmente la profundidad de una red CNN. Así que, se debe tomar en cuenta al momento de realizar la base de datos el tener solamente un vehículo por celda. En la figura 3.6 se observa la interfaz de usuario del programa DCP.

Este programa generará imágenes de 500x250 pixeles, si bien YOLO maneja resoluciones cuadradas se ha decidido manejar una resolución rectangular debido a como resultaban las grabaciones del cruce vehicular, ya que al tomar una resolución cuadrada no era completamente indicado debido al ángulo de visión de la cámara utilizada, esto porque si se quiere capturar una región especifica del cruce vehicular se termina con muchos vehículos recortados por los laterales de la imagen al momento de que estos vayan saliendo del campo de visión. Así que elegir esta resolución de 500x250 permite observar de cerca y a su vez que existan menos casos de vehículos recortados. En la figura 3.7 se puede observar un ejemplo de cómo es una imagen de la base de datos y las áreas de revisión: la división en los carriles delimitada por las líneas punteadas rojas y en los triángulos formados en ambos lados de la imagen donde se tiene una difusión para delimitar la zona que se debe evitar para capturar vehículos debido a que en su mayoría no cumplirían con los criterios establecidos.

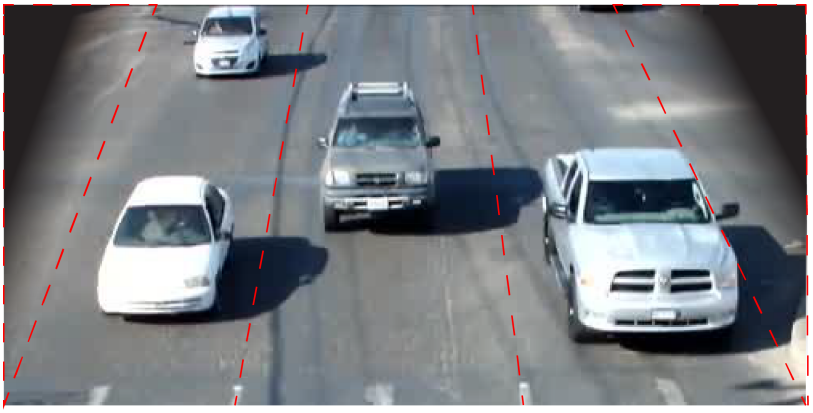


FIGURA 3.7: Ejemplificación de las zonas para seleccionar a los vehículos

El DCP además de guardar una imagen con los vehículos para la base de datos, también generará un archivo de texto por cada imagen y ambos llevarán el mismo nombre al momento de guardarlos, esto para evitar cualquier error al momento de cargar los datos a la red. En los archivos de texto encontraremos la siguiente información: El punto inicial (P_1) y final (P_2) que crean el recuadro de selección

(Bounding Box), el tipo de clase a la que pertenece el vehículo en cuestión y por último la resolución de la imagen en análisis.

3.2.2. Programa de Verificación

El programa para verificación es creado para realizar una revisión a las imágenes de la base de datos y sus archivos de texto con la información de los Bounding Boxes. Este programa se aplica después del DCP, ya que este programa de verificación renombra los archivos de la siguiente manera "nombre+#", el número será en orden ascendente empezando desde el uno. Utilizar este programa es importante, ya que si no se renombran los archivos los demás programas no los identificarán. La interfaz de usuario puede observarse en la figura 3.8, donde solo hay dos entradas de control, el botón "Next" el cual es para cambiar a la siguiente imagen y el indicador de control "Offset" que se encarga de continuar con la numeración correcta dado que es muy probable que la base de datos se cree por partes. El indicador "Bounding Box Data" muestra los datos de la caja los cuales son punto inicial (primeras dos casillas), punto final (siguiente dos casillas) y por último la clase a la que pertenece el objeto delimitado por dicha caja (Bounding Box).

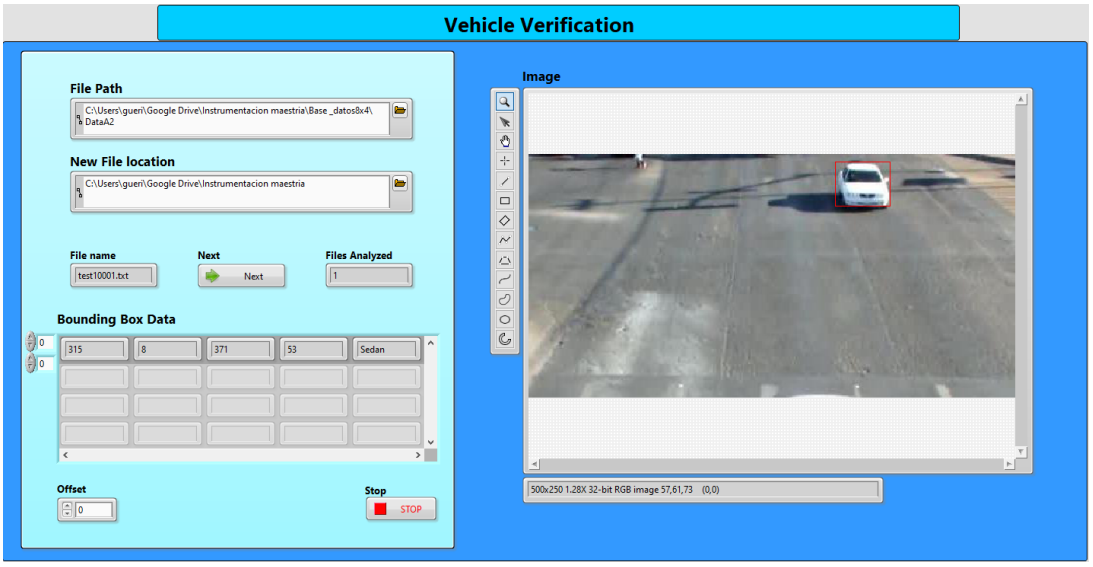


FIGURA 3.8: Interfaz de usuario del "Verification Program"

3.2.3. Programa para el Aumento de Datos

Posteriormente de haber utilizado el programa de verificación, se creó un programa para hacer manualmente la aumentación de la base de datos. Este programa de aumentación de datos (DA) puede aplicar tres acciones sencillas a las imágenes, estas acciones son el desplazamiento en los ejes horizontal y vertical, además de aplicar el efecto espejo a la imagen, con estas tres se pueden crear combinaciones y así incrementar la base de datos. La interfaz de usuario del programa de DA se observa en la figura 3.9

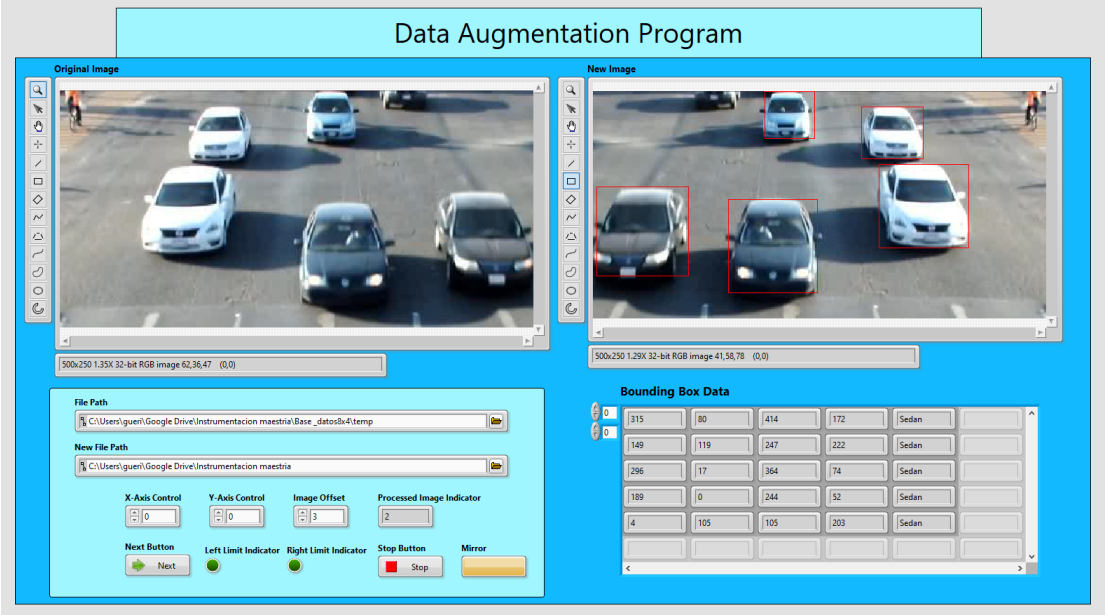


FIGURA 3.9: Programa de aumentación de datos, se muestra un ejemplo donde se aplica el efecto espejo a la imagen

En el ejemplo de la figura 3.9 se puede observar cómo se aplica el efecto de espejo a la imagen y el mismo programa de DA se encarga de acondicionar los valores originales de los Bounding Boxes para que estos coincidan con la nueva imagen creada, con este programa se ha incrementado la base de datos hasta las 16 mil imágenes, combinando los tres efectos que tiene este programa.

3.2.4. Programa de Acondicionamiento de los Bounding Boxes

Los programas anteriores sirven para crear las imágenes y los archivos de texto que contienen la información de los Bounding Boxes, pero solo se registran los

puntos (P_1) y (P_2) de dichas cajas (como referencia visual de los puntos (P_1) y (P_2) ver la figura 3.10), por lo que hay que extraer de estas medidas las variables que se necesitan para el entrenamiento de la red, las cuales son las coordenadas del centro $C(x, y)$ y dimensiones $D(h, w)$, el nivel de confianza registrado ser de valor de uno en la celda en la que el centro del Bounding Box del vehículo se encuentre.

FIGURA 3.10: Ejemplo para calcular (x, y), (h,w)

Observando la figura 3.10 se puede identificar que tenemos un Bounding Box creado por dos puntos $\tilde{P}_1(x_1, y_1)$ y $\tilde{P}_2(x_2, y_2)$ con los cuales podemos realizar los siguientes cálculos para encontrar $C(x, y)$ y $D(h, w)$:

$$D(h, w) = \tilde{P}_2 - \tilde{P}_1 \tag{3.5}$$

$$C(x, y) = \frac{(\tilde{P}_1 + \tilde{P}_2)}{2} \tag{3.6}$$

Una vez encontrado el centro y las dimensiones se procede a normalizar dichos valores (debido a que los datos que manejan las CNN tienen un rango de cero a uno), w con respecto a la resolución en el eje horizontal (500) y h con respecto a la resolución vertical (250), para el centro del Bounding Box se tiene que normalizar con respecto a la dimensión de la celda (S_i) en la que se encuentra, por lo que es necesario el haber definido el tamaño de la cuadrícula (S, S), por lo que de manera general se usa la siguiente ecuación 3.7 para normalizar.

$$C_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.7}$$

Donde X es un punto interno con coordenadas (x, y) en pixeles, de la celda S_i.

Al haber normalizado todos los datos, el programa los guarda en un nuevo archivo el cual llevará el mismo nombre al de la imagen con la que deben coincidir los datos y se almacenarán en una nueva carpeta juntos, esto para mantener un orden de parejas de archivos. La interfaz de usuario del programa para acondicionamiento de los datos de los Bounding Boxes se muestra en la figura 3.11.

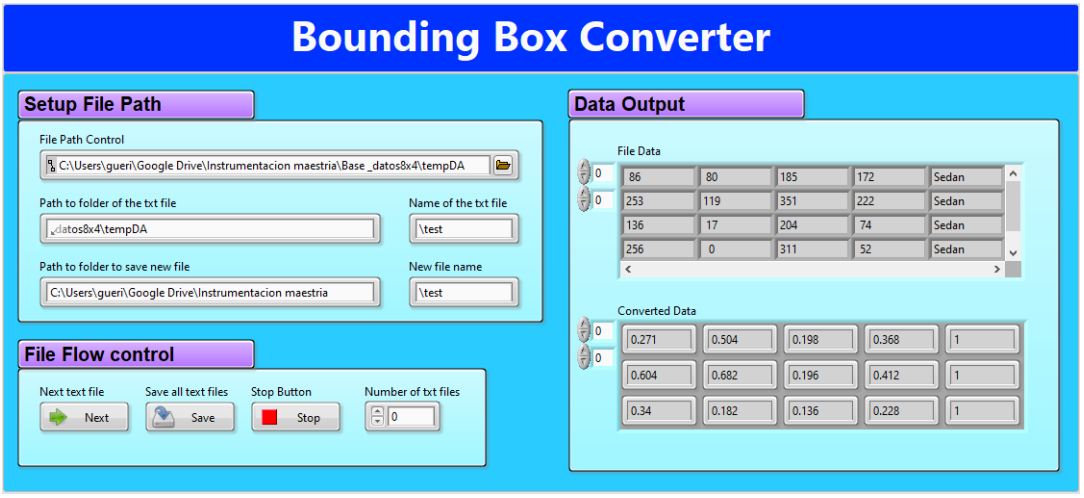


FIGURA 3.11: Interfaz de usuario para el acondicionamiento de datos de los Bounding Boxes

3.2.5. Red de Detección

El código para la red de detección se puede ver en la hoja B de anexos, a continuación observamos el código 3.2 el cual contiene la arquitectura de dicha red.

```
model = tf.keras.models.Sequential([
# primera convolucion
tf.keras.layers.Conv2D(64, (3, 6), input_shape=(250, 500, 3), activation='relu',
                       strides=(2, 2), name='conv1'),
tf.keras.layers.MaxPooling2D(2, 2),
# Segunda Convolucion
tf.keras.layers.Conv2D(128, (3, 6), strides=(2, 2), activation='relu', name='conv2'),
tf.keras.layers.MaxPooling2D(2, 2),
# Tercera Convolucion
tf.keras.layers.Conv2D(256, (3, 6), strides=(2, 2), activation='relu', name='conv3'),
tf.keras.layers.MaxPooling2D(2, 2),
# aplicamos flat para alimentar la ANN
tf.keras.layers.Flatten(),
# Utilizamos 3 capas ocultas de 1024 neuronas cada una
tf.keras.layers.Dense(1024, activation='relu'),
tf.keras.layers.Dense(1024, activation='relu'),
```

```
tf.keras.layers.Dense(1024, activation='relu'),
tf.keras.layers.Dense(grid_x * grid_y * (N_box), activation='linear'),
# Esta capa de reestructuracion es para acomodar nuestra salida en el volumen
# de datos SxSx(B*5+C)
tf.keras.layers.Reshape([grid_y, grid_x, (N_box)], name='last')
])
```

LISTING 3.2: Código de la arquitectura de la CNN para detección

En este código 3.2 se puede observar que se disminuyó la red a solo 3 capas de convolución, pero a su vez se incrementaron las capas del ANN junto con la cantidad de neuronas, en comparación con la red para clasificación mostrada en la figura 3.2. La diferencia más notoria en cuanto a códigos entre la CNN de detección y de clasificación, viene dado por el hecho de manejar dos formas distintas de crear el flujo de imágenes y datos hacia la red, ya que se debió de realizar de manera manual para la detección debido a que se debían cargar ambas partes juntas (Imagen y Bounding Box), pero aunado a esto se debía de realizar un proceso específico para la extracción de los Bounding Boxes, el cual correspondía a tener que formar por cada imagen su volumen de datos correspondiente, con las mismas dimensiones $(S, S, (B * 5 + C))$ con las que la red en el código 3.2 entrega, esto para realizar el proceso de Backpropagation. La red de detección en el código 4.2 muestra que la resolución definida es de (250, 500) y por consecuencia para mantener la relación de los mapas de características se utiliza una ventana de convolución de (3, 6). Esta red utiliza la sección de detección de objeto en la celda la cual corresponde a la ecuación 3.3 de la función de error completa de YOLO.

3.2.6. Programas de Prueba de CNN

Como parte de la presente tesis, se han desarrollado dos programas para poner a prueba las redes creadas para detección y clasificación con una transmisión de video, utilizando una función existente en LabVIEW 2018 y posteriores, dicha función lleva por nombre "Python Node", esta función permite correr códigos de Python en LabVIEW. Cabe destacar que si bien el Python Node permite correr códigos de Python este tiene un detalle en específico, el cual es solo poder cargar funciones de Python, ya que la función de LabVIEW necesita tener especificado el tipo de dato que será regresado al terminar la ejecución del código de Python.

El programa de prueba para clasificación que lleva por nombre "Sistema de Reconocimiento Vehicular" (SRV) es creado para poner a prueba la velocidad de

ejecución de la red de clasificación desarrollada, además de poder contar con una interfaz de usuario. El funcionamiento básico de este programa es el de enviar una transmisión de video capturado por la cámara de la computadora a la función de LabVIEW la cual carga el modelo de CNN de clasificación para realizar la predicción de la clase del vehículo en la imagen, luego el resultado de la predicción es regresado a LabVIEW para mostrarlo.

En la figura 3.12 se tiene la interfaz de usuario del programa de prueba para clasificación, la cual cuenta con una sección de inicialización (Setup) y dos secciones para la salida de resultados, la primera seccion "Tiempo de Procesado" proporciona la información sobre la cantidad de cuadros por segundo (FPS) que son procesados y el tiempo en que se tardó en procesar cada uno, la segunda sección "Reconocimiento de Vehículos" indica el resultado de la predicción de la CNN.

FIGURA 3.12: Interfaz de usuario para el Sistema de Reconocimiento Vehicular, donde se observa que se está capturando una imagen de prueba de un vehículo de clase sedán y es confirmado por la CNN de clasificación

El segundo programa de prueba es para el algoritmo de detección, el cual trata de simular la aplicación en campo para detectar vehículos que circulan por una vialidad, por lo que lo hace muy similar al SRC en cuanto funcionamiento. Los cambios que se realizaron fueron los de sustituir los VI's para realizar la captura de video por VI's encargados de realizar la lectura de un archivo de video, este cambio nos impide usar el VI para ver los FPS procesados directamente, otro cambio es el dato de salida de la red la cual ahora es una matriz de datos con las dimensiones establecidas para el algoritmo de detección (S, S, (B*5+C)). Aunque no se pueda

analizar los FPS directamente desde un VI existe una forma alternativa, la cual es el analizar el tiempo de procesado que le toma a LabVIEW junto con el "Python Node" en analizar una imagen del video, esta medición se encuentra en el indicador de salida "Tiempos de Procesado".

La figura 3.13 muestra la interfaz de usuario donde indica que el nombre de este programa es el de "Sistema de Detección Vehicular", esta interfaz esta compuesta por la sección inicialización (Setup), la sección de "Tiempos de Procesado" y por último sección de "Resultados de Red". El indicador de imagen "Detección de Vehículo" nos muestra el video analizado junto con la detección del vehículo la cual la vemos representada por la activación de las celdas en color rojo en la figura 3.13

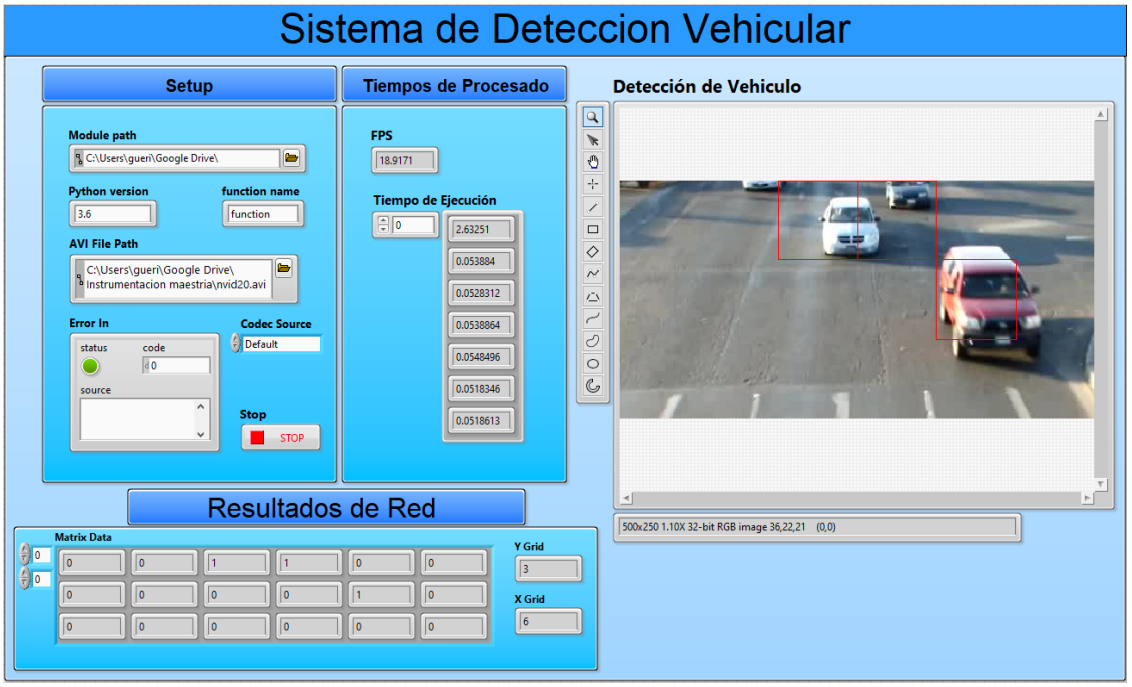


FIGURA 3.13: Interfaz de usuario para el Sistema de Detección Vehícular, en la imagen se observan activadas las celdas correspondientes al centro del vehículo detectado

Capítulo 4

Análisis de Resultados

4.0.1. Análisis de Clasificación

El proceso para encontrar una red adecuada y ligera en términos de consumo de recursos computacionales se llevó a cabo en dos partes: en un análisis de literatura y la realización de pruebas de CNN's. En la literatura encontramos un manejo común de desarrollo de las CNN, el cual es replicar redes existentes y realizar cambios a su estructura o el utilizar la misma red para realizar lo que se conoce como un "Fine Tuning". Este proceso es una práctica para entrenar la red CNN con un número específico de objetos, esto para que aumente su porcentaje de clasificación, pero solo para esos objetos seleccionados. Sin embargo, esta práctica de desarrollo no es lo ideal debido a que las CNN utilizadas son redes comprobadas que manejan buen nivel de clasificación superior al 90 %, pero su profundidad las hace requerir una mayor cantidad de recursos en memoria para su entrenamiento. Debido a esto, se decidió desarrollar desde cero la CNN, definiendo el número de capas de convolución, el tipo de Pooling y por último el tamaño de la ANN. Esto porque se busca poder utilizarla en tiempo real y lograr su aplicación en un sistema embebido. El establecer que se quiere utilizar en tiempo real, se indica que esta red debe procesar al menos una imagen cada 0.05 segundos, para mantenerse superior a los 20 FPS.

Durante el desarrollo de la red de clasificación se obtuvo un modelo el cual su arquitectura se observa en la figura 3.2 y lograbá superar el 95 % de exactitud en el entrenamiento. Debido a este resultado positivo se decidió utilizarlo como modelo base, pensando en tener una arquitectura poco profunda para optimizar los

recursos de procesamiento. El modelo base es nombrado "Modelo 1" y en base a este se desarrollaron 5 modelos similares, los cuales fueron utilizados para analizar el impacto que tienen los elementos de las CNN sobre los tiempos de entrenamiento, además de ver como son afectados los porcentajes de clasificación tanto en entrenamiento como validación, esto para identificar si el Modelo 1 podía obtener una mejora en su rendimiento haciendo modificación del número de parámetros y levemente la arquitectura. Los resultados de estos modelos se muestran en la tabla 4.1, donde observamos la exactitud (entrenamiento y validación), tiempos de entrenamiento y cantidad de épocas realizadas, además de utilizar Aumentación de Datos (DA). Como se mencionó, para los entrenamientos de las redes se estableció un límite de exactitud para detener el entrenamiento, el cual es de 95 %, es por esto que vemos que todos los modelos en la tabla 4.1 manejan valores cercanos en "Acc. Entrenamiento".

Tabla 4.1: Tabla de modelos entrenados

	Acc. Entrenamiento	Acc. Validación	D.A.	Épocas	T. de Entrenamiento
Modelo 1	0.9662	0.7647	no	10	<3hrs
Modelo 2	0.9507	0.8137	si	50	<4hrs
Modelo 3	0.9339	0.8199	si	200	40hrs
Modelo 4	0.9524	0.8327	si	172	33hrs
Modelo 5	0.9511	0.8392	si	187	62hrs
Modelo 6	0.9512	0.8382	si	172	58hrs

Para complementar la información de la tabla 4.1, en la tabla 4.2 se muestran las especificaciones internas (filtros, pooling y neuronas) de la estructura de cada modelo.

Tabla 4.2: Tabla de Arquitecturas de los modelos CNN

	Convolution	Max-Pooling	Convolution	Max-Pooling	Convolution	Max-Pooling	Convolution	Max-Pooling	Convolution	Max-Pooling	Dense	Dense
Modelo 1	16,(300x300)	2,2	32	2,2	64	2,2	64	2,2	64	2,2	512	5
Modelo 2	16,(300x300)	2,2	32	2,2	64	2,2	64	2,2	64	2,2	512	5
Modelo 3	16,(300x300)	2,2	32	2,2	64	2,2	64	2,2	64	2,2	49	5
Modelo 4	16,(300x300)	2,2	32	2,2	64	2,2	64	2,4	128	2,2	512	5
Modelo 5	2x(16,300x300)	2,2	32	2,2	64	2,2	64	2,2	64	2,2	512	5
Modelo 6	2x(16,300x300)	2,2	32	2,2	64	2,2	64	2,2	128	2,2	1024	5

Con la información disponible por las tablas 4.1 y 4.2, se puede analizar como fue el comportamiento de la red original con los cambios realizados. Iniciamos con el Modelo 2, este utiliza exactamente los mismos valores de filtros y neuronas que el Modelo 1, con la diferencia de que para el Modelo 2 se utilizó DA para analizar los cambios en los porcentajes de exactitud que se obtendrían y observamos que se obtuvo un 1.6 % menor en la exactitud de entrenamiento, pero un incremento en validación el cual fue aproximadamente de un 5 %. Este incremento, si bien es un resultado positivo, pero vino acompañado de un incremento moderado en

el tiempo de entrenamiento, pasando de un tiempo menor a tres horas a tardar alrededor de cuatro horas, por lo que esto nos indica que el uso de DA es una buena práctica para mejorar el rendimiento de la red sin afectar seriamente el tiempo de entrenamiento.

Observando la tabla 4.2, la arquitectura del Modelo 3 nos indica que para este modelo vemos disminuido el valor de neuronas en la primera capa de la ANN, se pasa de 512 a tener solo 49 neuronas. El disminuir la cantidad de neuronas ocasiona un incremento sustancial en el tiempo de entrenamiento ya que pasó de menos de tres horas a 40 horas, lo que permitió que se llegara al límite de épocas definidas para los entrenamientos de todos los modelos, la cual era de 200 épocas, por lo que no es recomendable el utilizar un número bajo de neuronas en la CNN. Para el Modelo 4 se realiza una modificación similar, en este modelo solo se incrementa la cantidad de filtros en la última capa de convolución, pasa de tener 64 a 128 filtros. Con esta modificación vemos que este modelo en la tabla 4.1 su tiempo de entrenamiento y de épocas realizadas aumenta considerablemente en comparación con el Modelo 1, el tiempo de entrenamiento aumenta más de 10 veces y el número épocas en 17.2 veces. Lo que esto nos indica que el incrementar la cantidad de filtros en una sola capa puede afectar considerablemente los tiempos de entrenamiento.

Para el Modelo 5 se agrega una capa de convolución adicional al inicio de la estructura y se encuentra que el tiempo de entrenamiento se ve incrementado nuevamente, casi el doble que el Modelo 4 lo que nos indica que el agregar más capas afecta en mayor medida que agregar más filtros a la red. Por último, el Modelo 6 es un caso donde se combina el incremento de filtros del Modelo 4 junto con la capa de convolución extra al inicio de la estructura del Modelo 5, además de incrementar al doble las neuronas de la ANN como se observa en la tabla 4.2, con estos cambios se obtuvo un porcentaje de exactitud similar con una disminución en el tiempo de entrenamientos comparado con el Modelo 5. Con estos resultados se identificó que el tiempo de entrenamiento no fue aumentado considerablemente en comparación de los modelos 4 y 5, ya que para el Modelo 6 se esperaba encontrar tiempos más extensos de entrenamiento al incluir más parámetros en su estructura.

Por último en el análisis de los modelos de clasificación se dará una vista a las gráficas de los comportamientos de las redes tanto en entrenamiento como validación. Las figuras 4.1 y 4.2 corresponden al entrenamiento de las redes y se observa como fue su evolución con respecto al transcurso de las épocas realizadas.

En la figura 4.1 la cual corresponde a la medición de la exactitud y la figura 4.2 al error calculado con el MSE se puede observar a más detalle como fue el entrenamiento para cada modelo, se logra apreciar cómo se vieron afectados los modelos por los cambios realizados, en todos los casos se ve como se incrementa el número de épocas necesarias para llegar al 95 % de exactitud, pero la diferencia destacable entre los entrenamientos es el tiempo que le tomó a cada uno llegar a ese porcentaje de exactitud, los cuales se observan en la tabla 4.1.

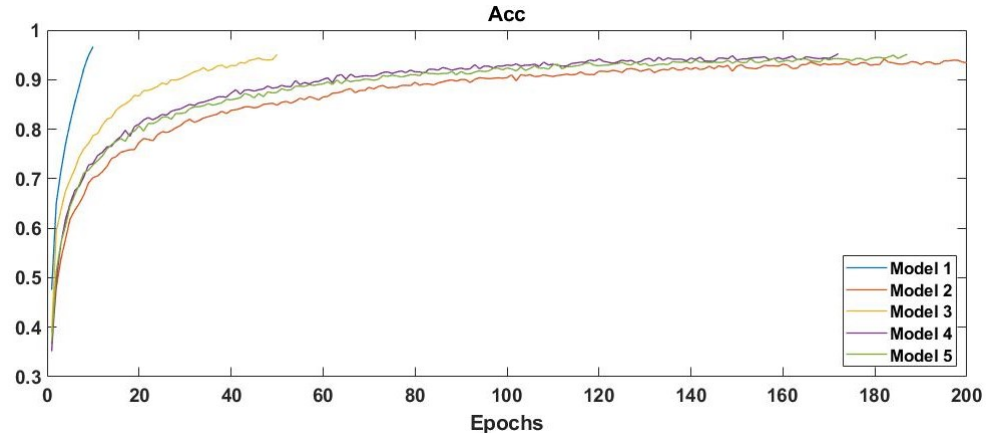


FIGURA 4.1: Exactitud de Entrenamiento

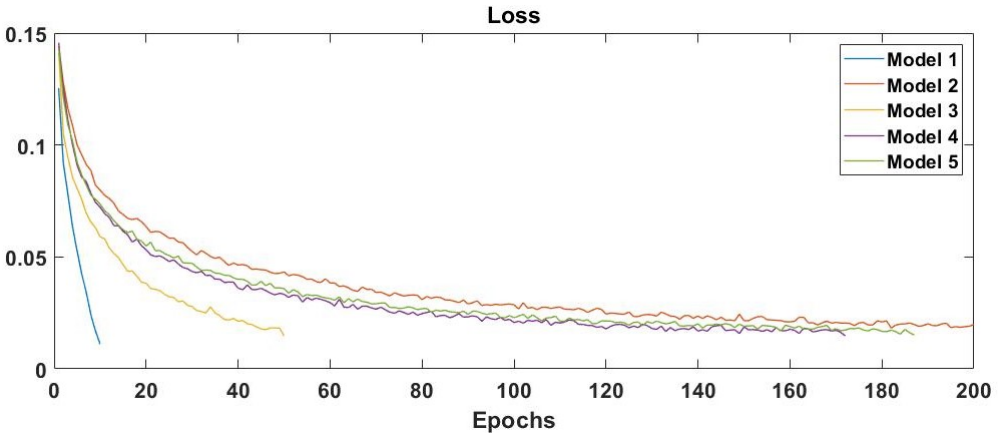


FIGURA 4.2: Error de entrenamiento calculado por el MSE

En el caso de la validación se observa el comportamiento de estas redes en cuanto a observar imágenes no incluidas en su proceso de entrenamiento, cabe hacer mención que la validación es un proceso para poner a prueba las redes neuronales para conocer sus porcentajes de exactitud ante nuevos datos nunca antes vistos. Por lo que se puede observar en las figuras 4.3 y 4.4 que las líneas de cada modelo de exactitud y de error de validación tienen variaciones más pronunciadas en

comparación con las de entrenamiento (figuras 4.1 y 4.2), esto indica que el sistema aprende bien del conjunto de imágenes de entrenamiento, pero se le dificulta tratar de clasificar nuevas imágenes, pero aún así se logra incrementar su valor de exactitud en validación hasta un 83 %.

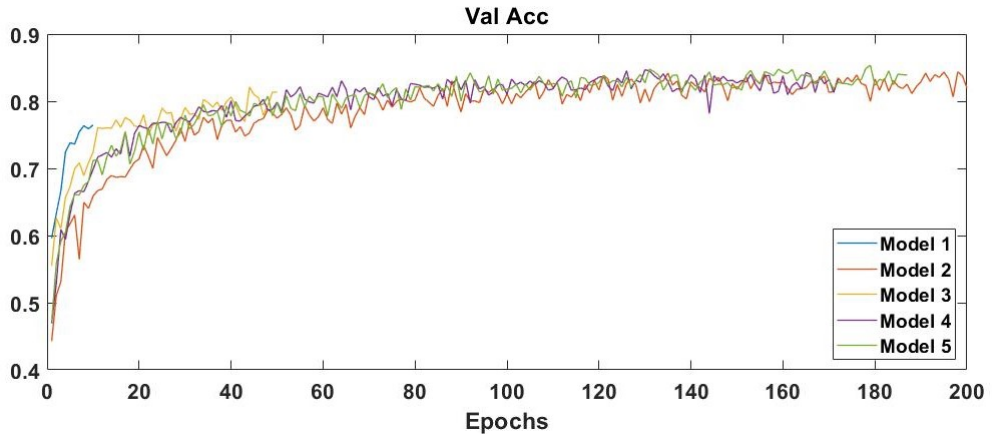


FIGURA 4.3: Exactitud de Validación

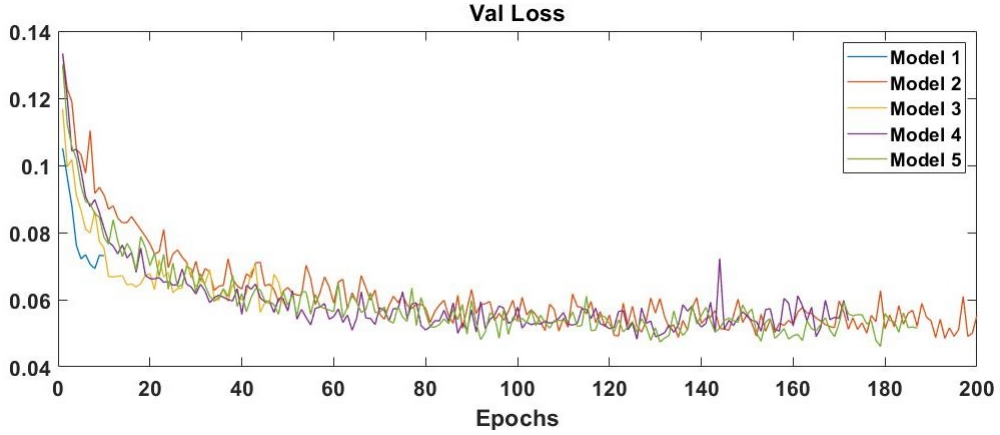


FIGURA 4.4: Error de validación calculado por el MSE

Para analizar correctamente el funcionamiento de la red de clasificación se hace uso de la herramienta conocida como matriz de confusión (Confusion Matrix). Esta herramienta nos muestra los aciertos y errores de clasificación en una cuadrícula la cual se puede observar en la figura 4.5, en esta matriz de confusión podemos observar una prueba realizada de 30 imágenes nuevas de distintos vehículos y lo que hace es comparar las clases verdaderas (True labels) contra las clases predichas (Predicted label), se utiliza una escala de color para denotar el caso con mayor número de aciertos, que en este caso es el color elegido es azul y la clase con

un mayor número de aciertos es la de "Autos" ya que se puede identificar por el número y la tonalidad del color.

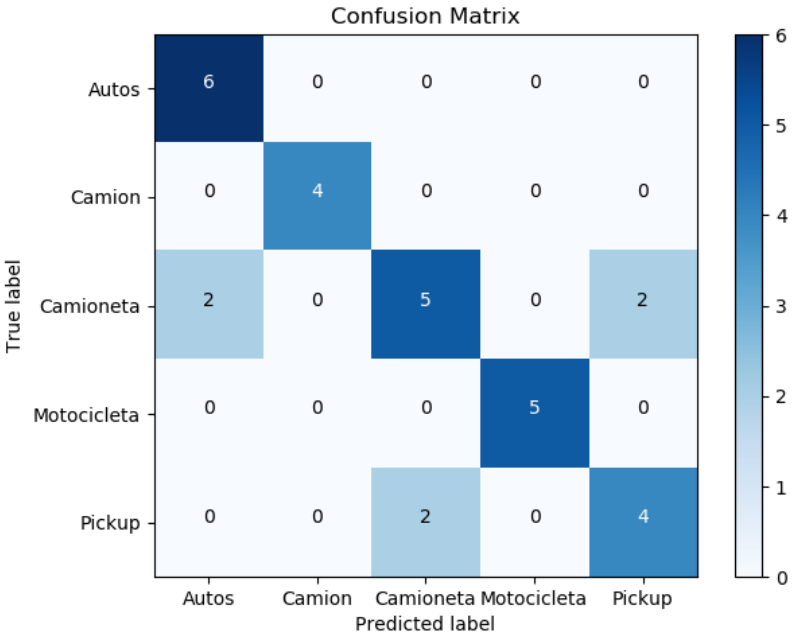


FIGURA 4.5: Matriz de Confusión.

Si se desea tener información más detallada del rendimiento de la red se puede hacer uso de la función de Python "metrics.classification_report()" la cual proporciona información como precisión, exhaustividad (recall), puntaje f1 (f1-score) y exactitud de la red. En la tabla 4.3 y 4.4 se muestran los resultados de aplicar la función mencionada.

TABLA 4.3: Tabla de mediciones de rendimiento de la red CNN de clasificación

	Precisión	Exhaustividad	f1	# vehículos
Auto	0.75	1.0	0.86	6
Camión	1.0	1.0	1.0	4
Camioneta	0.71	0.56	0.63	9
Motocicleta	1.0	1.0	1.0	5
Pickup	0.67	0.67	0.67	6

La precisión de la red está dada por la relación que existe entre los verdaderos positivos y el total de clasificaciones positivas (verdaderos positivos más falsos positivos) de la clasificación de las imágenes de prueba. La Exhaustividad (recall) es una métrica para medir qué tan bien la red logra predecir una "clase" contemplando los verdaderos positivos y falsos negativos de las predicciones únicamente.

Ahora, el puntaje f1 (f1-score) es una métrica que asemeja a obtener una media entre la precisión y la exhaustividad debido a que en algunos casos no es posible decidir cual de las dos mediciones es la más importante a considerar, por lo que en esos casos se emplea el f1-score.

La tabla 4.4 muestra los valores promediados de la red donde se puede ver que la prueba con 30 imágenes nuevas, da como resultado un 83 % de precisión y un 80 % de exactitud, lo que ayuda a corroborar el resultado de entrenamiento de la red de clasificación.

Tabla 4.4: Tabla de mediciones promediadas y de exactitud total del rendimiento de la red CNN de clasificación

	Precisión	Exhaustividad	f1	Exactitud	# Vehículos
Exactitud				0.80	30
Macro avg	0.83	0.84	0.83		30

El último resultado de las pruebas es el de la velocidad de ejecución de la red, la cual se probó en una computadora portátil y se obtuvo una velocidad aproximada de 20 cuadros por segundo (FPS), lo que permite que este algoritmo pueda trabajar tiempo real, porque para uso en un sistema embebido aún se necesitan ciertas mejoras de código y uso de librerías específicas ya que el sistema embebido utilizado es el Jetson Nano de Nvidia el cual tienen un GPU de 128 núcleos, pero en la prueba realizada no se logró utilizar el GPU lo que nos hizo ver mermados los resultados del tiempo de ejecución por imagen.

4.0.2. Análisis de Detección

Como fue mencionado en el capítulo 3, la metodología a utilizar para lograr la detección de vehículos es la de YOLO. Para el desarrollo de la red de detección se decidió implementar por secciones la función de error que YOLO utiliza, por lo que en primera instancia se inició con la función de error para el nivel de confianza de la ecuación 3.3. De igual forma, en el capítulo 3 se mencionó que se estaría utilizando una resolución de imagen de (500, 250), esto para lograr tener un campo de visión más amplio para la vialidad y limitarlo a solo una porción del cruce vehicular, debido a esta resolución al momento de seleccionar el tamaño de la cuadrícula de YOLO, se elegirá (S, S/2), esto para utilizar celdas cuadradas. Los resultados obtenidos de la red de detección se muestran en las figuras 4.6 y 4.7.

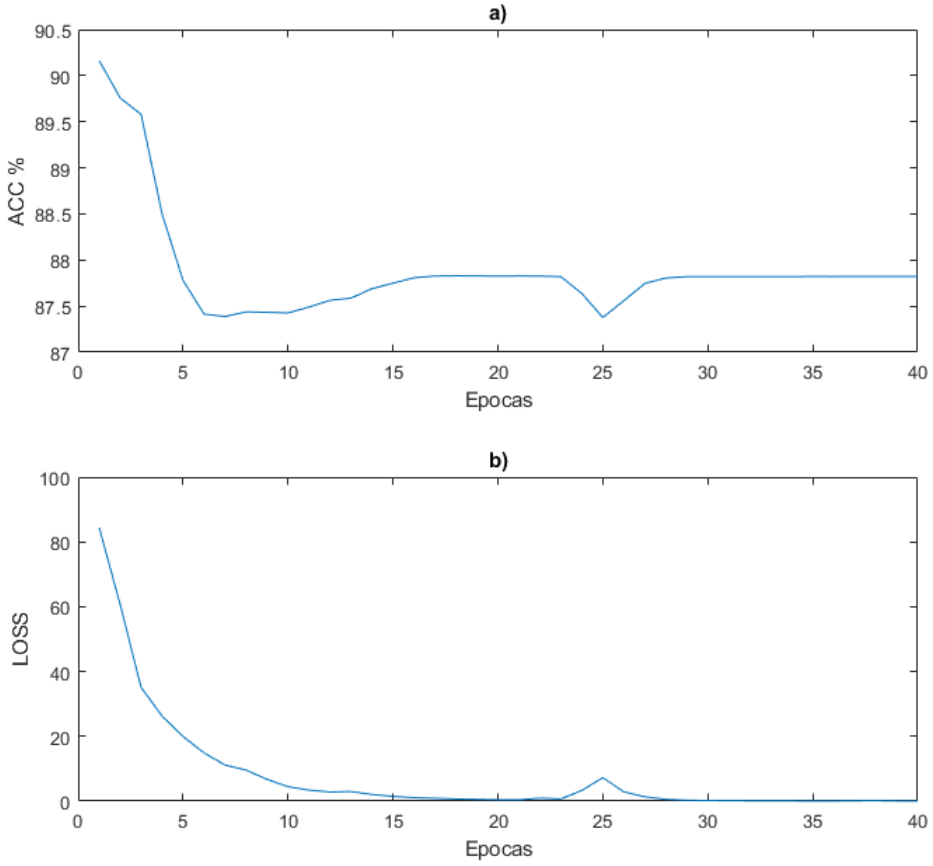


FIGURA 4.6: Resultados de Entrenamiento: *a*)Exactitud (ACC), *b*)Error (LOSS).

Los resultados de la figura 4.6 *a*) , muestran como la red desarrollada se comporta durante el entrenamiento, si se compara con los de la red de clasificación (4.1), vemos que para detección se comienza desde las primeras épocas en un valor elevado de exactitud (90 %) y después solamente disminuye hasta un 87.8 %. Esto se atribuye a que en muchos casos las imágenes de entrenamiento y validación contarán con pocos vehículos, por lo que la red estará analizando la mayoría de las celdas con ceros, lo que genera que virtualmente la red está funcionando correctamente en la detección de objetos. Los resultados de error se aprecian en la figura 4.7 *b*), donde se observa como la red empieza a aprender la información proporcionada ya que vemos como el valor de error disminuye considerablemente.

La figura 4.7 muestra las gráficas de Exactitud y Error de validación de la red y en ella se observa como el comportamiento de los resultados entre ambas figuras 4.6 y 4.7 es parecido, ya que se logra apreciar como el valor de exactitud en la

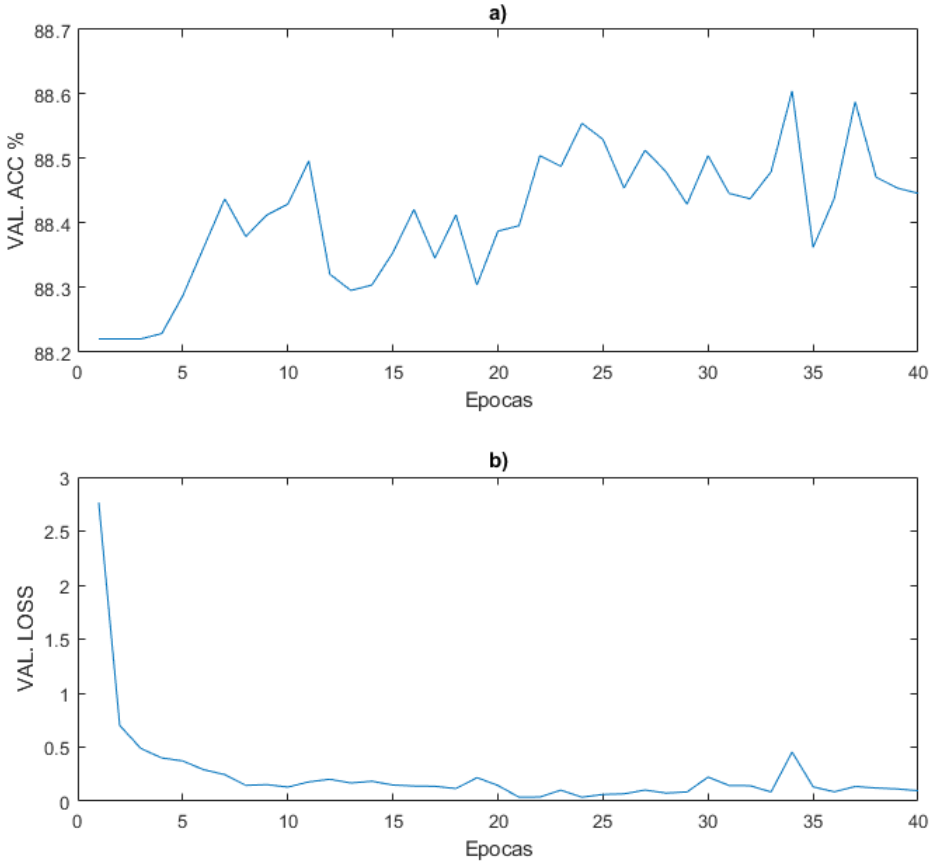


FIGURA 4.7: Resultados de Validación: *a*)Exactitud (ACC), *b*)Error (LOSS).

figura 4.7 *a*) se mantuvo casi estático, moviendose algunas milésimas desde que inicio hasta la última época. La figura 4.7 *b*) muestra el valor de error y aunque no es tan elevado como la del entrenamiento 4.6 *b*), muestra un comportamiento similar y se logra disminuir considerablemente hasta un valor de 0.096 para la última época.

Los resultados mostrados son de la red ganadora, debido que durante el desarrollo de esta red se realizaron distintas modificaciones a la función de error para lograr que su funcionamiento fuera el correcto, ya que si bien los resultados se mantenían siempre con valores elevados de exactitud y bajos para el error, cuando la red era colocada a prueba se identificaba que existían celdas que siempre eran activadas aún sin tener vehículo dentro de ella, además de encontrar limitaciones de entrenamiento ya que se tuvo que modificar la razón de aprendizaje (Learning rate), debido a que se presentaban cambios grandes en el valor de error, lo que permitía la existencia de un fenómeno conocido como el gradiente explosivo,

el cual sucede cuando el gradiente de error comienza a aumentar sin control. Se logró disminuir las ocasiones en las que este fenómeno se presentaba penalizando de distinta manera a la ecuación de error de YOLO, ya que se le dio más peso al error calculado de las celdas que no cuenten con un vehículo en comparación con las que sí llegan a tener un vehículo dentro.

Para poner a prueba la red de detección se utilizó el programa SDV mostrado en la sección 3.2.6 y del cual se extrajeron las figuras 4.8 y 4.9 mientras este programa se encontraba en funcionamiento para demostrar que la red puede realizar la activación de las celdas cuando detecta a un vehículo dentro de la misma.

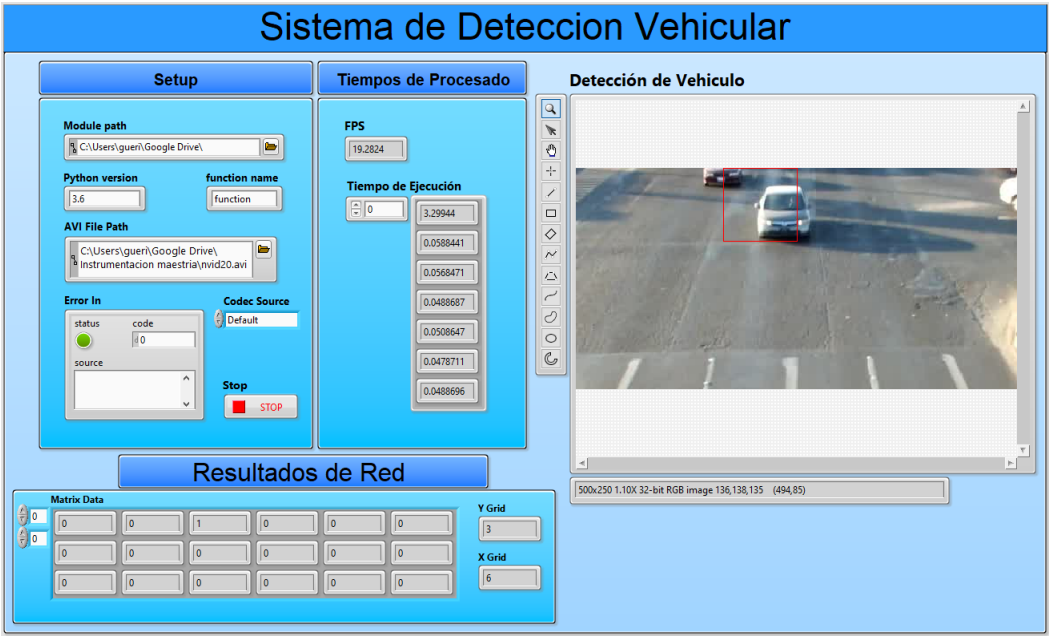


FIGURA 4.8: Prueba de la detección por parte de una de las celdas de la red

En la figura 4.8 se aprecia como la mayoría del área del vehículo está dentro de la celda y por consecuente esta es activada señalando que allí se encuentra un vehículo. Para la figura 4.9 el video continúa alrededor de dos segundos para mostrar como la siguiente celda es capaz de detectar al vehículo una vez que este entra en su mayoría a dicha celda, además de que ingresa un nuevo vehículo a la imagen y es detectado por su celda correspondiente.

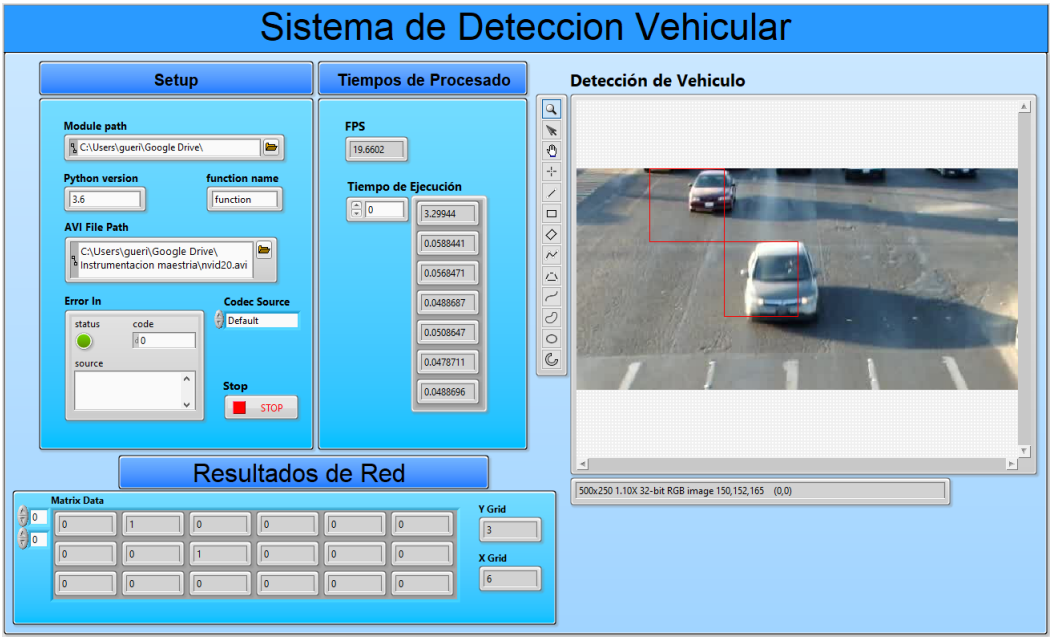


FIGURA 4.9: Continuación de la prueba de detección.

Las dos figuras 4.8 y 4.9 muestran los casos similares con los que la red está entrenada, donde se tienen vehículos separados y que coinciden casi en su totalidad en una celda, pero las figuras 4.10 y 4.11 se tiene los casos para múltiples vehículos y se observa como el algoritmo demuestra la detección de objetos, pero con ciertos detalles, ya que si bien la red resulto con un 88 % de exactitud. Esta aún tiene campo de mejora debido a que existirán casos como en las figuras 4.10 y 4.11 donde las celdas con partes de distintos vehículos podrán activarse debido a la cantidad de área que estos ocupen dentro de la celda, además de casos donde un vehículo puede estar dividido en varias celdas y solo una de estas se activará debido a que la CNN está entrenada para activar celdas cuando se encuentre la mayor parte del vehículo.

Como se mencionó la red obtuvo un 88 % de exactitud de validación y además se mostraron las figuras del SDV donde se demuestra comó hace la detección, lo siguiente a analizar es el tiempo de ejecución de la red. En cada figura 4.8, 4.9, 4.10 y 4.11 se muestra la sección "Tiempos de Procesado" donde vemos los FPS a los que el programa se encuentra funcionando y se observa que en todos los casos se está logrando llegar aproximadamente a 20 cuadros por segundo lo que es un resultado positivo ya que se logra trabajar en "tiempo real", siendo aplicado en una computadora portátil. Ahora, si bien la idea es ver su posible aplicación en un sistema embebido la red se introdujo al Jetson Nano de Nvidia para identificar

FIGURA 4.10: Continuación del video de la prueba de detección

los tiempos de procesamiento y se encontró que la red de detección logra procesar aproximadamente un cuadro del video cada 0.27 segundos lo que indica 4 cuadros por segundo aproximadamente, lo cual es un resultado positivo ya que todo el procesamiento se está realizando en el CPU del sistema embebido y no en su GPU. Por lo que si se aplica en el GPU la red de detección se lograría obtener un mayor número de FPS para usarlo en tiempo real.

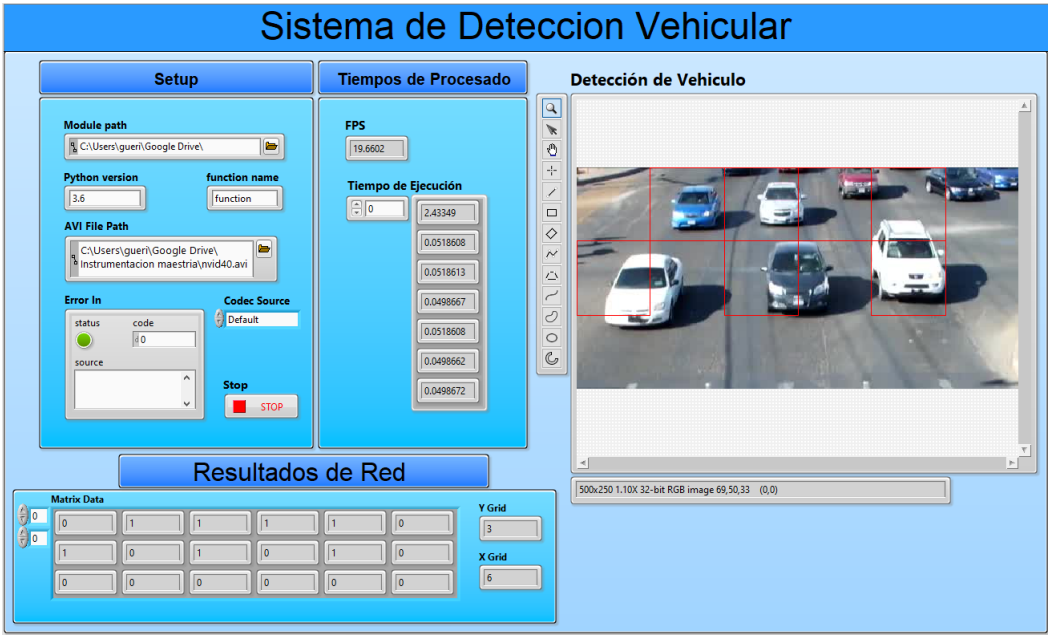


FIGURA 4.11: Finalización de la prueba de detección.

Capítulo 5

Conclusión

En la presente tesis se trabajó para desarrollar dos algoritmos de IA para las tareas de clasificación y detección de vehículos, estos dos algoritmos desarrollados pasaron por una serie de procesos y cambios para lograr obtener los resultados deseados y a continuación se dará mención de cada red:

La red de clasificación desarrollada fue diseñada sin contar con alguna arquitectura previamente definida o algún tipo de método de desarrollo proporcionado por literatura existente, por lo que se comenzó por hacer un análisis literario de las investigaciones recientes sobre las CNN y se encontró que si bien, existen algunas propuestas metodológicas, no existe un estándar universal para el desarrollo de este tipo de redes así que mediante lo analizado se comenzó con el diseño de una red de pocas capas para analizar su comportamiento. Con este proceso de prueba de redes CNN se logró encontrar una red la cual era lo suficientemente capaz de aprender las distintas clasificaciones de los vehículos (5 tipos) con un 95 % de exactitud durante su entrenamiento y un 78 % de exactitud en validación, sin hacer uso de una práctica conocida como "Data Augmentation". En base al modelo preliminar desarrollado (Modelo 1) se decidió alterar los parámetros de esta red y utilizar la aumentación de datos para probar la capacidad de crecimiento de rendimiento, por lo que se implementaron 5 modelos distintos donde cada uno solo veía modificado un parámetro en específico (filtros, neuronas o hasta una capa de convolución extra) con el fin de incrementar la exactitud de la red. Estos cambios demostraron aumentar ligeramente esta métrica, pero se obtuvo con una desventaja, la cual es haber incrementado considerablemente los tiempos de entrenamiento de la red. De estos 5 modelos adicionales desarrollados se decidió

seleccionar el Modelo 2 ya que este logra superar el 80 % de exactitud en validación y lo mantiene aun realizando pruebas externas al proceso de entrenamiento como muestra la tabla 4.4 y la matriz de confusión 4.5.

Cabe destacar que el nivel de exactitud del 95 % en entrenamiento y 81 % en validación, se logró por el tipo de red implementada (Modelo 2) y por el uso de Data Augmentation para llegar a las 10,880 imágenes en el entrenamiento, lo que proporciona una gran variedad de ejemplos de cada tipo de vehículo, dicha base de datos se creó realizando un compendio de distintas bases de datos disponibles en el internet.

La red de clasificación creada no solo logró obtener buenos resultados de clasificación, sino que también se implementó con una transmisión de video y obtuvo en promedio de 20 cuadros por segundo, utilizando una interfaz gráfica creada en LabVIEW. Para lograr esta implementación de Python-LabVIEW se empleó una nueva función de LabVIEW la cual permite correr funciones de archivos de Python, lo que permitió crear una función de Python donde se cargará el modelo de CNN desarrollado para realizar las predicciones de las imágenes entrantes de la transmisión de video, la captura de video y los resultados de la predicción se muestran en la interfaz gráfica en LabVIEW como se puede ver en la figura 3.12. Esta implementación se debió gracias al tamaño de la red desarrollada ya que en comparación con redes ya establecidas en el área de la IA como VGG16 o Alexnet, nuestra red es más esbelta que las mencionadas.

El algoritmo desarrollado demostró tener una buena aplicación en transmisión de video llegando aproximadamente a los 20 cuadros por segundo, todo el procesamiento se llevó a cabo en el CPU de una computadora portátil, y debido a este resultado se implementó en un sistema embebido (Jetson Nano), el cual nos demostró que trabajando igualmente con el CPU únicamente logró tener tiempos aproximados de 0.79 segundos por cuadro, sin embargo, aún cabe la posibilidad de mejora implementando la red en el GPU del Jetson Nano y mejorar estos tiempos de ejecución.

La red de detección desarrollada obtuvo un 87.7 % de exactitud en entrenamiento y un 88 % de exactitud en validación, además se logró implementar en tiempo real con un promedio de 20 cuadros por segundo. También se logró implementar en un sistema embebido con una tasa de procesamiento alrededor de 3.7 cuadros por

segundo trabajando con el CPU únicamente. Para lograr lo mencionado anteriormente se necesitó realizar varios programas de apoyo y un análisis a la metodología de YOLO.

Se fabricó una base de datos específica para el entrenamiento de la red de detección, esto porque se es necesario contar con la información de la ubicación de los vehículos en la imagen. Para fabricar dicha base de datos se utilizaron cuatro programas distintos elaborados en LabVIEW, con los cuales se fabrican las imágenes con su respectivo archivo de texto, el cual contiene las ubicaciones de los vehículos para la detección, esta base de datos cuanta con 16 mil imagenes.

Ambas redes (clasificación y detección) fueron implementadas para funcionar en el CPU de una computadora y en ambos casos se obtienen buenos tiempos de procesamiento, pero en el caso de la red de detección cuando esta es implementada en el sistema embebido se obtienen tiempos de análisis por imagen de 3.7 cuadros por segundo, el cual aún es posible mejorar su rendimiento haciendo uso del GPU con el que el sistema embebido cuenta, sin embargo esto se establece como trabajo futuro en una siguiente investigación.

En conclusión, para lograr la implementación de algoritmos de inteligencia artificial en plataformas embebidas en tiempo real, se identificó que se debe de manejar una arquitectura esbelta, es decir, se debe buscar una red que mantenga un nivel adecuado de exactitud para dicha aplicación sin que se convierta en una red profunda. La recomendación para desarrollo de una CNN esbelta por parte de esta investigación es iniciar con pocas capaz internas e ir modificando sus parámetros (profundidad) con respecto a los límites de exactitud, profundidad y consumo de memoria que se hayan establecido. En base a esta idea de desarrollo se encontró que para las CNN al tener un número menor a 7 capaz internas permite el funcionamiento de las redes desarrolladas a 20 FPS, además de cumplir con los niveles de exactitud establecidos. En cuanto a su implementación en el sistema embebido se identificó que si es posible llegar a implementarlo con tiempos de procesados adecuados si se aplican las redes en el GPU.

Apéndice A

Anexo I: Código para CNN de Clasificación

```
import os
import tensorflow as tf
import json
import numpy as np
from tensorflow import keras #Utilizamos Keras porque facilita trabajar con redes neuronales
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator

class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if (logs.get('acc') > 0.95):
            print("\nSe alcanzo 95% de exactitud por lo que cancelamos el entrenamiento!")
            self.model.stop_training = True



# Directorio con las imagenes de entrenamiento de Autos
train_Autos_dir = os.path.join('training\Autos')

# Directorio con las imagenes de entrenamiento de camioneta
train_Camionetas_dir = os.path.join('training\Camionetas')

# Directorio con las imagenes de validacion de autos
validation_Autos_dir = os.path.join('validation\Autos')

# Directorio con las imagenes de validacion de camioneta
validation_Camionetas_dir = os.path.join('validation\Camionetas')

# Directorio con las imagenes de entrenamiento de pickup
train_Pickups_dir = os.path.join('training\Pickups')

# Directorio con las imagenes de validacion de pickup
validation_Pickups_dir = os.path.join('validation\Pickups')
```

```
# Directorio con las imagenes de entrenamiento de motocicletas
train_Motocicletas_dir = os.path.join('training\Motocicletas')

# Directorio con las imagenes de validacion de motocicletas
validation_Motocicletas_dir = os.path.join('validation\Motocicletas')

# Directorio con las imagenes de entrenamiento de camiones
train_Camion_dir = os.path.join('training\Camion')

# Directorio con las imagenes de validacion de camiones
validation_Camion_dir = os.path.join('validation\Camion')

# Callback
callbacks = myCallback()

import tensorflow as tf

model = tf.keras.models.Sequential([
    # primera convolucion
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(300, 300, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Segunda Convolucion
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Tercera Convolucion
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Cuarta Convolucion
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Quinta convolucion
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # aplicamos flat para alimentar la CNN
    tf.keras.layers.Flatten(),
    # tenemos 512 neuronas en la capa oculta
    tf.keras.layers.Dense(49, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])



model.summary()



#from tensorflow.keras.optimizers import Adam

model.compile(loss='mean_squared_error',
              optimizer='Adam',
              metrics=['acc'])


#from keras.preprocessing.image import ImageDataGenerator
```

```
# todas las imagnes se reescalan a 1/255
train_datagen = ImageDataGenerator(rescale=1 / 255,
                                   rotation_range=20,
                                   zoom_range=0.15,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.15,
                                   horizontal_flip=True,
                                   fill_mode="nearest")

validation_datagen = ImageDataGenerator(rescale=1 / 255,
                                        rotation_range=20,
                                        zoom_range=0.15,
                                        width_shift_range=0.2,
                                        height_shift_range=0.2,
                                        shear_range=0.15,
                                        horizontal_flip=True,
                                        fill_mode="nearest")

# Utilizando Flow se obtienen las imagenes de entrenamiento en lotes de 128 utilizando el
# generados train_datagen
train_generator = train_datagen.flow_from_directory(
    'training',  # Este es el directorio fuente de las imagenes para entrenamiento
    target_size=(300, 300),  # todas las imagenes se reescalan a 300x300
    batch_size=128,
    # como en el 'loss' se utiliza binary_crossentropy creamos etiquetas binarias
    class_mode='categorical')

# Utilizando Flow se obtienen las imagenes de validacion en lotes de 32 utilizando el
# generados validation_datagen
validation_generator = validation_datagen.flow_from_directory(
    'validation',  # Este es el directorio fuente de las imagenes de validacion
    target_size=(300, 300),  # todas las imagenes se reescalan a 300x300
    batch_size=32,
    # como en el 'loss' se utiliza binary_crossentropy creamos etiquetas binarias
    class_mode='categorical')

history = model.fit_generator(
    train_generator,
    steps_per_epoch=85,
    epochs=200,
    verbose=1,
    validation_data=validation_generator,
    validation_steps=102,
    callbacks=[callbacks])

# save to json:
history_json= history.history
with open("history11320.json", mode='w') as json_file:
    json.dump(str(history_json), json_file)

model_json = model.to_json()
with open("modelprueba11320.json", "w") as json_file:
    #modifical el nombre cada vez que se vaya a correr el codigo
```

```
    json_file.write(model_json)
# serializar los pesos a HDF5
model.save_weights("modelprueba11320.h5")
print("Modelo Guardado!")
```

Apéndice B

Anexo II: Código para CNN de Detección

```
import os
from pathlib import Path
import json
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from tensorflow import keras  # Utilizamos Keras porque facilita trabajar con redes neuronales
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras.utils import Sequence
from tensorflow.python.keras.preprocessing import image
from tensorflow.python.keras import activations
from tensorflow.python.keras import layers

c = 5
grid_x = 6
grid_y = 3
N_box = 1
batch_s = 133


class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if (logs.get('acc') > 0.96):
            print("\nSe alcanzo 95% de exactitud por lo que cancelamos el entrenamiento!")
            self.model.stop_training = True


class DataGenerator(Sequence):

    def __init__(self, list_IDs, labels, image_path, label_path,
                 to_fit=True, batch_size=133, dim=(250, 500),
```

```
                n_channels=3, n_classes=10, shuffle=False):

        self.list_IDs = list_IDs
        self.labels = labels
        self.image_path = image_path
        self.label_path = label_path
        self.to_fit = to_fit
        self.batch_size = batch_size
        self.dim = dim
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):

        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):

        # Generate indexes of the batch
        indexes = self.indexes[index * self.batch_size:(index + 1) * self.batch_size]

        # Find list of IDs
        list_IDs_temp = [self.list_IDs[k] for k in indexes]

        # Generate data
        X, y = self._generate_(list_IDs_temp)
        # print(y)
        return X, y

    def on_epoch_end(self):

        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)

    def _generate_(self, list_IDs_temp):

        # Initialization
        X = np.empty((self.batch_size, *self.dim, self.n_channels))
        y = np.zeros((self.batch_size, 3, 6, 1))
        # Generate data
        # print(len(list_IDs_temp))
        for i, ID in enumerate(list_IDs_temp):
            # Store sample
            # print(self.image_path)
            # print(list_IDs_temp[i])
            file = os.path.join(self.image_path, str(list_IDs_temp[i]))
            file2 = os.path.splitext(file)[0]
            file3 = file2 + '.jpg'
            X[i,] = self._load_image(file3)
            # print(list_IDs_temp[i])
            # Store labels
            control = 0
```

```
            file4 = os.path.join(self.label_path, str(list_IDs_temp[i]))
            file5 = os.path.splitext(file4)[0]
            file6 = file5 + "g." + "txt"
            y[i,] = self.label_file(file6, control)

            # print(list_IDs_temp[i])
            # print(y[i, :, :, :])
        return X, y

    def _load_image(self, image_path):
        img = mpimg.imread(image_path)
        img = img / 255
        return img

    def label_file(self, label_path, cont):
        control = cont
        t = np.loadtxt(label_path, dtype='float', delimiter='\t', ndmin=2)
        # print(t.shape)
        [r, c] = t.shape
        x_res = t[0, 9]
        y_res = t[0, 10]
        cell_y = x_res / grid_x
        cell_x = y_res / grid_y
        y = np.zeros((grid_y, grid_x, 1))
        # print(x_res)
        for dim in range(r):
            pc = 0
            temp = t[dim, :]
            B_box = temp[0:4]
            clss = temp[4:9]
            # print(B_box, clss)
            for row in range(grid_y):
                for col in range(grid_x):
                    lim_yI = (col * cell_x) / x_res
                    lim_xI = (row * cell_y) / y_res
                    lim_yS = ((col + 1) * cell_x) / x_res
                    lim_xS = ((row + 1) * cell_y) / y_res
                    # print(lim_xI, lim_yI, lim_xS, lim_yS)
                    ctr_x = B_box[0]
                    ctr_y = B_box[1]
                    # print(ctr_x, ctr_y)
                    if (lim_yI <= ctr_x <= lim_yS) and (lim_xI <= ctr_y <= lim_xS):
                        cell_p = np.array([row, col])
                        pc = np.array([1])
                        # print(cell_p)
            B = B_box.reshape(1, 4)
            C = clss.reshape(1, 5)
            PC = pc
            y_t = np.concatenate((PC), axis=None)
            # print(np.shape(pc), np.shape(B), np.shape(clss))
            y[cell_p[0], cell_p[1]] = y_t

        if control == 0:
            Ytrue = y
            control = 1
```

```
        else:
            Ytrue = np.vstack((Ytrue, y))
        return Ytrue


import numpy
import tensorflow as tf


# Funcion de Loss especifica de yolo
def my_loss_function(y_true, y_pred):
    # y_true_clss = y_true[..., 5:10]
    # y_pred_clss = y_pred[..., 5:10]
    # y_true_xy = y_true[..., 1:3]
    # y_pred_xy = y_pred[..., 1:3]
    # y_true_hw = y_true[..., 3:5]
    # y_pred_hw = y_pred[..., 1:3]
    y_true_conf = y_true[..., 0]
    y_pred_conf = y_pred[..., 0]
    indices_0 = y_true_conf == 0
    # print(indices_0)
    indices_1 = y_true_conf == 1
    # print("indice 1:", indices_1)
    obj_true = y_true_conf[..., indices_1]
    obj_pred = y_pred_conf[..., indices_1]
    noobj_true = y_true_conf[..., indices_0]
    noobj_pred = y_pred_conf[..., indices_0]
    # obj_hw_true = y_true_hw[..., indices_1]
    # obj_hw_pred = y_pred_hw[..., indices_1]
    # hw = keras.backend.sum(keras.backend.square(keras.backend.sqrt(obj_hw_true)-
    # keras.backend.sqrt(obj_hw_pred)))
    # print(obj_true, obj_pred)
    # print(noobj_true, noobj_pred)
    # clss_loss =(keras.backend.sum(keras.backend.square(y_true_clss - y_pred_clss),
    # axis=-1)
    # * y_true_conf)
    # print("clases:", clss_loss)
    # c = (keras.backend.sum(keras.backend.square(y_true_xy - y_pred_xy), axis=-1)
    # * y_true_conf)
    # print("xy:", c)
    # c_1 = c[..., 0]
    # c_2 = c[..., 1]
    # print(np.reshape(c_1, (3, 3, 1)))
    # print(c_2*y_true_conf)
    # xy_loss = np.concatenate((np.reshape(c_1 * y_true_conf, (7, 7, 1)), np.reshape(c_2 *
    # y_true_conf, (7, 7, 1))),
    # axis=2)
    # print(xy_loss)
    # y_true_conf[1, 1] = 0
    # xy_loss = np.concatenate((np.reshape(c_1*y_true_conf, (3, 3, 1)),
    # np.reshape(c_2*y_true_conf, (3, 3, 1))), axis=2)
    # print((xy_loss))
    # c_wh = (keras.backend.sum(keras.backend.square(keras.backend.sqrt(y_true_hw) -
    # keras.backend.sqrt(y_pred_hw)),
    # axis=-1) * y_true_conf)
```

```
# print("wh:", c_wh)
# c_wh1 = c_wh[..., 0]
# c_wh2 = c_wh[..., 1]
# wh_loss = np.concatenate((np.reshape(c_wh1 * y_true_conf, (7, 7, 1)),
# np.reshape(c_wh2 * y_true_conf, (7, 7, 1))),
#                          axis=2)
# print(wh_loss)
# conf_loss = np.reshape(np.square(y_true_conf - y_pred_conf), (7, 7, 1))
# print(conf_loss)
# loss = np.concatenate((conf_loss, xy_loss, wh_loss, clss_loss), axis=2)
# x, y, z = Ytrue.shape
# print(loss)
# y = np.zeros((3, 3, 3, 10))
# print(y)
# for f in range(2):
#    y[f, ] = np.ones((1, 10))
# print(y)
conf_loss = keras.backend.reshape(keras.backend.square(y_true_conf - y_pred_conf),
                                  (batch_s, 6, 3, 1))
# hw_true = y_true_hw
# hw_pred = y_pred_hw
# xy_true = y_true_xy
# xy_pred = y_pred_xy
# boxA = [xy_true[..., 0:1] - hw_true[..., 1:2] / 2, xy_true[..., 1:2] -
#         hw_true[..., 0:1] / 2,
#         hw_true[..., 1:2] / 2 + xy_true[..., 0:1], hw_true[..., 0:1] /
#         2 + xy_true[..., 1:2]]
# boxB = [xy_pred[..., 0:1] - hw_pred[..., 1:2] / 2, xy_pred[..., 1:2] -
#         hw_pred[..., 0:1] / 2,
#         hw_pred[..., 1:2] / 2 + xy_pred[..., 0:1], hw_pred[..., 0:1] / 2 +
#         xy_pred[..., 1:2]]

# xA = keras.backend.maximum(boxA[0], boxB[0])
# yA = keras.backend.maximum(boxA[1], boxB[1])
# xB = keras.backend.minimum(boxA[2], boxB[2])
# yB = keras.backend.minimum(boxA[3], boxB[3])

# compute the area of intersection rectangle
# interArea = keras.backend.maximum(keras.backend.zeros_like(hw_true[..., :1]),
# xB - xA + keras.backend.ones_
# like(hw_true[..., :1])) * keras.backend.maximum(
#     keras.backend.zeros_like(hw_true[..., :1]), yB - yA +
#     keras.backend.ones_like(hw_true[..., :1]))
# print(interArea)
# pre_union
# boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
# boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
# iou
# iou = interArea / (boxAArea + boxBArea - interArea)
# print("iou", iou)
# print("tconf", np.reshape(iou, (1, 7, 7)) * y_true_conf)
# print("pconf", y_pred_conf)
# conf = keras.backend.sum(keras.backend.square((keras.backend.reshape(iou,
# (batch_s, 1, 3, 3))
# * y_true_conf) - (y_pred_conf)), axis=-1)
```

```
    # print("conf", conf)
    conf2 = (1.6 * keras.backend.sum(keras.backend.square(obj_true - obj_pred))
             + 0.65 * keras.backend.sum(keras.backend.square(noobj_true - noobj_pred)))
    loss = conf2 + 0.05 * conf_loss
    return loss


# Callback
callbacks = myCallback()

model = tf.keras.models.Sequential([
    # primera convolucion
    tf.keras.layers.Conv2D(64, (3, 6), input_shape=(250, 500, 3), strides=(2, 2),
                           activation='relu', name='conv1'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Segunda Convolucion
    tf.keras.layers.Conv2D(128, (3, 6), strides=(2, 2), activation='relu', name='conv2'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Tercera Convolucion
    tf.keras.layers.Conv2D(256, (3, 6), strides=(2, 2), activation='relu', name='conv3'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # aplicamos flat para alimentar la CNN
    tf.keras.layers.Flatten(),
    # tenemos 512 neuronas en la capa oculta0
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(grid_x * grid_y * (N_box), activation='linear'),
    tf.keras.layers.Reshape([grid_y, grid_x, (N_box)], name='last')
])  # Se puede utilizar la misma funcion de softmax con dos neuronas si lo desearan

model.summary()

# from tensorflow.keras.optimizers import Adam
opt = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(loss=my_loss_function,
              optimizer=opt,
              metrics=['acc'])

# from keras.preprocessing.image import ImageDataGenerator
# todas las imagnes se reescalan a 1/255
train_labels_dir = os.path.join('Test8')
train_labels_names = sorted(os.listdir(train_labels_dir))
Nt = len(os.listdir(train_labels_dir))
train_images_dir = os.path.join('Imag8')
# print(train_images_dir)
train_images_names = sorted(os.listdir(train_images_dir))
Nt = len(os.listdir(train_images_dir))

train_labels_dirv = os.path.join('valtest')
train_labels_namesv = sorted(os.listdir(train_labels_dirv))
# Nt = len(os.listdir(train_labels_dirv))
train_images_dirv = os.path.join('valIMG')
# print(train_images_dir)
train_images_namesv = sorted(os.listdir(train_images_dirv))
```

```
# Nt = len(os.listdir(train_images_dir))
training_generator = DataGenerator(list_IDs=train_images_names,
                                   labels=train_labels_names,
                                   image_path=train_images_dir,
                                   label_path=train_labels_dir,
                                   batch_size=133)
validation_generator = DataGenerator(list_IDs=train_images_namesv,
                                     labels=train_labels_namesv,
                                     image_path=train_images_dirv,
                                     label_path=train_labels_dirv,
                                     batch_size=133)
# agreagar los datos de os archivos txt
history = model.fit(
    training_generator,
    steps_per_epoch=117,
    epochs=40,
    verbose=1,
    validation_data=validation_generator,
    validation_steps=5,
    callbacks=[callbacks]
)

# save to json:
history_json = history.history
with open("history30521_strides3N15500(loss1+0.05loss2)-noobj0.65l.json",
          mode='w') as json_file:
    json.dump(str(history_json), json_file)

"""La sigueinte porsion del codigo es para guardar el modelo entrenado"""
model_json = model.to_json()
with open("modelprueba30521_strides3N15500(loss1+0.05loss2)-noobj0.65l.json",
          "w") as json_file:  # modifical el nombre cada vez que se vaya a correr el codigo
    json_file.write(model_json)
# serializar los pesos a HDF5
model.save_weights("modelprueba30521_strides3N15500(loss1+0.05loss2)-noobj0.65l.h5")
print("Modelo Guardado: modelprueba30521_strides3N15500(loss1+0.05)loss2)-noobj0.65l")
```

Apéndice C

Anexo III: Artículo de Congreso

A Lean Convolutional Neural Network for Vehicle Classification

Jonathan J. Sanchez-Castro,
Julio C. Rodríguez-Quiñonez*,
Luis R. Ramírez-Hernández,
Facultad de Ingeniería
Universidad Autónoma de Baja California
Mexicali, México
julio.rodriguez81@uabc.edu.mx

Guillermo Galaviz,
Daniel Hernández-Balbuena,
Gabriel Trujillo-Hernández,
Wendy Flores-Fuentes,
Facultad de Ingeniería
Universidad Autónoma de Baja California
Mexicali, México

Paolo Mercorelli
Control and Drive Systems Lab
Leuphana University of Leuneburg
Leuneburg, Germany
marcorelli@uni.leuphana.de

Wilmar Hernández-Perdomo
Ingeniería Electrónica y Redes de Información
Universidad de Las Américas
Quito, Ecuador
wilmar.hernandez@udla.edu.ec

Oleg Sergiyenko,
Félix Fernando González-Navarro,
Instituto de Ingeniería
Universidad Autónoma de Baja California
Mexicali, México
srgnk@uabc.edu.mx

***Abstract*—Image classification is an important task in machine vision, in which vehicle classification is used for different applications like traffic analysis, autonomous driving, security, among others. Recent studies made with Convolutional Neural Networks (CNN) have shown that these networks have surpassed older algorithms like Support Vector Machine (SVM) and K-Nearest Neighbor (KNN) in terms of accuracy, speed, and resources management. Even though that CNN have better accuracy and speed they still are heavy in resource consumption on computers which makes them not suitable to deploy on an embedded platform. This paper proposes a lean CNN that has a smaller number of parameters and still maintaining the best accuracy possible on vehicle classification.**

***Index Terms*—CNN, Artificial Intelligence, Vehicle Classification,**

I. Introduction

Former Artificial Neural Networks (ANN) are capable of grouping, pattern recognition, prediction, approximation, among others applications [1] [2], but the arrival of Deep Neural Networks has opened a vast area of applications for artificial intelligence like image classification which can be applied on medical, security, automobile, commerce and other sectors around the world [3] [4] [5].

In recent years the increasing volume of vehicles on city roads has leaded to the creation of different kind of machine vision applications like security, surveillance, traffic analysis on roads, parking slots, among others [6], where the classification of vehicles is an important task and also is a difficult challenge for image classification algorithms. Vehicle classification has been done with algorithms like K-Nearest Neighbor and Support Vector Machine, the downside of this algorithms is that they are not suitable for remote applications using hardware with limited resources like an embedded system, due to the characteristics of this methods which is the process of extracting the features in background subtraction of images and also the operations done by the classifier [6], [7], [8], however the arrival of Convolutional Neural Networks which have become a relevant study topic due to their achievements in machine vision applications have turn into the go to algorithms for object classification and even object detection with Regional Convolutional Neural Networks (RCNN). CNN are one of the most used algorithms on image classification and have a wide area of application such as security, health care, traffic monitoring, autonomous driving, among others [9], [10], [11].

Convolutional Neural Networks have obtained great results in object classification which have produced different kinds of distinguished CNN structures like Alexnet, VGG16, GOOGLE-LeNet, which have demonstrated outstanding capabilities in the classification of different kinds of objects [9], [11], [12]. Although these CNN structures have their advantages, they also use a considerable amount of computers resources and are not suitable for embedded systems. Our approach to this kind of complication is to use a lean network architecture proposed for this paper, a CNN that has the minimum number of layers but also has high accuracy, compared with network architecture that requires a significant amount resources for their training.

CNN are comprised by multiple convolution layers, followed by pooling layers and finally a fully connected Artificial Neural Network (ANN) as seen in Fig. I. The function of convolutional and pooling layers is to extract the characteristics from the input images to have the best classification possible using the ANN.

In this work we perform a research with CNN to find the

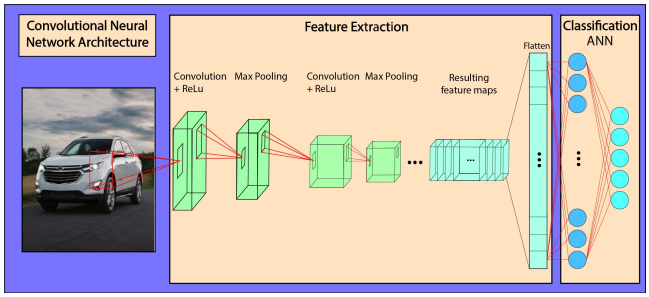


Fig. 1. Convolutional Neural Networks Architecture

structure with the smaller number of possible layers, applied to vehicle classification. Also, this research has another objective which is to compile information about how CNN behave by modifying their parameters and layers.

II. Related Works

Vehicle classification is an ongoing study, where different type of algorithms are proposed to solve this specific task in which a great number use background subtraction with other characteristics of a vehicle image for the classification. There are implementations like the use of SVM with Histogram of Oriented Gradients (HOG) descriptors, SVM with SIFT features, KNN and virtual zones for vehicle counting, detection, among others [13], [14], [15]. Most of these implementations requires several algorithms to function like the feature extraction methods, the data conditioning and finally the classifier implemented, all of these algorithms consume considerable computer resources which can be seen as time consuming and this would not make them suitable for embedded applications.

The high accuracy in classification task and implementation of CNN have made them one of the most used algorithms in machine vision applications like vehicle classification. Well known CNN structures are evidence of this, like Alexnet, VGG16 and VGG19 which has been used to classify numerous objects. The way these structures are implemented in recent researches, is by using a pre-train model and then fine tune the network for better classification with vehicles, the disadvantage of this is that it's not possible to introduce more classes to the network [16], [17], [18].

III. Methodology

This paper presents a Convolutional Neural Network that is comprised by feature extraction layers (Convolution and Max pooling) and finally a fully connected layer (ANN), which can be seen in Fig. 1.

Figure 2 presents the architecture of our CNN, the number of layers was decided by a previous research where it was found that with small structure, CNN still have a good performance [19], [20]. Our structure as seen in Fig. 2 is comprised buy five layers of convolution and five layers of Max Pooling, in the classification section we have an ANN of two layers which classifies 5 types of vehicles: Sedan, Buses, SUV, Pickup trucks and Motorcycles.

The initial image resolution is 300x300 pixels which is fed to the first convolutional layer with sixteen filters of 3x3 and then passes through the Max Pooling Layer which reduces the size of the generated feature maps by using a 2x2 array, this process is then repeated four times with 32, 64, 64 and 64 filters per convolution layer, the Max Pooling layers stay with the same array. Finally, the resulting feature maps of the last Pooling layer are flattened and fed into the ANN for classification, the complete network architecture can be seen in Fig. 2.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 298, 298, 16)	448
max_pooling2d (MaxPooling2D)	(None, 149, 149, 16)	0
conv2d_1 (Conv2D)	(None, 147, 147, 32)	4640
max_pooling2d_1 (MaxPooling2	(None, 73, 73, 32)	0
conv2d_2 (Conv2D)	(None, 71, 71, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 35, 35, 64)	0
conv2d_3 (Conv2D)	(None, 33, 33, 64)	36928
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 64)	0
conv2d_4 (Conv2D)	(None, 14, 14, 64)	36928
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 512)	1606144
dense_1 (Dense)	(None, 5)	2565

Total params: 1,706,149
Trainable params: 1,706,149
Non-trainable params: 0

Fig. 2. Proposed Model Architecture and parameters

A high resolution image brings more fine details for the CNN to extract and use for the classification, using high resolutions also brings some drawbacks like a bigger demand of memory usage and more computational operations being done to the image, so with a comparison between different researches with architectures like VGG16, VGG19, Alexnet and ResNet50 where there would be input resolutions of 224x224 pixels, but also there are cases were the datasets use images below 100x100 pixels, so it was decided on a resolution of 300x300 aiming to have more details on our images. The number of filters and their sizes was also selected by the comparison of different architectures like VGG16, Alexnet, ResNet50, where the most common use filter size would be 3x3, but the quantity would vary from 64 to 512 filters per convolution, and since we are looking for a lean network this architecture started with a small number of filters but they would increase over each convolutional layer as mentioned before (16, 32, 64, 128).

A. Convolution Layers

The convolution layers are the core layers of this algorithm, because these are responsible for the feature extraction of the input image, each convolution layer has their own kernel and parameters. The eq.1 shows the mathematical operation for this procedure,

$$S = max(0, X * K) \tag{1}$$

Where S is the resulting feature maps, K is the filter bank used as kernel, X is the pixels of the entered image. The use of ReLu activation (max(,)) function is to gain non-linearity on the convolution layers.

B. Activation Function

The activation function indicates when the neuron will activate, there are different activation functions like Sigmoid, SoftMax, Tanh, Leaky ReLu, Maxout, among others with different characteristics, in this model we use Relu for the first layer and for the last layer we use SoftMax for the ANN.

C. Max Pooling Layer

The Max Pooling layers serve as a robustness enhancer of the CNN to handle translations, usually the pooling layer is after the convolution layer. The pooling layer can be Max or Average Pooling, in our architecture we use Max Pooling to handle any existing translations.

D. Fully Connected Layer

The Fully Connected Layer is the neural network responsible for the learning of the different objects of interest, in our network we have a two layer Neural Network where the first layer has 512 neurons and the second layer has five neurons, the use of five neurons in the last layer is for the five objects to be classified.

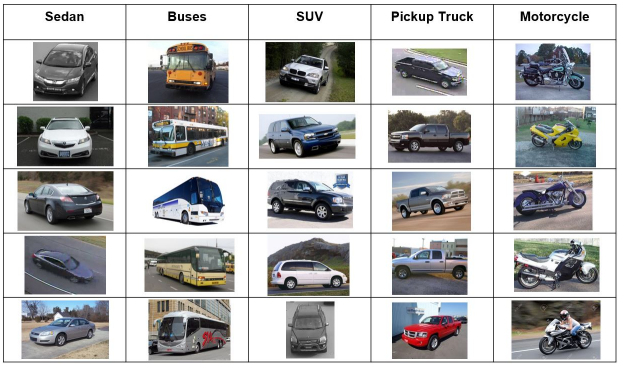


Fig. 3. Samples of Image Dataset

IV. Experiments and Results

All of our experiments were done with Python using libraries like Keras, TensorFlow on a server with 12 physical cores and 100 GB of RAM, also our test have a callback to stop the training process if we have reached an accuracy equal or above of 95% in training, this to prevent the over fitting of our network. The proposed CNN is trained with an image dataset that was created, with the use of existing free image sets on the internet, the images were selected manually to create the training set and validation set, used to feed the proposed algorithm. This dataset has 14144 in total images, 10880 for training and 3264 for validation, in which they are divided in 5 different classes Sedan, Buses, SUV, Pickup trucks and Motorcycles, which can be seen in Fig. 3. The images in this dataset have various resolutions, also Data Augmentation was implemented for robustness and to create more training and validation samples without the necessity to find new images for the image dataset. In our tests the networks were not allowed to surpass the 95% limit in training accuracy due the problem known as over fitting, this happens when a network learns to much information from the training images that it will affect its performance when classifying new data.

TABLE I
Table of Model Results

	% Ac. Training	% Ac. Validation	D. A.	Epochs	T. time
Model1	0.9662	0.7647	no	10	<3hrs
Model2	0.9339	0.8199	yes	200	40hrs
Model3	0.9507	0.8137	yes	50	<4hrs
Model4	0.9524	0.8327	yes	172	33hrs
Model5	0.9511	0.8392	yes	187	62hrs
Model6	0.9512	0.8382	yes	172	58hrs

The first trained Model which is proposed in the methodology section, achieves a 96% in training accuracy and 76% in validation accuracy in ten epochs, and complete the training process in less than three hours and without Data Augmentation. This network is taken as reference to perform modifications on the proposed structure to analyze the difference in time of training, implications on having more neurons in the first layer of the ANN and also the impact of having more training parameters with implementing a second convolution layer. The results of the models trained are shown in Table I, and the different model architectures are shown in Table II.

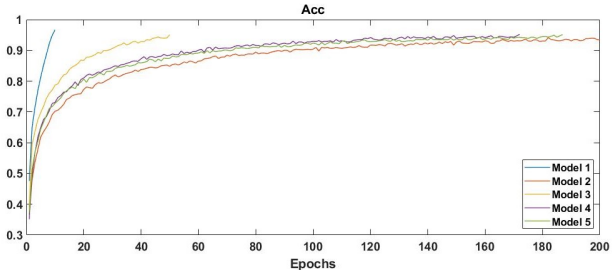


Fig. 4. Models Training Accuracy

Model 1 has shown that with 10 epochs of training its sufficient to reach a 95% in training and 76% on validation of the CNN and this without Data Augmentation. The comparison with the other models has shown that Model 4, 5 and 6 have better results in validation accuracy almost 7% approximately,

TABLE II
TABLE OF MODEL ARCHITECTURES

	Convolution	Max-Pooling	Convolution	Max-Pooling	Convolution	Max-Pooling	Convolution	Max-Pooling	Convolution	Max-Pooling	Dense	Dense
Model1	16,(300x300)	2,2	32	2,2	64	2,2	64	2,2	64	2,2	512	5
Model2	16,(300x300)	2,2	32	2,2	64	2,2	64	2,2	64	2,2	49	5
Model3	16,(300x300)	2,2	32	2,2	64	2,2	64	2,2	64	2,2	512	5
Model4	16,(300x300)	2,2	32	2,2	64	2,2	64	2,4	128	2,2	512	5
Model5	2x(16,300x300)	2,2	32	2,2	64	2,2	64	2,2	64	2,2	512	5
Model6	2x(16,300x300)	2,2	32	2,2	64	2,2	64	2,2	128	2,2	1024	5

with the downside that this models have longer training times due to Data Augmentation being used and that they have more training parameters by the extra convolution in the first layer for both models, and model 6 also has the double number of filters in the last convolutional layer and the double number of neurons in the first input layer. Model 4 performs nearly as equal to model 5 and 6, with half of the training time which can be due to the smaller number of parameters by not having a second convolution layer and double the number of neurons in the initial layer on the ANN. Model 2 has shown that with a smaller number of neurons on the input of the neural network takes a longer time to reach 95% in training accuracy, even with our test having a limit of 200 epochs in training process Model 2 still did not reach the desired level. Model 3 has shown nearly as good result as Model 4, 5 and 6 but in less training time.

Figure 4 shows the values of accuracy over 200 epochs of training where we can see that Model 1 is the first one to pass above the 95% accuracy mark in less than ten epochs, this result is related to the training time with table I, and also considering that this model does not have Data Augmentation. Model 3 has the same architecture as Model 1 but with Data Augmentation being used, which takes five times more epochs to pass 95% in training accuracy. Viewing this result from a time perspective we can see that it does not take five times more than Model 1, but in direct comparison with the rest models (Model 2, 4, 5 and 6) it takes nearly as ten times less to reach the stop mark.

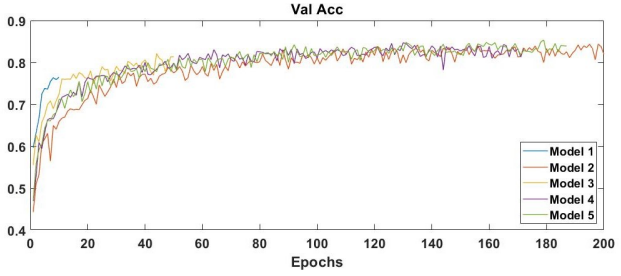


Fig. 5. Models Validation Accuracy

Validation accuracy of the models in Fig. 5 shows a similar behavior of the results during the 200 epochs in training process, but there is a difference in the results, the models do not surpass an 84% of accuracy on the validation set, which indicate that our network still needs improvement. From Fig. 5 there is a notable stagnation of the validation results which is

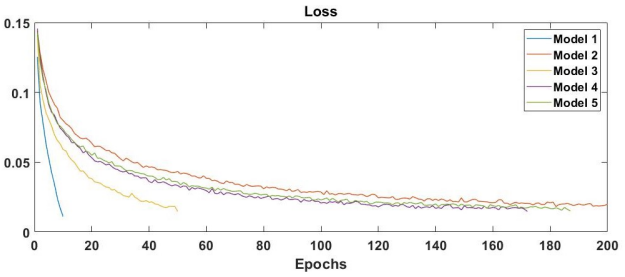


Fig. 6. Models Training Loss

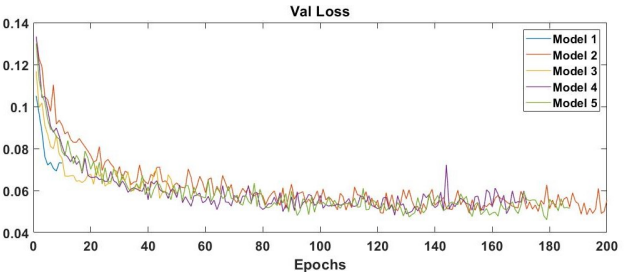


Fig. 7. Models Validation Loss

below 84%, there is also something notable in the trace of the result which is that the values obtained after each epoch varies more than those in Fig. 4, also there seems that almost all the models perform similar when surpassing 80% in validation accuracy.

The graphs from Fig. 6 are the results of the training loss function, in which we observe that the behavior of the different models results is maintained, as we see in Fig. 4 Model 1 is the first to achieve the specified limit of training accuracy in the less amount of time and this translates also to having the value of the loss function to decrease rapidly in accord to the accuracy obtained. In Fig. 7 we have the validation loss function values, which fallows the same behavior found with Fig. 4 and Fig. 6, while the accuracy rises the loss function decreases and also in Fig. 7 the same behavior of how the values vary more is seen, like with models validation accuracy in Fig. 5.

To prove that the proposed models are close to 80% of accuracy we use Model 5 to analyze 24 new vehicle images to see if the relation of 80% on accuracy is maintained, Fig. 6 shows a set of vehicle images with a color background of green

for the correctly classified, and red for the incorrect vehicles.

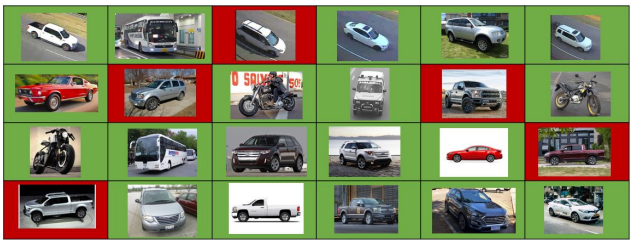


Fig. 8. Vehicle Classification Test

V. Conclusions

In this work a CNN was develop for vehicle classification, and several tests were performed to show the impact of how modifications to the network parameters and structures can have on their performance. For the development of this network we also created our own image dataset of vehicles with different categories: Sedan, SUV, Bus, Pickup Truck and Motorcycle. With the proposed CNN model and the image dataset build the obtained validation accuracy was above 80% using Data Augmentation and a lean network, these results showed that we could obtain a better classification accuracy by improving the image dataset for the network. As mentioned, this work also looks for a lean CNN by comparing different approaches, and given the results of Table I, Model 3 is the best choice with a score on validation accuracy of 81.37% in less than four hours of training time. Model 3 did not obtain the top result in validation accuracy from all the proposed networks but it also wasn't to far from the best ones, which is model 5 with 83.92% in validation accuracy, but Model 5 took approximately 62 hours to complete the training process, making it unsuitable for an embedded system. Future projects of this work would consist on increasing the image dataset and improve the network to increase the validation accuracy above 92%, but still maintaining a lean network for implementations on embedded platforms.

Acknowledgment

This work was supported by the Universidad Autónoma de Baja California, Universidad Autonoma de Sinaloa and CONACyT.

References

[1] J. C. Rodriguez-Quinonez, O. Sergiyenko, F. F. Gonzalez-Navarro, L. Basaca-Preciado, and V. Tyrsa, "Surface recognition improvement in 3d medical laser scanner using levenberg–marquardt method," *Signal Processing*, vol. 93, no. 2, pp. 378–386, 2013.

[2] G. Trujillo-Hernández, J. C. Rodríguez-Quiñonez, L. R. Ramírez-Hernández, M. J. Castro-Toscano, D. Hernández-Balbuena, W. Flores-Fuentes, O. Sergiyenko, L. Lindner, and P. Mercorelli, "Accuracy improvement by artificial neural networks in technical vision system," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, pp. 5572–5577, IEEE, 2019.

[3] K. Makantasis, K. Karantzalos, A. Doulamis, and N. Doulamis, "Deep supervised learning for hyperspectral data classification through convolutional neural networks," in *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 4959–4962, IEEE, 2015.

[4] Y. Seo and K.-s. Shin, "Image classification of fine-grained fashion image based on style using pre-trained convolutional neural network," in *2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA)*, pp. 387–390, IEEE, 2018.

[5] P. Xia, J. Hu, and Y. Peng, "Emg-based estimation of limb movement using deep learning with recurrent convolutional neural networks," *Artificial organs*, vol. 42, no. 5, pp. E67–E77, 2018.

[6] A. Arinaldi, J. A. Pradana, and A. A. Gurusinga, "Detection and classification of vehicles for traffic video analytics," *Procedia computer science*, vol. 144, pp. 259–268, 2018.

[7] N. Seenouvong, U. Watchareeruetai, C. Nuthong, K. Khongsomboon, and N. Ohnishi, "Vehicle detection and classification system based on virtual detection zone," in *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 1–5, July 2016.

[8] M. A. Manzoor and Y. Morgan, "Vehicle make and model classification system using bag of sift features," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–5, IEEE, 2017.

[9] Q. Fan, L. Brown, and J. Smith, "A closer look at faster r-cnn for vehicle detection," in *2016 IEEE intelligent vehicles symposium (IV)*, pp. 124–129, IEEE, 2016.

[10] B. B. Traore, B. Kamsu-Foguem, and F. Tangara, "Deep convolution neural network for image recognition," *Ecological Informatics*, vol. 48, pp. 257–268, 2018.

[11] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li, "Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4224–4231, 2018.

[12] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li, "Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4224–4231, 2018.

[13] S.-H. Lee, M. Bang, K.-H. Jung, and K. Yi, "An efficient selection of hog feature for svm classification of vehicle," in *2015 International Symposium on Consumer Electronics (ISCE)*, pp. 1–2, IEEE, 2015.

[14] M. A. Manzoor and Y. Morgan, "Vehicle make and model classification system using bag of sift features," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–5, IEEE, 2017.

[15] N. Seenouvong, U. Watchareeruetai, C. Nuthong, K. Khongsomboon, and N. Ohnishi, "Vehicle detection and classification system based on virtual detection zone," in *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 1–5, IEEE, 2016.

[16] R. Mehra *et al.*, "Breast cancer histology images classification: Training from scratch or transfer learning?," *ICT Express*, vol. 4, no. 4, pp. 247–254, 2018.

[17] Q. Fan, L. Brown, and J. Smith, "A closer look at faster r-cnn for vehicle detection," in *2016 IEEE intelligent vehicles symposium (IV)*, pp. 124–129, IEEE, 2016.

[18] A. K. Rangarajan, R. Purushothaman, and A. Ramesh, "Tomato crop disease classification using pre-trained deep learning algorithm," *Procedia computer science*, vol. 133, pp. 1040–1047, 2018.

[19] Y. Gao and H. J. Lee, "Vehicle make recognition based on convolutional neural network," in *2015 2nd International Conference on Information Science and Security (ICISS)*, pp. 1–4, IEEE, 2015.

[20] X. Wu, R. He, Z. Sun, and T. Tan, "A light cnn for deep face representation with noisy labels," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2884–2896, 2018.

Referencias

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., Deng, L., Penn, G., and Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545.

Aftab, M., Chen, C., Chau, C.-K., and Rahwan, T. (2017). Automatic hvac control with real-time occupancy recognition and simulation-guided model predictive control in low-cost embedded system. *Energy and Buildings*, 154:141–156.

Alippi, C., Disabato, S., and Roveri, M. (2018). Moving convolutional neural networks to embedded systems: the alexnet and vgg-16 case. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 212–223. IEEE.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., and Farhan, L. (2021). Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74.

Chakma, A., Vizena, B., Cao, T., Lin, J., and Zhang, J. (2017). Image-based air quality analysis using deep convolutional neural network. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3949–3952. IEEE.

Chollet, F. et al. (2018). *Deep learning with Python*, volume 361. Manning New York.

Dundar, A., Jin, J., Martini, B., and Culurciello, E. (2016). Embedded streaming deep neural networks accelerator with applications. *IEEE transactions on neural networks and learning systems*, 28(7):1572–1583.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587.

Gokhale, V., Jin, J., Dundar, A., Martini, B., and Culurciello, E. (2014). A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 682–687.

Gong, T., Fan, T., Guo, J., and Cai, Z. (2017). Gpu-based parallel optimization of immune convolutional neural network and embedded system. *Engineering Applications of Artificial Intelligence*, 62:384–395.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Hsiao, P.-Y., Lin, S.-Y., and Huang, S.-S. (2015). An fpga based human detection system with embedded platform. *Microelectronic Engineering*, 138:42–46.

Hu, H., Shah, S. A. A., Bennamoun, M., and Molton, M. (2017). 2d and 3d face recognition using convolutional neural network. In *TENCON 2017-2017 IEEE Region 10 Conference*, pages 133–132. IEEE.

Jain, D. K., Shamsolmoali, P., and Sehdev, P. (2019). Extended deep neural network for facial emotion recognition. *Pattern Recognition Letters*, 120:69–74.

Jeong, J., Cho, J.-H., and Lee, J.-G. (2021). Filter combination learning for cnn model compression. *ICT Express*, 7(1):5–9.

Kondo, T., Ueno, J., and Takao, S. (2014). Medical image recognition of abdominal multi-organs by hybrid multi-layered gmdh-type neural network using principal component-regression analysis. In *2014 Second International Symposium on Computing and Networking*, pages 157–163. IEEE.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

LeCun, Y. et al. (1989). Generalization and network design strategies. *Connectionism in perspective*, 19:143–155.

LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*, pages 253–256. IEEE.

Lee, S., Son, K., Kim, H., and Park, J. (2017a). Car plate recognition based on cnn using embedded system with gpu. In *2017 10th International Conference on Human System Interactions (HSI)*, pages 239–241. IEEE.

Lee, S., Son, K., Kim, H., and Park, J. (2017b). Car plate recognition based on cnn using embedded system with gpu. In *2017 10th International Conference on Human System Interactions (HSI)*, pages 239–241. IEEE.

Lee, S.-H., Bang, M., Jung, K.-H., and Yi, K. (2015). An efficient selection of hog feature for svm classification of vehicle. In *2015 International Symposium on Consumer Electronics (ISCE)*, pages 1–2. IEEE.

Mao, H., Yao, S., Tang, T., Li, B., Yao, J., and Wang, Y. (2016a). Towards real-time object detection on embedded systems. *IEEE Transactions on Emerging Topics in Computing*, 6(3):417–431.

Mao, H., Yao, S., Tang, T., Li, B., Yao, J., and Wang, Y. (2016b). Towards real-time object detection on embedded systems. *IEEE Transactions on Emerging Topics in Computing*, 6(3):417–431.

Osio, J. R. (2019). *Procesamiento digital de imágenes médicas sobre plataformas FPGAs*. PhD thesis, Universidad Nacional de La Plata.

Park, C., Jang, J., Zhang, L., and Jung, J.-I. (2018). Light-weight visual place recognition using convolutional neural network for mobile robots. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–4. IEEE.

Pereira, F. C. and Pereira, C. E. (2015). Embedded image processing systems for automatic recognition of cracks using uavs. *Ifac-PapersOnline*, 48(10):16–21.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Redmon, J. and Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*.

Russell, S. J. and Norvig, P. (2004). *Inteligencia Artificial: un enfoque moderno*. Number 04; Q335, R8y 2004.

Sardogan, M., Tuncer, A., and Ozen, Y. (2018). Plant leaf disease detection and classification based on cnn with lvq algorithm. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pages 382–385. IEEE.

SCT (2016). Manual para obtener los volúmenes de tránsito en carreteras. *Secretaria de Comunicaciones y Transporte de México*.

Seo, Y. and Shin, K.-s. (2019). Hierarchical convolutional neural networks for fashion image classification. *Expert systems with applications*, 116:328–339.

Shi, B., Bai, X., and Yao, C. (2016). An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304.

Traore, B. B., Kamsu-Foguem, B., and Tangara, F. (2018). Deep convolution neural network for image recognition. *Ecological Informatics*, 48:257–268.

Winston, P. H. (1992). *Artificial Intelligence*. Addison-Wesley.

Yang, W., Wang, W., Gao, Y., and Jin, Z. (2018). An embedded tracking system with neural network accelerator. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE.

Yao, G., Lei, T., and Zhong, J. (2019). A review of convolutional-neural-network-based action recognition. *Pattern Recognition Letters*, 118:14–22.