Universidad Católica de Santa María

Facultad de Ciencias e Ingenierías Físicas y Formales

Escuela Profesional de Ingeniería Mecánica, Mecánica - Eléctrica y Mecatrónica

DISEÑO DE UN ALGORITMO DE RECONOCIMIENTO DE PLACAS UTILIZANDO REDES NEURONALES PARA OPTIMIZAR EL USO DE LA ORDENANZA MUNICIPAL N° 927-MPA

Tesis presentada por el Bachiller:

Ballón Benavente, Bruno

para optar el Título Profesional de

Ingeniero Mecatrónico

Asesores:

Mg. Mestas Ramos, Sergio Orlando

Arequipa- Perú

2020

UCSM-ERP

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

INGENIERIA MECANICA, MECANICA-ELECTRICA Y MECATRONI

DICTAMEN APROBACIÓN DE BORRADOR DE TESIS

Arequipa, 23 de Octubre del 2020

Dictamen: 000931-C-EPIMMEM-2020

Visto el borrador de tesis del expediente 000931, presentado por:

2015202581 - BALLÓN BENAVENTE BRUNO

Titulado:

DISEÑO DE UN ALGORITMO DE RECONOCIMIENTO DE PLACAS UTILIZANDO REDES NEURONALES PARA OPTIMIZAR EL USO DE LA ORDENANZA MUNICIPAL N° 927-MPA

Nuestro dictamen es:

APROBADO

1936 - MESTAS RAMOS SERGIO ORLANDO
DICTAMINADOR

2397 - CUADROS MACHUCA JUAN CARLOS
DICTAMINADOR

RESUMEN

El presente trabajo de investigación tiene por finalidad, dar una alternativa de solución al control de vehículos que ingresan al centro histórico, de acuerdo a las medidas de la ordenanza municipal de Arequipa N° 927-MPA.

El proyecto consta de cuatro capítulos, los mismos que serán descritos a continuación:

El **Capítulo I Fundamentación** muestra el problema a investigar, así como los objetivos generales y específicos que se esperan obtener de este estudio.

El **Capítulo II Planteamiento Teórico** contiene toda la información necesaria para abordar el problema planteado. Está dividido en cuatro temas principales, dentro de cada uno se profundizará lo necesario para entender bien que es lo que estamos realizando. Se incluyen temas como inteligencia artificial, Redes neuronales, procesamiento de imágenes y software necesario.

El **Capítulo III Planteamiento Practico** contiene a detalle todo es proceso de creación del algoritmo en un notebook con lenguaje Python, que nos ayudara a resolver el problema. El algoritmo está dividido en 3 partes. La primera nos habla de la detección de placas usando redes neuronales convolucionales la segunda parte conta de reconocer el texto de las placas localizadas y la parte final crea una tabla en Excel indicando si un vehículo debió o no ingresar al centro histórico.

El **Capítulo IV Resultados** contiene los resultados obtenidos una vez aplicado el algoritmo para ver si se lograra aplicar este proyecto. Se divide en 2 partes, la primera en la cual se verá si se obtuvo buenos resultados al momento de detectar la ubicación de la placa, y la segunda parte para ver si se obtuvo buenos resultados al reconocer el texto

Palabras claves:

Redes neuronales, Procesamiento de imágenes, Ordenanza municipal N° 927-MPA, Python.

ABSTRACT

The purpose of this research work is to provide an alternative solution to the control of vehicles that enter the historic center, according to the measures of Arequipa's municipal ordinance No. 927-MPA.

The project consists of four chapters, the same ones that will be described below:

Chapter I Foundations shows the problem to be investigated, as well as the general and specific objectives that are expected to be obtained from this study.

Chapter II Theoretical Approach contains all the information necessary to address the problem. It is divided into four main themes, within each one we will deepen what is necessary to fully understand what we are doing. Topics such as artificial intelligence, neural networks, image processing, and required software are included.

Chapter III Practical Approach contains in detail all the process of creating the algorithm in a notebook with Python language, which will help us solve the problem. The algorithm is divided into 3 parts. The first part tells us about the detection of plates using convolutional neural networks, the second part is about recognizing the text of the located plates and the final part creates a table in Excel indicating whether or not a vehicle should have entered the historic center.

Chapter IV Results contains the results obtained after applying the algorithm to see if this project could be applied. It is divided into 2 parts, the first in which it will be seen if good results were obtained when detecting the location of the plate, and the second part to see if good results were obtained when recognizing the text

Key words:

Neural Networks, Image Processing, Municipal Ordinance No. 927-MPA, Python.

INTRODUCCIÓN

Uno de los problemas más graves en la ciudad de Arequipa es el incremento inmesurado del parque automotor, el cual genera contaminación, congestión y malestar por parte de la ciudadanía al momento de trasladarse de un lugar a otro. Esto se debe a la gran cantidad de población y a la falta de rutas alternas que ayuden a evitar la congestión de vehículos, siendo el centro histórico uno de los lugares más perjudicados por su ubicación estratégica.

Como alternativa de solución, la municipalidad provincial de Arequipa decidió tomar diferentes medidas, tales como la implementación del Sistema Integrado de Transporte (SIT) y la restricción del ingreso vehicular al centro histórico, por día, según número de placa.

Esta última medida, regulada según Ordenanza Municipal N° 927-MPA, cuenta con serios problemas al momento de entrar en vigor, debido a la dificultad de controlar el ingreso de los vehículos permitidos al centro histórico por parte del personal policial, y agravada por la falta de conciencia de los conductores. Por esta razón, se enfoca la presente tesis en la optimización de este control.

Figure 1: Mapa de delimitación del centro histórico

La solución a este problema se encuentra en el uso de la tecnología; si en una industria la solución lógica sería la automatización de la planta, en este caso, es necesario identificar a todo aquel que incumpla la medida para sancionarlo y entienda el malestar que genera, para ello propongo usar los últimos avances en inteligencia artificial (IA), como vemos más adelante los principales campos de estudio de la IA son: Lógica difusa, Redes Neuronales Artificiales y Algoritmos genéticos.

Actualmente, existen estudios, como los realizados por Mg. Ing. Kristians Edgardo Díaz Rojas, Mg. Ing. Miguel Angel Cataño Sánchez y Gerardo Alfonso Joel Espinoza Vásquez, que ayudan a detectar la ubicación de una placa mediante el procesamiento de imágenes. Estos tienen en común el uso de la transformada de Hough para detectar la ubicación de la placa, ya que esta ayuda a detectar bordes, siempre y cuando estos sean del mismo tamaño.

Esto significa que la ubicación de los vehículos siempre debe ser la misma y no debe existir ángulo de inclinación en la toma, resultando difícil operacionalmente. También existe un estudio realizado por Alvaro Gonzalo Delgado Boza, el cual utiliza lógica difusa para la detección de caracteres en una placa, sin embargo, la imagen de entrada debe ser únicamente la placa y esto dificulta su aplicación. Hay empresas privadas como Neural Labs, Hikvision o ZKTeco que proponen detectar las placas de los vehículos usando redes neuronales, sin embargo, estas incluyen el uso de cámaras especiales y costosas.

Según la Unesco, BBC, Ayming entre otros, se aseguran de que la inteligencia artificial, junto con las redes neuronales, es parte de una 4ta revolución industrial, generando un avance sin precedentes, mayor aún que la creación de Internet. HP define a la inteligencia artificial como *"Un conjunto de disciplinas de software, lógica, informática y filosofía que están destinadas a hacer que los PC realicen funciones que se pensaba que eran exclusivamente humanas; como percibir el significado en el lenguaje escrito o hablado, aprender, reconocer expresiones faciales, etc."*(hewlett packard enterprice, n.d.).

En este estudio se propone realizar un algoritmo utilizando redes neuronales, pues es uno de los campos de la IA que está tomando más relevancia en la actualidad. La invención de las redes neuronales data de 1956, sin embargo, estas no fueron aprovechadas debido a la necesidad de cantidades importantes de recursos de un ordenador. Actualmente, habiendo superado este obstáculo, estas ofrecen capacidad de aprendizaje, proporcionando grandes soluciones en los siguientes aspectos:

- Reconocimiento de caracteres
- Reconocimiento de imágenes
- Reconocimiento de voz
- Realización de predicciones
- Generación de texto

- Traducción de idiomas
- Análisis genético
- Etc.

El algoritmo propuesto funcionará de la siguiente manera:

1. Se ingresa una imagen al algoritmo.
2. Esta pasa por una etapa de preprocesamiento.
3. La imagen procesada ingresa a una red neuronal, la cual dará como resultado las coordenadas de ubicación de la placa dentro de la imagen.
4. Se recorta la imagen de acuerdo a las coordenadas obtenidas.
5. Esta ingresa a otra red neuronal, la cual detecta los caracteres requeridos dentro de la placa.
6. Finalmente, se exporta una tabla en Excel con el registro de placas ingresadas y cuales incumplieron la ordenanza.

Figure 2: Esquema de trabajo de Investigación

Este proceso se hará Offline para su posterior sanción. Realizarlo Online no sería eficiente, dado que implicaría sancionar en el momento o en su defecto evitar el paso de los infractores, con lo cual se ocasionaría más tráfico.

Cabe destacar que, en la presente tesis, no se tomará en cuenta la instalación de cámaras alrededor del centro histórico, y, por ende, la captura de imágenes; esta se enfocará en el desarrollo del algoritmo empezando por el procesamiento de las imágenes. Las capturas que se utilizarán para el entrenamiento y prueba de las redes neuronales serán tomadas con un celular (Moto G8 Plus).

ÍNDICE

ÍNDICE DE ILUSTRACIONES

1. FUNDAMENTO

1.1. PROBLEMA DE INVESTIGACIÓN

1.1.1. Determinación del problema

En la ciudad de Arequipa encontramos un serio problema en el día a día al momento de desear trasladarnos de un lugar a otro. Las autoridades decidieron volver peatonal algunas calles del centro histórico mediante la ordenanza municipal N° 556-2008, la cual entro en vigor en el año 2017, para proteger la transitividad en el centro ya que esta fue declarada patrimonio cultural de la humanidad por la UNESCO. Asimismo, restringió el acceso a las demás calles del centro histórico dependiendo del número de placa de los vehículos según la ordenanza N° 927-MPA, puesta en funcionamiento en el año 2019. Por lo cual surgieron interrogantes que siguen hoy en día ¿Cómo controlaran el ingreso de estos?

1.1.2. Enunciado del problema

¿Cómo podemos detectar a los automóviles que infringen la ley cuando no les corresponde su ingreso al centro histórico?

1.1.3. Descripción del problema

- La vigilancia del plaqueo es hecha por personas.
- Falta de personal policial.
- Dificultad de control en todas las entradas al centro historio.
- El tiempo de vigilancia es prolongado.
- La vigilancia no está sistematizada.
- Falta de cultura y/o ética profesional

Según lo ordenado por las autoridades, los días lunes pueden ingresar vehículos cuyas placas terminen en cero y uno; los días martes, dos y tres; miércoles, cuatro y cinco; jueves, seis y

siete; viernes, ocho y nueve; y finalmente, permitiendo los fines de semana transitar a cualquier vehículo. ¿Pero cómo podemos controlar que estas medidas se cumplan?

Imagen 1: Esquina Av. La paz con Don Bosco

1.2. Justificación

Existe una norma la cual permite controlar el caos vehicular; sin embargo, su implementación es deficiente; por lo que es necesario optimizar el control de ingreso de los vehículos usando la tecnología existente al servicio de la comunidad para la solución de diferentes problemas urbanos.

1.3. OBJETIVOS

1.3.1. Objetivo general

- Formular un algoritmo en Python utilizando redes neuronales que dé como resultado un registro del número de placa de los conductores que infringen la ordenanza municipal N° 927-MPA.

1.3.2. Objetivos específicos

- Analizar los diferentes tipos de redes neuronales.
- Crear una base de datos de imágenes y texto para el entrenamiento de la red neuronal.
- Seleccionar los parámetros correctos para el procesamiento de las imágenes.
- Diseñar y entrenar una red neuronal usando arquitecturas existentes como Resnet o Yolo.
- Explicar porque el uso del lenguaje de programación.

2. PLANTEAMIENTO TEÓRICO

2.1. Inteligencia Artificial

Definir la inteligencia artificial (IA) ha de ser muy confuso, esto debido a que definir la misma incluye definir el concepto de Inteligencia, el cual biólogos y científicos de todo el mundo no han definido con toda totalidad. Tomando el concepto de la Real Academia de la Lengua Española (RAE) *"Inteligencia Artificial: Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico* "(RAE Inteligencia Artificial, 2020).

Por lo que la inteligencia artificial es la capacidad de imitar el comportamiento humano es decir *"habilidad de enfrentar nuevas situaciones, la habilidad de resolver problemas, de responder preguntas, elaborar planes, etc."* (Ponce Cruz, 2010)

El ser humano pretende con el estudio de la IA, construir entidades inteligentes. Los cuales ocupen campos como el aprendizaje y percepción, o algunos más específicas como el ajedrez, escritura, demostración de teoremas o diagnóstico de enfermedades.

A lo largo de la historia se han seguido 4 enfoques diferentes, como definen en el libro de Russel.

Sistemas que piensan como humanos	Sistemas que piensan racionalmente
«El nuevo y excitante esfuerzo de hacer que los computadores piensen… máquinas con mentes, en el más amplio sentido literal». (Haugeland, 1985) «[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje…» (Bellman, 1978)	«El estudio de las facultades mentales mediante el uso de modelos computacionales». (Charniak y McDermott, 1985) «El estudio de los cálculos que hacen posible percibir, razonar y actuar». (Winston, 1992)
Sistemas que actúan como humanos	**Sistemas que actúan racionalmente**
«El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren de inteligencia». (Kurzweil, 1990) «El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor». (Rich y Knight, 1991)	«La Inteligencia Computacional es el estudio del diseño de agentes inteligentes». (Poole *et al.*, 1998) «IA… está relacionada con conductas inteligentes en artefactos». (Nilsson, 1998)

Imagen 2: Enfoques de la IA

Fuente: Rusell & Norvig, (2004)

Como método de medición de la IA, existe la prueba de Turing, la cual fue desarrollada por Alan Turing en 1950, en la cual dice que, para definir la inteligencia, un sistema debe estar conformado por 6 campos:

1. Procesamiento de lenguaje natural: para entender el lenguaje inglés.
2. Representación del conocimiento: para almacenar conocimiento y sentimientos.
3. Razonamiento automático: para usar la información almacenada para responder preguntas.
4. Aprendizaje Automático: para extrapolar patrones y adaptarse a nuevas circunstancias.
5. Visión computacional: para visualizar los objetos.
6. Robótica: para poder manipular los objetos.

Dentro de las ramas que estudian la inteligencia artificial encontraremos tres grandes campos:

1. Lógica difusa
2. Redes Neuronales Artificiales

3. Algoritmos genéticos

2.2. Historia de la Inteligencia Artificial

Para saber cuáles fueron los primeros pasos hacia la IA nos remontarnos a Aristóteles en 384-322 a.C, ya que intento explicar y codificar un estilo de razonamiento deductivo llamado silogismos. Sin embargo, no fue hasta 1958 cuando John McCarthy; quien introdujo el termino de inteligencia artificial; propuso usar el cálculo proposicional como un idioma que represente el conocimiento en un sistema llamado “Advise Taker.”.

Continuando con su trabajo tenemos a grandes lógicos como Kart Codel, Stephen Kleene, Emili Post, Alonzo Church y Alan Turing, que en el siglo XX profundizaron en el tema y aclararon gran parte de los que se podía lograr y lo que no se podía lograr en esa época. Estudios más cercanos al siglo XXI, como los realizados por los científicos Stephen Cook y Richard Karp, mostraron clases de cálculos que parecían posibles de realizar, pero necesitaban mucho tiempo y memoria de almacenamiento.

2.3. Redes Neuronales Artificiales

Las redes neuronales artificiales (RNA) también conocidas como (Conexionismo, procesamiento distribuido paralelo o computación neuronal), así como, las otras ramas de la inteligencia artificial, intentar imitar el comportamiento del cerebro humano.

Una neurona en una célula del cerebro cuya función es recoger, procesar y emitir señales eléctricas. Se creé que la capacidad del cerebro para procesar la información está relacionada a las redes neuronales del cerebro. Es por ello que, se pretenden crear redes neuronales artificiales.

Para que una red neuronal artificial funcione es necesario que esta tenga ejemplos de problemas similares, de los cuales pueda aprender, para que posteriormente pueda extrapolar esta información y resolver problemas más complejos.

2.3.1. Historia de las RNA

En 1943 Warren McCulloch y Walter Pitts propusieron el primer modelo simple de neurona obteniendo como resultado un empate entre la neuropsicología y la lógica matemática. Posteriormente en 1950 B. Windrow y M.E.Hoof trabajaron con una maquina llamada Adaline (Adaptive Linear Element).

Posteriormente en 1956 se realizó la conferencia de Dartmouth, con motivo de estudio Inteligencia Artificial, esta tuvo como premisa que cualquier aspecto de aprendizaje o cualquier otro rasgo inteligente puede ser descrito con tanta precisión, que una maquina podría simularlo, así como la definición de ese entonces “cada aspecto del aprendizaje y cada característica de la inteligencia podían ser tan precisamente descritos que se podían crear máquinas que las simularan.” de donde se destaca que inteligencia artificial es imitar comportamientos inteligentes.

Después de esa conferencia entre 1974 y 1980 la inteligencia tuvo una era oscura conocida como el invierno de la inteligencia artificial debido a que no se lograron los frutos esperados y se dejó de invertir en este campo; uno de los intentos fallidos fue el de Marvin Minsky, quien escribió el libro Perceptrón en el cual nos habla de cómo al no contar con la tecnología suficiente, diseñar una red neuronal se podía volver muy tedioso y lento en un computador.

En la década de los 80’s se retomó el estudio en este campo sin embargo estos solo se basaban en algoritmos básicos y no fue hasta 1996 que se logró una verdadera motivación para continuar con el estudio, esta fue lograr vencer al a Garri Kasparov por un ordenador de IBM “Deep blue” en una partida de ajedrez, campo el cual se consideraba de gran habilidad cognitiva.

En 1995 David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams publicaron un artículo titulado “Learning Representations by Back-Propagating errors” en el cual mencionan el concepto de Back-Propagation o retroalimentación, mencionado por primera vez 1969 por Bryson y Ho, el cual revivió a las redes neuronales. Con esta técnica; el cálculo tedioso y largo se volvió mucho más sencillo mediante el uso de derivadas.

Las redes neuronales como el propio Marvin Minsky declaró, están inspiradas en el ser humado precisamente en las neuronas; donde las dendritas vienen a ser nuestra información de entrada y el axón nuestra salida.

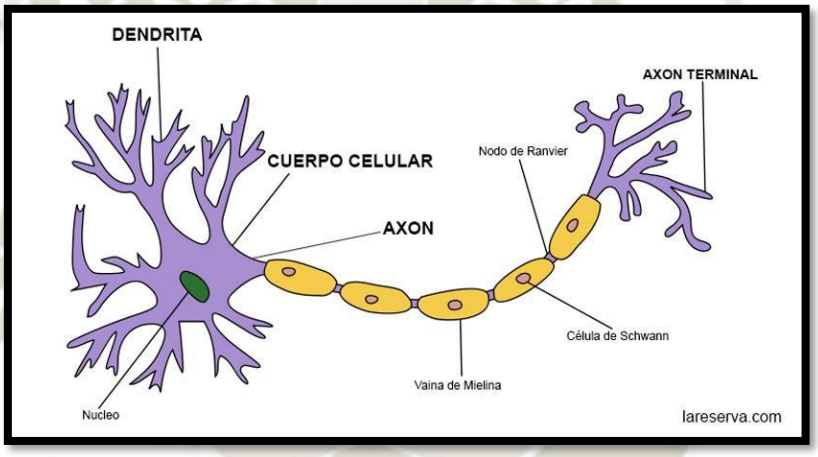

Imagen 3:Neurona

Fuente: LaReserva.com

2.3.2. ¿Qué son las Redes Neuronales artificiales?

Las redes neuronales Artificiales (RNA) generalizan el conocimiento entregado por el ser humano, ya sea de tablas, bases de datos, imágenes, etc. Con el fin de obtener una relación entre ellos la cual puedan aplicar más adelante para predecir un resultado o acontecimiento.

2.3.3. Ventajas de una red neuronal artificial

- Aprendizaje Adaptativo: Capacidad de ir aprendiendo con el entrenamiento
- Autoorganización: Crea su propia forma de representar la información

- Tolerancia a fallos: La destrucción de una parte de la red no provoca que se pierdan todas las propiedades de la red
- Operación en tiempo real: La computación neuronal se puede realizar en paralelo (muy rápido).
- Fácil inserción dentro de la tecnología existente: Se pueden desarrollar chips especializados en esta tecnología facilitando su inserción en el mundo actual

2.3.4. Elementos básicos de una red neuronal artificial

Una RNA está compuesta por dos partes como se puede observar en la imagen 4:

- Capas
- Pesos

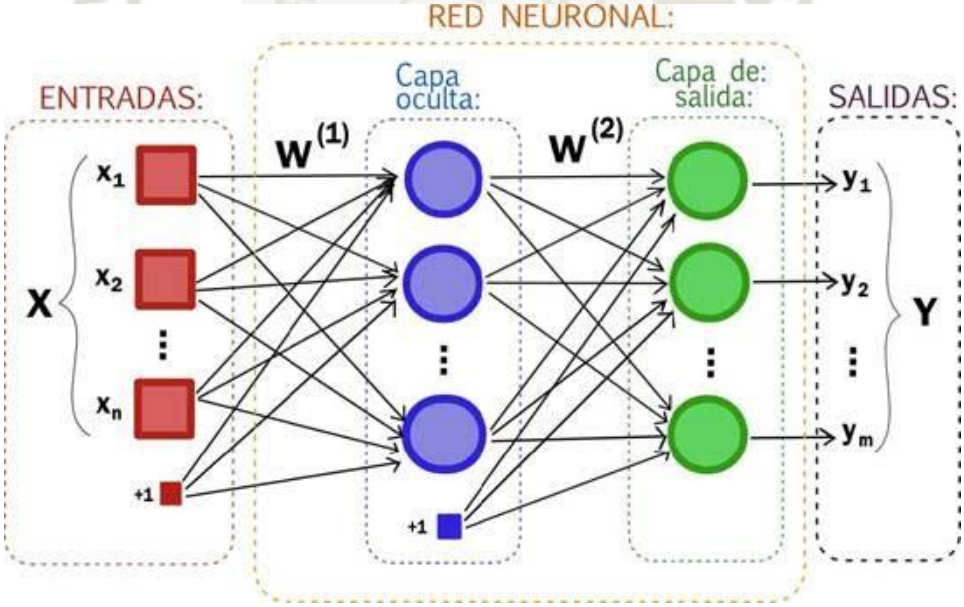

Imagen 4: RNA

Fuente: Google sites

Las capas de una RNA están divididas en tres:

- Capa de entrada.
- Capa(s) oculta(s).

- Capa de salida.

Estas se pueden observar en la imagen 5.

Cabe resaltar que la capa de salida no siempre será una, esta dependerá del número de salidas que tenga nuestro problema, es decir si tenemos un problema en el cual nuestra salida sea binaria 1 o 0 tendremos dos neuronas en la capa de salida.

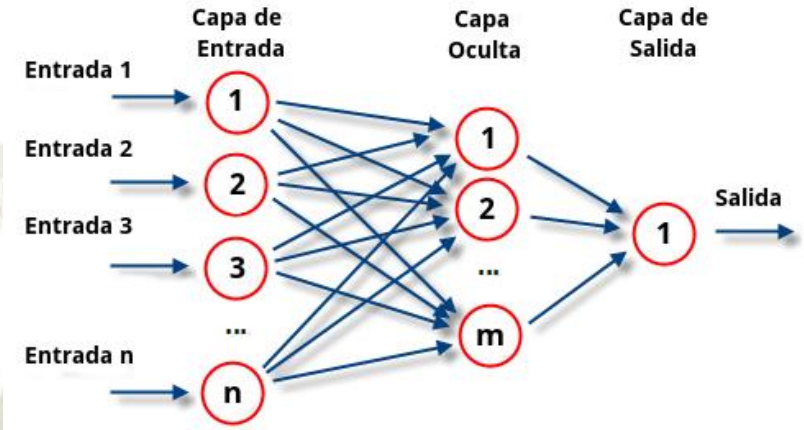

Imagen 5: RNA

Fuente:Google images

La cantidad de pesos en la imagen 5 será:

$$N * M + M * S$$

Donde:

- M: número de neuronas en la capa oculta
- N: número de neuronas en la capa de entrada
- S: Número de neuronas en la salida

Esta fórmula variara dependiendo del número de capas ocultas que tenga la red neuronal

2.3.5. ¿Qué es un perceptrón?

Un perceptrón es lo que vemos como una neurona, es la unidad básica de una RNA. Este es la suma de señales de entrada multiplicada por los valores de los pesos inicialmente escogidos de manera aleatoria. Este pasa por una etapa de aprendizaje el cual denominaremos entrenamiento, donde la entrada se compara con un patrón preestablecido que determina una salida y se calcula el error.

Al inicio el perceptrón no está en la capacidad de distinguir patrones de entrada muy complejos, sin embargo, con el entrenamiento este será capaz de adquirir esta capacidad. El entrenamiento consiste en variar los valores de los pesos que codifican la sinapsis, estos pueden incrementar o disminuir.

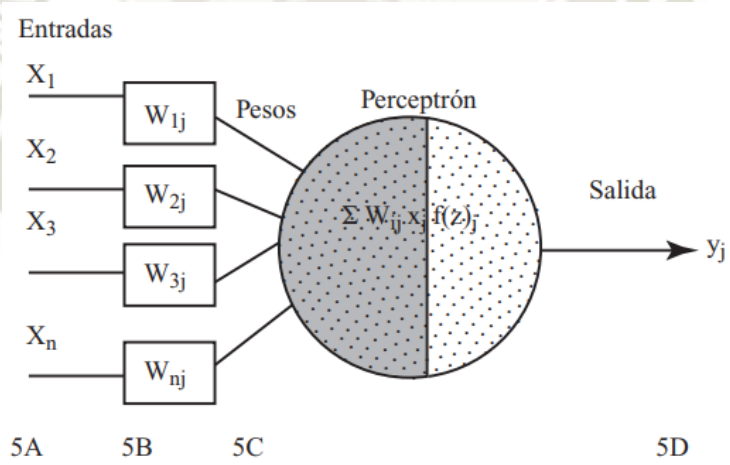

Imagen 6: Perceptrón

Fuente: Ponce Cruz (2010)

En la imagen 6 se puede observar las entradas, los pesos, el perceptrón y la salida donde se cumple que:

$$y = f(z) * \sum (W_{ij} * X_i)$$

Donde:

- f(z): Función de activación
- W_{ij}: Pesos
- X_i: Valores de entrada
- y: Salida

2.3.6. Función de activación

La función de activación es un parámetro que ayuda, a que la función obtenida por las entradas y pesos no sea una regresión lineal.

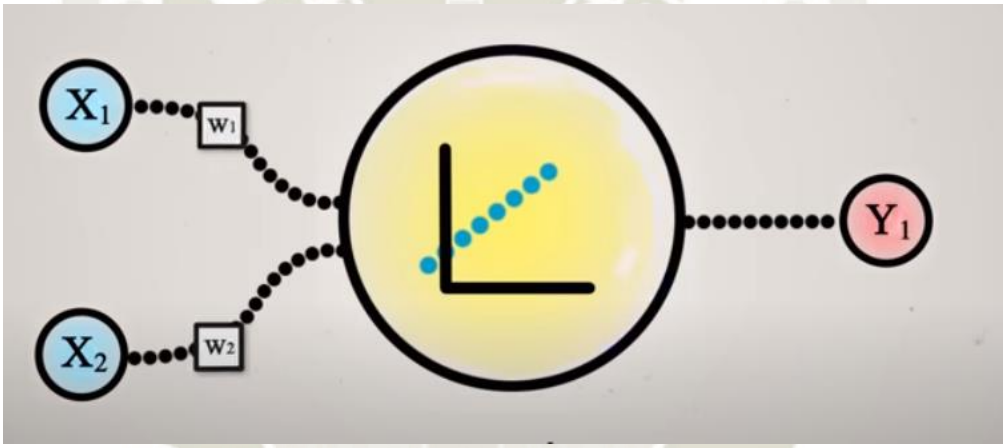

Imagen 7: Perceptrón básico

Fuente: DotCSV (2018)

La salida de la imagen 7 es:

$$y_1 = X_1 * W_1 + X_2 * W_2$$

La ecuación da como resultado una línea recta, a esta se le puede sumar un parámetro b el cual conocemos como bias para variar la posición de esta línea.

Esto puede parecer sencillo para ciertos problemas, otros problemas se resolverán usando más neuronas sin función de activación, como por ejemplo la función XOR se resolverán usando dos neuronas como observamos en la imagen 8.

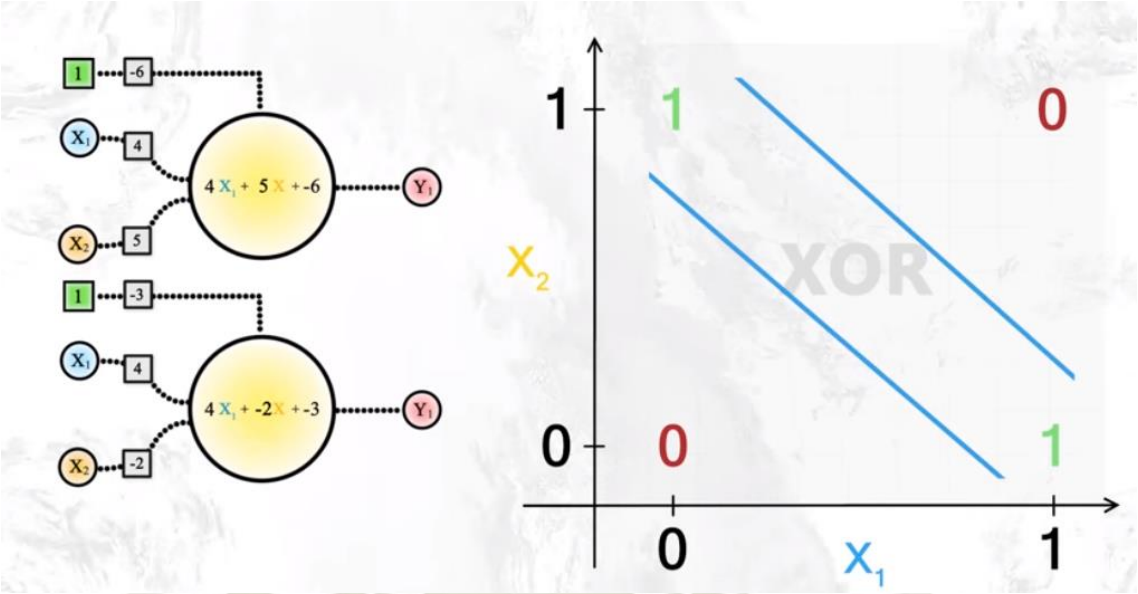

Imagen 8: Problema XOR

Fuente: DotCSV (2018)

Ahora, Si tenemos una red y esta está compuesta por funciones que al final dan una línea recta como resultado tendremos una línea recta lo cual no siempre es la solución y es ahí donde entra la importancia de una función de activación.

Dentro de las principales están:

1. Función Sigmoidea: En esta la función de activación f(x) será:

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

2. Función Tangente hiperbólica: En esta la función de activación f(x) será:

$$f(x) = \frac{2}{1 + e^{-2x}}$$

3. Función Rectificada Lineal (RELU) En esta la función de activación f(x) será:

$$f(x) = \max(0, x) = \begin{cases} 0 \; para \; x < 0 \\ x \; para \; x \geq 0 \end{cases}$$

Estas nos ayudan a obtener la no linealidad necesaria para resolver diferentes problemas observemos la imagen 9 e imagen 10.

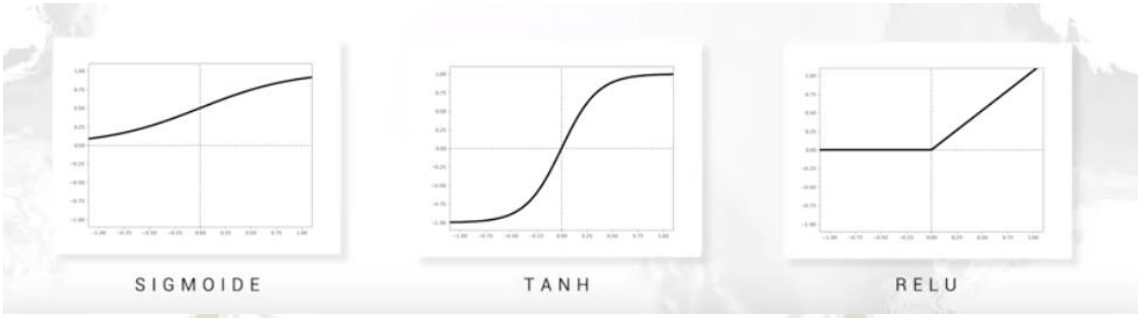

Imagen 9:Funciones de activación

Fuente: DotCSV (2018)

Imagen 10:Funciones de activación

Fuente: DotCSV (2018)

Si combinamos estas funciones de activación vamos a obtener funciones como la que se observan en la imagen 11.

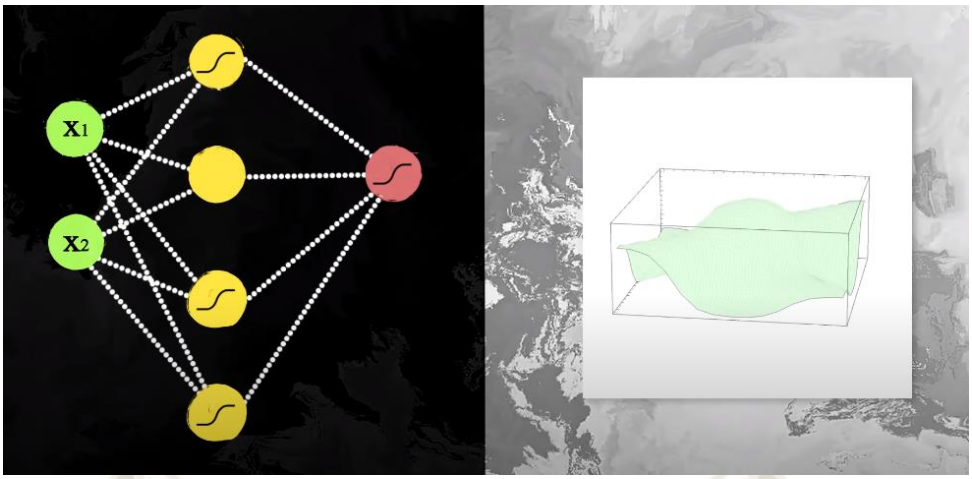

Imagen 11: Función de activación

Fuente: DotCSV (2018)

Tomando como ejemplo la imagen 7 al agregarle una función de activación la salida varia siendo y.

$$y_1 = f(x) * (X_1 * W_1 + X_2 * W_2)$$

Cabe resaltar que la función de activación se aplicara por cada capa. Existen otras funciones de activación un poco más complejas como:

- Leaky ReLU
- ELU
- Softmax
- Etc.

2.3.7. Entrenamiento de una red neuronal

La retroalimentación más conocida por su nombre en inglés backpropagation. Es un algoritmo el cual nos permite actualizar los pesos y ganancias de una manera muy sencilla. Este algoritmo funciona bajo aprendizaje supervisado, es decir tiene que conocer la salida de la red y el valor de salida esperado.

El entrenamiento es iterativo, en cada interacción los pesos varían usando nuevos pesos para cada nuevo conjunto de datos de entrenamiento. Los nuevos pesos se calculan usando el algoritmo de retropropagación.

Para entender mejor observaremos la imagen 12 donde vemos como se calcula los valores de salida de cada neurona.

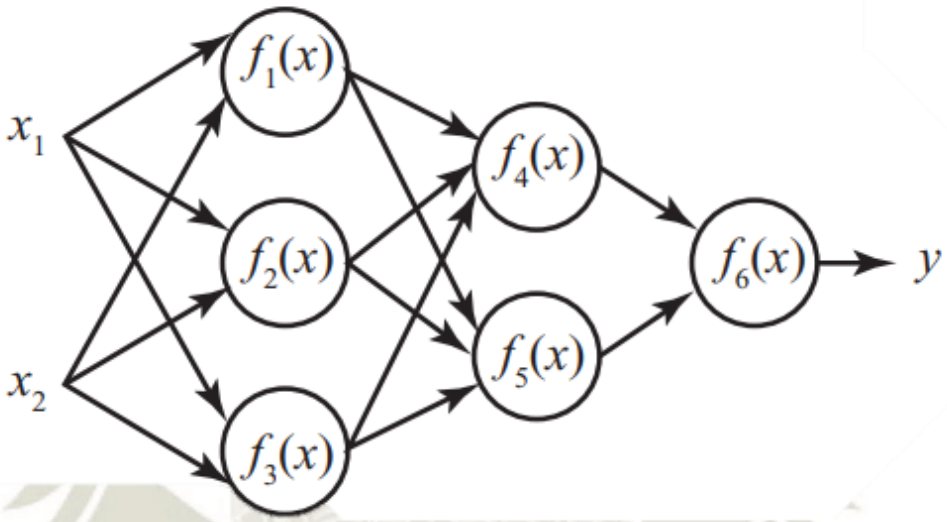

Imagen 12: Salida de una neurona

Fuente: Ponce Cruz (2010)

Donde:

$$y_5 = f_5 * (W_{15} * y_1 + W_{25} * y_2 + W_{35} * y_3)$$

Siendo:

- yi: Salida neurona i
- f5: Función de activación de la capa oculta
- Wij: Peso desde la neurona i hasta la neurona j

Una vez obtenida la salida de la última neurona se calcula la diferencia entre la salida obtenida y la salida esperada la cual denominaremos δ. Y con δ se obtiene el error en cada neurona como se muestra en la imagen 13 y 14.

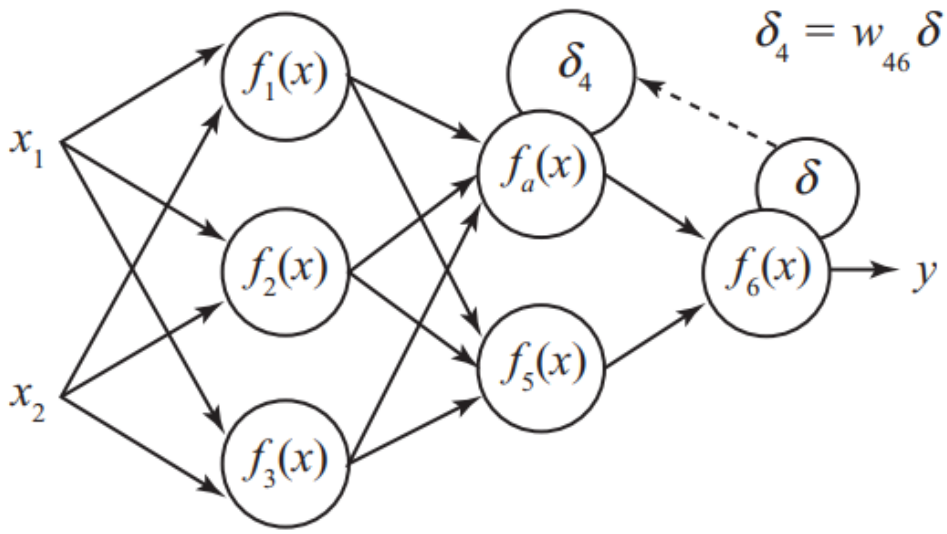

Imagen 13: Calculo del error

Fuente: Ponce Cruz (2010)

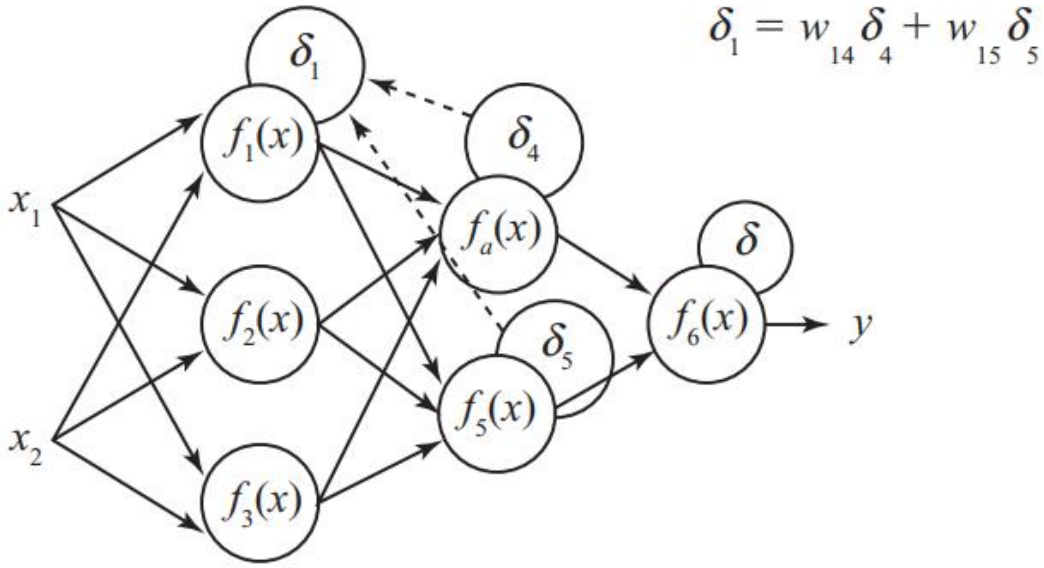

Imagen 14: Calculo de error algoritmo de retropropagación
Fuente: Ponce Cruz (2010)

Finalmente se calculan los nuevos pesos y se actualizan los valores dentro de nuestra red neuronal como se muestra en la imagen 15.

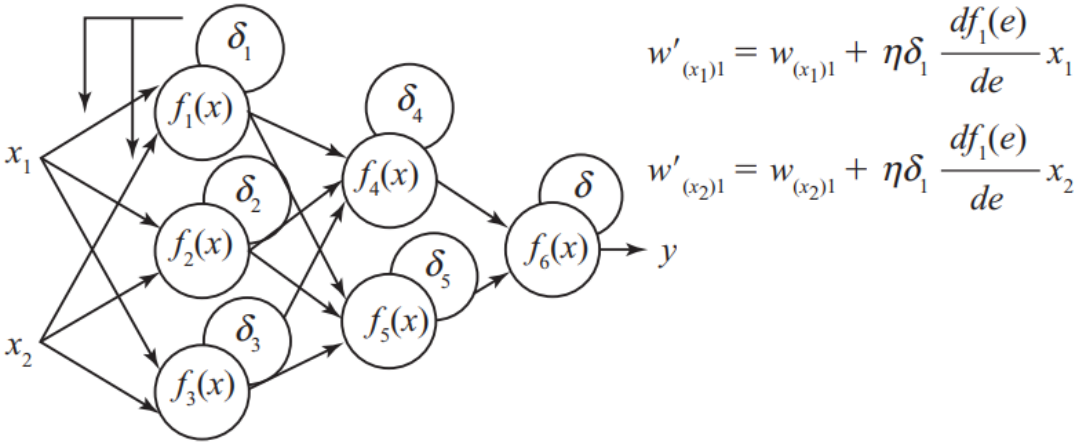

Imagen 15: Calculo de pesos

Fuente: Ponce Cruz (2010)

Donde η es un coeficiente que define la velocidad de aprendizaje, si este es un valor grande este se ira definiendo a medida pasan las épocas de entrenamiento, si es un valor pequeño este crecerá y luego volverá a disminuir.

2.3.8. Tipos de redes neuronales artificiales

Existen diferentes tipos de redes neuronales según su topología:

- Red neuronal monocapa: también conocido como perceptrón simple, es el tipo de red neuronal más simple, está compuesta solo por la capa de entrada y salida.

Imagen 16: Perceptrón Simple

Fuente: Calvo (2017)

- Red neuronal multicapa: también llamado perceptrón multicapa, este además de la capa de entrada y salida también cuenta con una o varias capas ocultas.

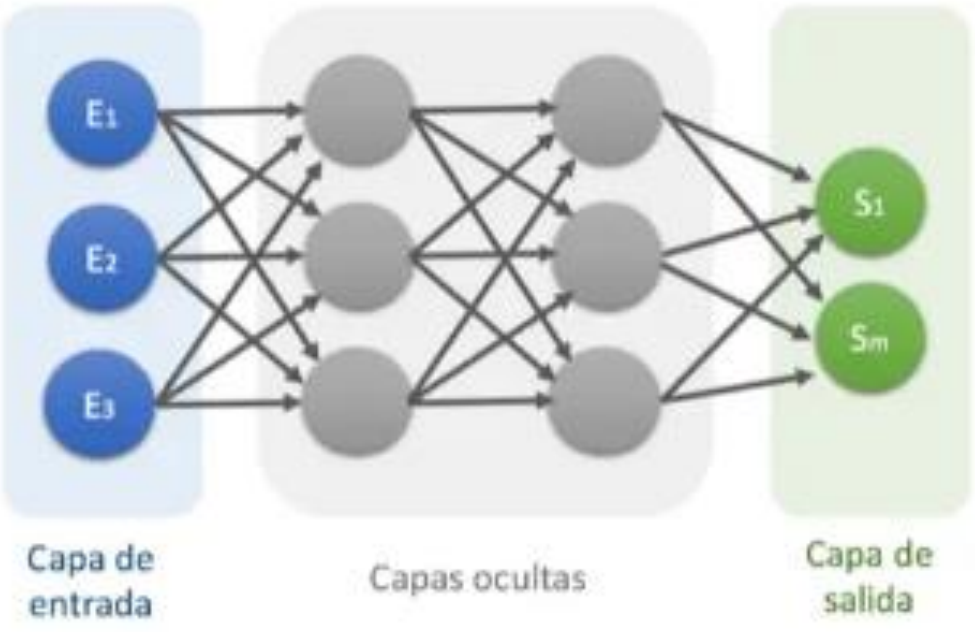

Imagen 17: Perceptrón Multicapa

Fuente: Calvo (2017)

- Red neuronal convolucionales: más conocido por sus siglas en ingles CNN (Convolutional Neural Networks), *"la principal diferencia de la red neuronal convolucional con el perceptrón multicapa viene en que cada neurona no se une con todas y cada una de las capas siguientes sino que solo con un subgrupo de ellas (se especializa), con esto se consigue reducir el número de neuronas necesarias y la complejidad computacional necesaria para su ejecución."*(Calvo, 2017)

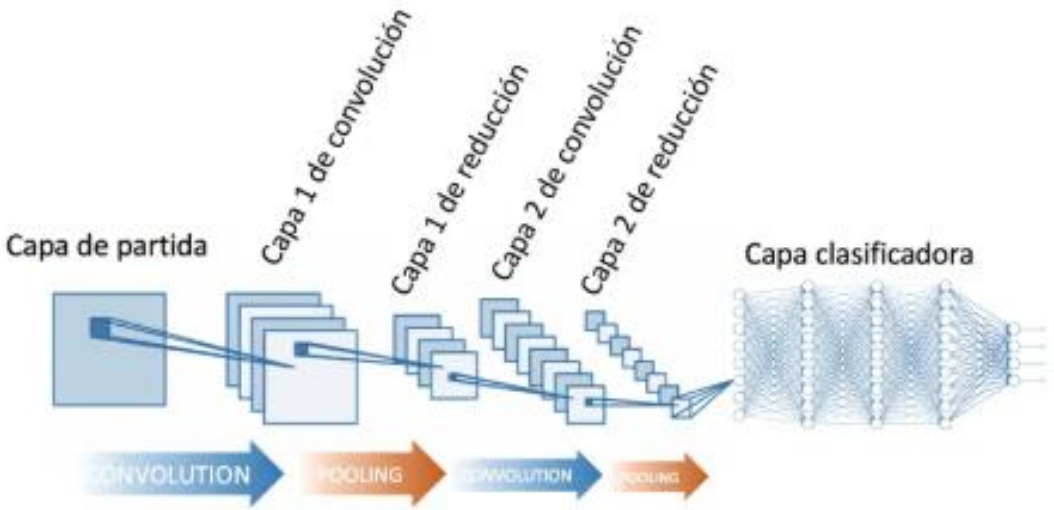

Imagen 18: CNN

Fuente: Calvo (2017)

- Red neuronal recurrente: o por sus siglas en ingles RNN, estas se diferencian ya que no tienen una estructura de capas sino tienen conexiones arbitrarias entre neuronas pudiendo crear ciclos, con lo cual obtienen crear una temporalidad, lo cual nos da memoria.

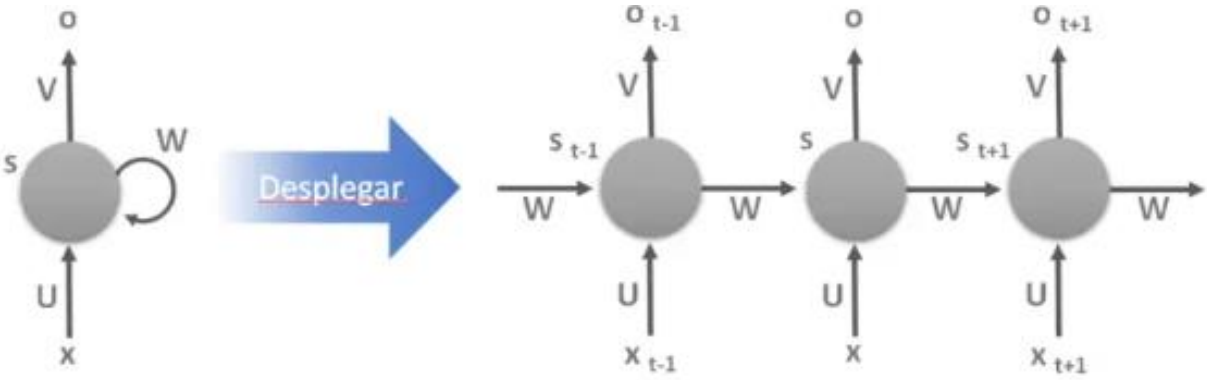

Imagen 19: RNN

Fuente: Calvo (2017)

- Redes de base radial: *"Las redes de base radial calculan la salida de la función en función de la distancia a un punto denominado centro. La salida es una combinación lineal de las funciones de activación radiales utilizadas por las neuronas individuales. Las redes de base radial tienen la ventaja de que no presentan mínimos locales donde la retropropagación pueda quedarse bloqueada."* (Calvo, 2017)

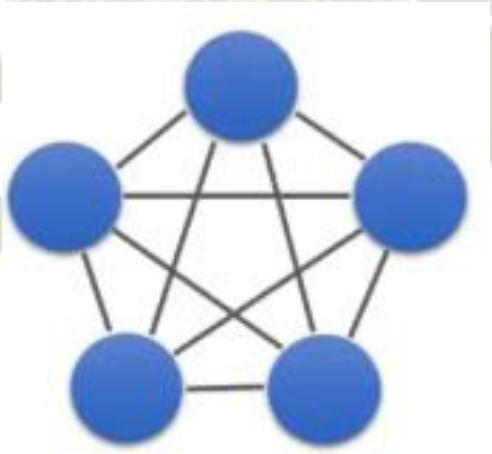

Imagen 20: RFB

Fuente: Calvo (2017)

También se pueden dividir por su método de aprendizaje:

- Aprendizaje supervisado: se divide en
    - Aprendizaje por corrección de error

- Aprendizaje estocástico

- Aprendizaje no supervisado: se divide en
  - Aprendizaje hebbiano
  - Aprendizaje competitivo y comparativo
- Aprendizaje por refuerzo

2.3.8.1. Aprendizaje supervisado:

Este se caracteriza por tener un entrenamiento controlado, es decir, conoce los valores de entrada y salida, si la salida no es correcta modifica los pesos para que esta sea correcta. En el aprendizaje por corrección de error se usa backpropagation. Mientras que en el estocástico varía los pesos de manera aleatoria.

2.3.8.2. Aprendizaje no supervisado:

A diferencia del aprendizaje supervisado este no conoce los valores de salida, por ello, busca una relación en los valores de entrada representando la salida en clustering o categorías.

2.3.8.3. Aprendizaje por refuerzo:

Muy parecido al Aprendizaje supervisado, la diferencia se encuentra en que no se conoce con exactitud la salida, pero se conoce si el dato es aceptable o no.

2.4. Redes neuronales convulsiónales

Las redes neuronales convolucionales (CNN), son similares a las redes neuronales normales en el sentido que aprenden los pesos y bias. La diferencia se encuentra en que una CNN fue diseñada para trabajar con imágenes por lo que tiene una parte para extraer características.

Una de las ventajas de una red neuronal convolucional es que asume que todas las imágenes se parecen lo cual permite, reducir el número de muestras necesarias.

Las CNN consiste en una secuencia de capas en la cual cada una transforma la salida de la capa anterior. Las capas más comunes en una CNN son capa de convolución, capa de agrupación y capa totalmente conectada.

Una diferencia clara con una RNA es que tienen capas de la red en dimensiones de canales, largo, ancho y número de filtros.

Las CNN están compuestas por dos partes, la parte donde se extraen características y la capa de mapeo o clasificadora.

2.4.1. Capa de convolución:

La capa de convolución es una capa en la cual se le aplica un filtro a la imagen. Esta nos devuelve la imagen original y una imagen con el filtro aplicado, lo cual aumenta la profundidad mas no altera las dimensiones de la imagen.

Como vemos en la imagen 21, la parte verde es la imagen original mientras que la parte amarilla es un filtro o también conocido como kernel.

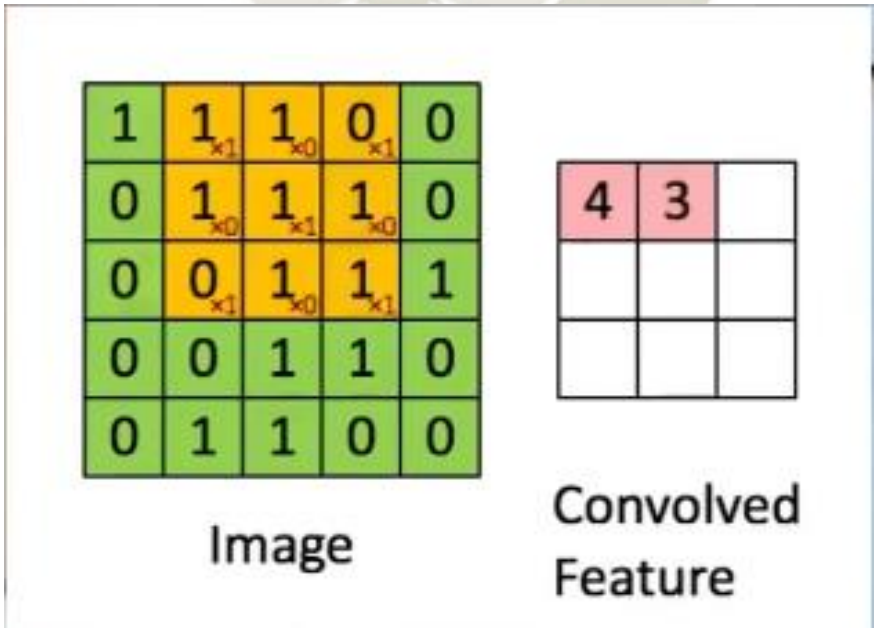

Imagen 21: Convolución

Fuente: AMPTech (2019)

El resultado de la imagen una vez aplicado el filtro como se observa en la imagen 23 es más pequeño. Para calcular el tamaño de la imagen resultante basta aplicar una simple formula.

$$m - n + 1$$

Donde

- m: Dimensión de la imagen
- n: Dimensión del filtro

Para que nuestra imagen con filtro sea del mismo tamaño que nuestra imagen original, se utiliza la técnica de padding, la cual rellena los contornos con un valor, este valor normalmente es 0 pero esto puede variar según necesidad del usuario.

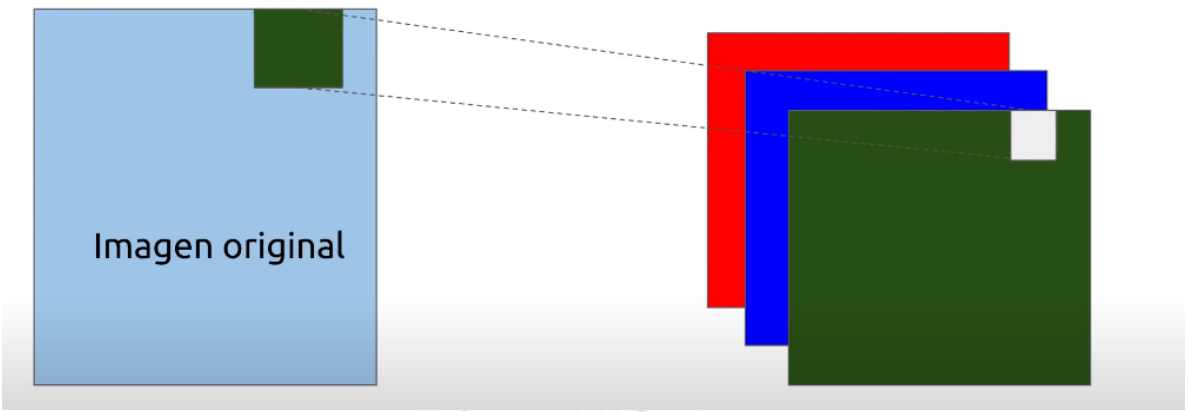

Imagen 22: Capas de Convolución

Fuente: AMPTech (2019)

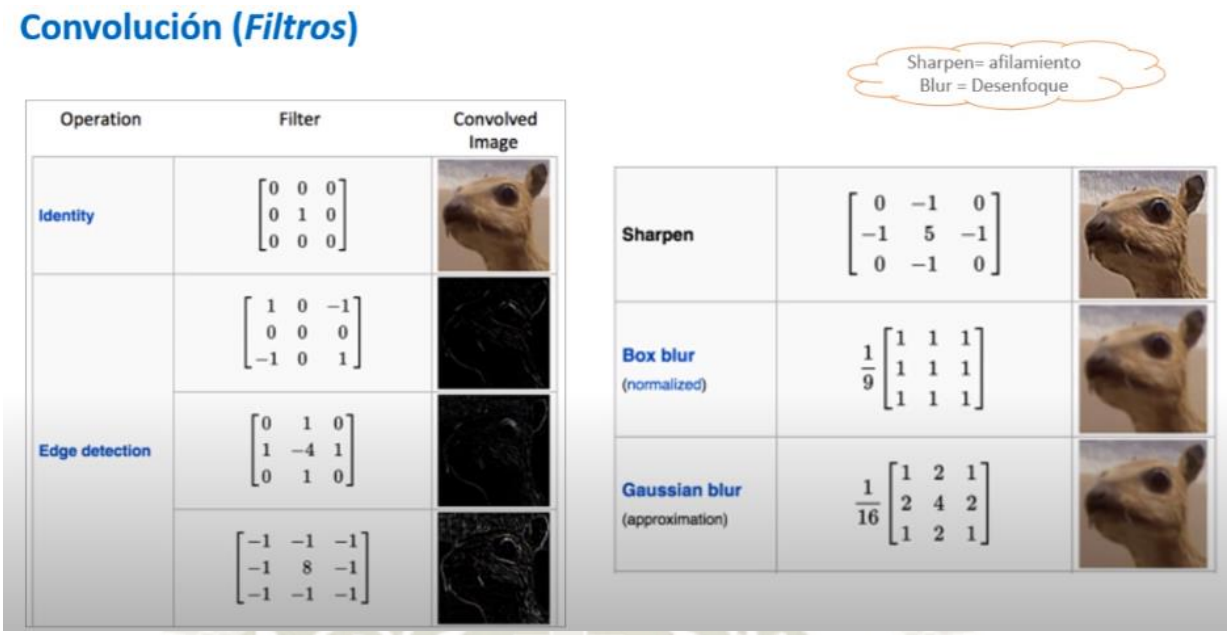

Imagen 23: Filtros convolución

Fuente: Nuñez (2019)

2.4.2. Agrupación o Pooling

La capa de pooling lo que hace es reducir el tamaño de nuestra imagen, está a diferencia de la capa de convolución no aumenta la profundidad. Para reducir el tamaño de nuestras imágenes utiliza un kernel que dependiendo si es Max Pooling o Average Pooling nos devolverá una imagen más pequeña. El tamaño de la reducción está definido por un valor conocido como stride o paso.

Imagen 24: Pooling

Fuente: AMPTech (2019)

2.4.3. Aplanado o Flatten

El Aplanado es el último paso antes de pasar a la capa clasificadora. Este convierte nuestra imagen de un formato matricial $m\ x\ n$ a una matriz $1\ x\ (m * n)$ como se observa en la imagen 27

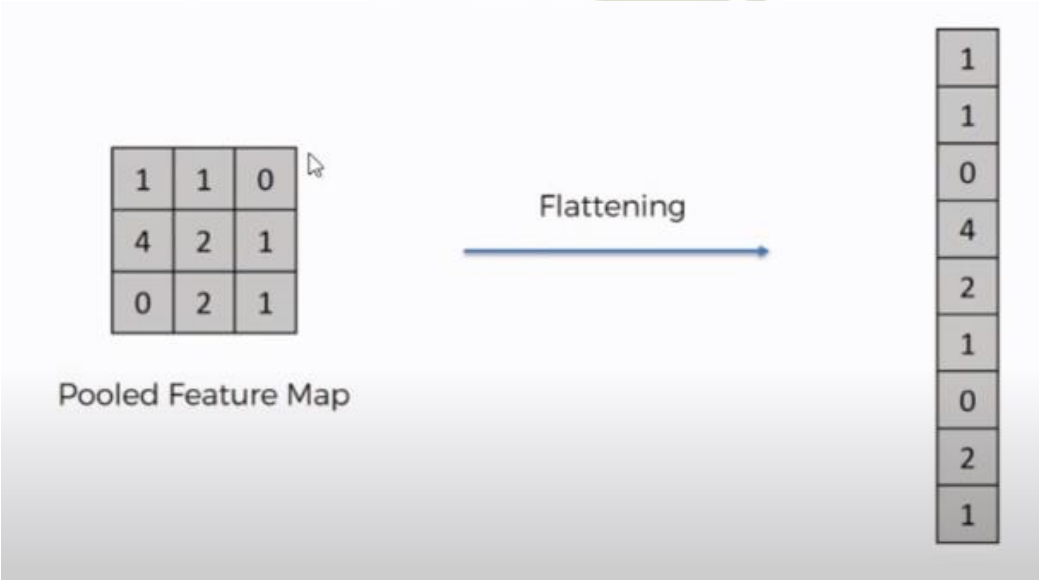

Imagen 25: Aplanado

Fuente: Nuñez (2019)

2.4.4. Capa totalmente conectada

Nuestra imagen una vez aplicada el aplanado entra a nuestra red clasificadora la cual se comporta igual que una RNA común.

2.5. Procesamiento de imágenes

2.5.1. Conceptos generales

Una imagen digital es una función de intensidad de luz $f(x, y)$ donde x y y son coordenadas dadas en pixeles

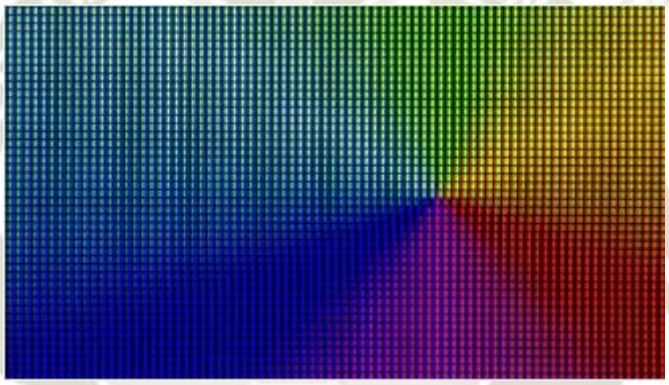

Imagen 26: Pixeles

Fuente: www.ocu.org

Una imagen RGB es una imagen digital representada en tres dimensiones, $a * b * c$ donde, a es el ancho de la imagen en pixeles, b denota el alto o largo de la imagen y c hace referencia a un plano.

Si c vale 1 hace referencia al color rojo, si es 2 verde y si es 3 azul. La unión de estos 3 colores nos darán cualquier color dependiendo de sus intensidades. Los valores de rojo, verde y azul varían entre 0 a 255.

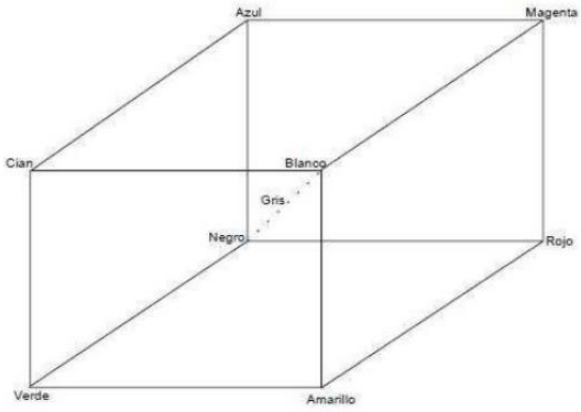

Imagen 27: Espacio de Colores

Fuente: Mundaca-vidarte (2016)

2.5.2. Técnicas básicas de procesamiento de imágenes

Estos se dividen en dos grandes grupos métodos de dominio espacial y métodos de dominio de frecuencia

2.5.3. Métodos de dominio espacial

Hace referencia a las modificaciones que se le hacen a la imagen con manipulación directa de los pixeles

2.5.3.1. Alteración global de brillo

Se obtiene sumándole una constante al valor de niveles de color/gris de cada imagen.

$$g(x, y) = f(x, y) + K$$

f(x,y) hace referencia a un pixel el cual tendrá un valor entre 0 y 255 donde 0 es negro y 255 blanco, K hace referencia al nivel de brillo que se desea aumentar o disminuir. Si el gris ya está en su valor máximo este se deja con el mismo nivel de gris.

a)

b)

c)

Imagen 28: Variación de brillo

Fuente: Toledo Muñoz (2005)

2.5.3.2. Binarizado

Los niveles de gris mayores a cierto número se convierten a 255 (blanco) y los menores se vuelven 0 (negro). Toma el mismo principio de la edición del brillo.

Imagen 29: Binarizado

Fuente: Toledo Muñoz (2005)

2.5.3.3. Cuantizado

Parecido al binarizado, solo que ahora definimos niveles de gris, los valores que estén dentro del rango se convierten a un valor previamente definido.

Imagen 30: Cuantizado

Fuente: Toledo Muñoz (2005)

2.5.3.4. Histograma

Nos permite observar una gráfica de los niveles de gris/verde/rojo/azul contra en número de pixeles.

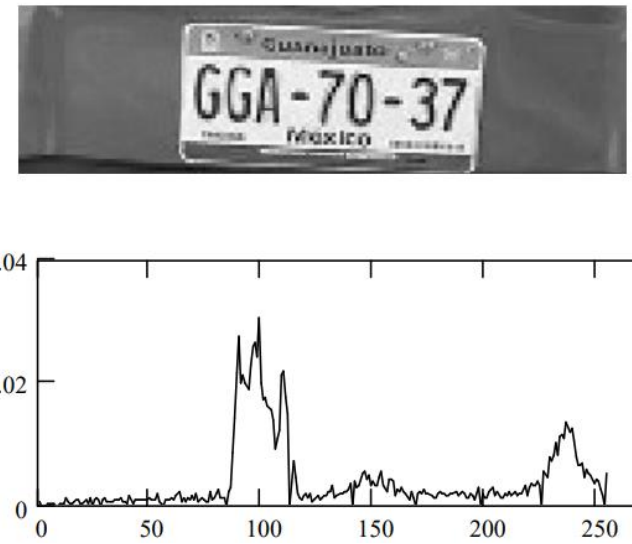

Imagen 31: Histograma

Fuente: Toledo Muñoz (2005)

2.5.3.5. Ecualizado

Modifica el contraste de una imagen, si la imagen es oscura aumenta el brillo y si es clara disminuye el brillo.

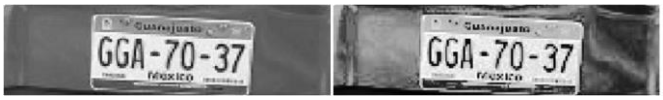

Imagen 32: Ecualizado

Fuente: Toledo Muñoz (2005)

2.5.3.6. Negativo de una imagen

Permite realizar una inversión de los niveles de color de una imagen, esto se obtiene mediante:

$$g(x, y) = 255 - f(x, y)$$

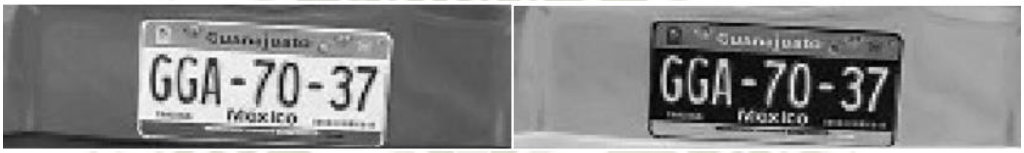

Imagen 33: Negativo

Fuente: Toledo Muñoz (2005)

De la formula podemos observar que toma un valor 255 (blanco) que es el valor mas alto y a este le reduce el valor que tiene cada pixel, siendo que si el valor del pixel es alto (claro) este tomara un valor pequeño (oscuro) y viceversa.

2.5.3.7. Pseudocolor

Permite agregar color a una imagen monocromática, se parece al cuantizado, pero en vez de usar niveles de gris se usan colores. Aquí el valor de cada pixel ya no será uno solo entre 0 y 255, sino tomara un arreglo de tres dimensiones RGB.

2.5.4. Métodos de dominio de la frecuencia

Hace referencia a modificaciones hechas mediante la transformada de Fourier de una imagen

2.5.4.1. Traslación

Estas ecuaciones realizan el efecto de trasladar una imagen en el espacio. Consiste en mover la imagen de un punto a otro.

$$f(x,y) * e^{[\frac{j2\pi(uox+voy)}{N}]} \quad \leftrightarrow F(u-uo, v-vo),$$

$$f(x-xo, y-yo) \leftrightarrow F(u,v) * e^{(-\frac{j2\pi(uxo+vyo)}{N})}$$

En la primera ecuación se ve que al multiplicar la función por el exponencial y luego realizar la transformada de Fourier de ese producto se mueve la imagen a los puntos uo y vo en un plano de frecuencia.

En la segunda ecuación muestra que al multiplicar F(u,v) por el valor de la exponencial y luego obtener la transformada inversa de ese producto se produce un desplazamiento al punto xo y yo en el plano de origen.

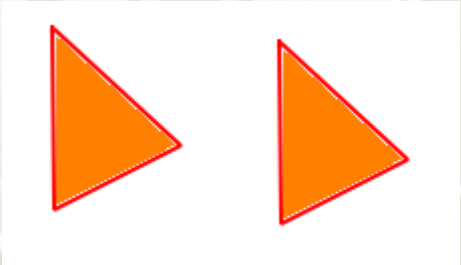

Imagen 34: Traslación

Fuente: Google sites

2.5.4.2. Reflexión

En términos de Fourier

$$F(u,v) = F(u+N,v) = F(u,v+N) = F(u+N,v+N)$$

Como se observa en la imagen 35 consiste en obtener simetría dado un eje en una posición

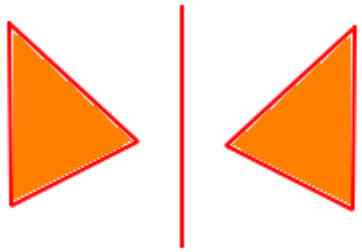

Imagen 35: Reflexión

Fuente: Google sites

De la ecuación se comprende que se le aumenta una contante a los valores de posición, obteniendo como resultado una simetría con respecto a un eje que puede ser verticar, horizontal o ambos.

2.5.4.3. Rotación

Consiste en trasformar una imagen girándola desde un punto dado. En términos de Fourier esto se explica cómo:

$$f(r, \theta + \theta o) \leftrightarrow F(\omega, \emptyset + \theta o)$$

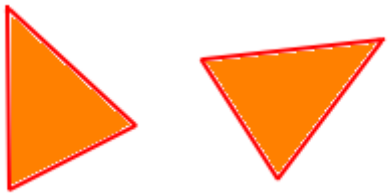

Imagen 36: Rotación

Fuente: Google sites

En esta ecuación la funcion f(x,y) se convierte en f(r,Θ) ya que se introducen coordenadas polares.

2.5.4.4. Distributividad y cambio de escala

Consiste en cambiar el tamaño de la imagen. En términos de Fourier.

$$F\{f_1(x,y) + f_2(x,y)\} = F\{f_1(x,y)\} + F\{f_2(x,y)\}$$

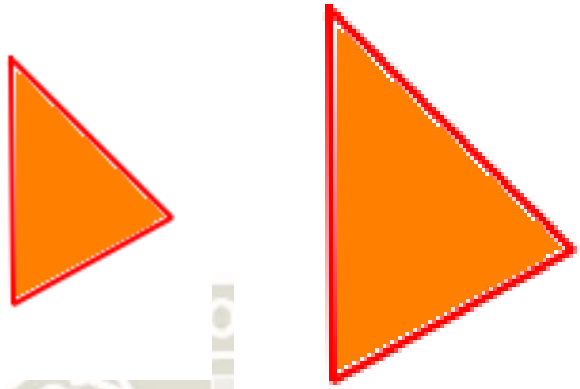

Imagen 37: Cambio de escala

En esta le aumentamos la escala agregando un valor esto debido a la propiedad de transformada de transformada de Fourier discreta.

2.5.4.5. Laplaciano

El laplaciano se define como

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2},$$

en términos de Fourier

$$F\{\nabla^2 f(x,y)\} \leftrightarrow -(2\pi)^2(u^2 + v^2)F(u,v),$$

este operador es útil para delimitar bordes

2.6. Software

2.6.1. Lenguaje Python

Python es un lenguaje de programación muy poderoso y flexible, a pesar de ello no deja de ser sencillo y fácil de aprender por cualquiera.

Python fue creado a inicios de los años noventa por Guido van Rossum en Holanda, este toma características de lenguajes predecesores, además de ser un software libre.

Python se desarrolla bajo una licencia OpenSource aprobada por OSI (Open Source Initiative) lo cual permite su uso de manera gratuita incluso de manera comercial.

Python cuenta con Python Package Index (PyPI) en español índice de paquetes de Python el cual contiene una gran variedad de módulos de terceros lo cual otorga infinitas posibilidades como:

- Desarrollo web
- Acceso a la base de datos
- GUIs de escritorio
- Científico numérico
- Desarrollo de software y juegos
- Otros

2.6.1.1. Ventajas

- Simple y rápido
- Elegante y flexible
- Productivo
- Ordenado y limpio

- Comunidad

2.6.1.2. Desventajas

- Curva de aprendizaje
- Hosting
- Librerías incluidas

2.6.1.3. Python en Windows

Python es muy sencillo de utilizar en cualquier plataforma y con cualquier sistema operativo sin importar si es Linux, MacOS o Windows. Para ello basta con instalar Python desde su página web para después tener un cuadro de dialogo donde programar.

Como si esto no fuera sencillo existen otras alternativas que ayudan a que la incorporación de Python sea más sencilla como por ejemplo Google colaboratory y Jupiter.

2.6.2. Google Colaboratory

Google Colaboratory es una alternativa para desarrollar algoritmos en Python sin importar el rendimiento o dispositivo desde el cual uno lo use.

Google Colab como también se le llama, se conecta a un servidor en la nube de Google el cual nos ofrecerá de manera gratuita 12GB de RAM y 107GB de almacenamiento.

Este a su vez tiene la ventaja que ya tiene instalado las librerías más usadas en el mundo de Python además de las que vienen por defecto.

2.6.3. Jupyter

Jupiter es muy parecido a Google Colab con la diferencia que este se ejecutara con los recursos de un computador.

La ventaja de tener un servidor local en vez de una en la nube como Google Colab, es los tiempos que se puede tener corriendo un código, o la posibilidad de continuar con el trabajo después de una pausa prolongada justo donde lo dejaste.

Para utilizar Jupiter Lab o Jupiter Notebook basta con instalar el programa Anaconda desde su página web este además de las librerías básicas de Python viene con unas extras, además de contar con otras plataformas de desarrollo como Spyder.

Otra ventaja de utilizar Anaconda es la posibilidad de tener diferentes entornos de programación.

2.6.4. Librerías

Mencionaremos algunas de las librerías más importantes.

2.6.4.1. Numpy

Numpy es una librería básica dentro de Python, esta nos proporciona lo equivalente a una estructura básica de array en MatLab. Es decir, Numpy nos ayuda con cualquier operación matemática que necesitemos. Una ventaja de Numpy es que fue escrito en lenguaje C por lo cual es muy rápido.

2.6.4.2. Pandas

Pandas es otra de las librerías que todo aquel que utiliza Python debe tener dentro de su repertorio este nos permite visualizar archivos .xls o .cvs entre otros, siendo .cvs la estructura más usada por científicos de datos para la creación de set de datos (dataset).

También permite la creación de “DataFrames” que son una forma sencilla de trabajar con datos

2.6.4.3. TensorFlow y Keras

TensorFlow es una librería desarrollada por Google esta absorbió a Keras que era una de sus principales competidoras dejando como principal competidor a Pytorch.

Estas librerías están enfocadas en el desarrollo de redes neuronales.

2.6.4.4. Pytorch

Pytorch es una librería de libre acceso desarrollado por Facebook's AI Research lab (FAIR). Este software es actualmente usado en Tesla Autopilot, Uber's Pyro entre otros.

Este es a la principal competencia de TensorFlow, enfocándose es el desarrollo de redes neuronales y computación de tensores como Numpy

2.6.4.5. Scikit-learn

También conocido como Sklearn es una librería en Python la cual nos permite trabajar con algoritmos de Aprendizaje de máquina, algoritmos de clasificación, regresión y análisis de grupos. Se pueden desarrollar arboles de decisión bosques aleatorios entre otros. Scikit-learn está diseñada para interoperar con Numpy.

2.6.4.6. Tesseract OCR

Tesseract es un sistema de reconocimiento de texto de libre acceso, disponible bajo la licencia Apache 2.0, desde 2006 este es financiado por Google.

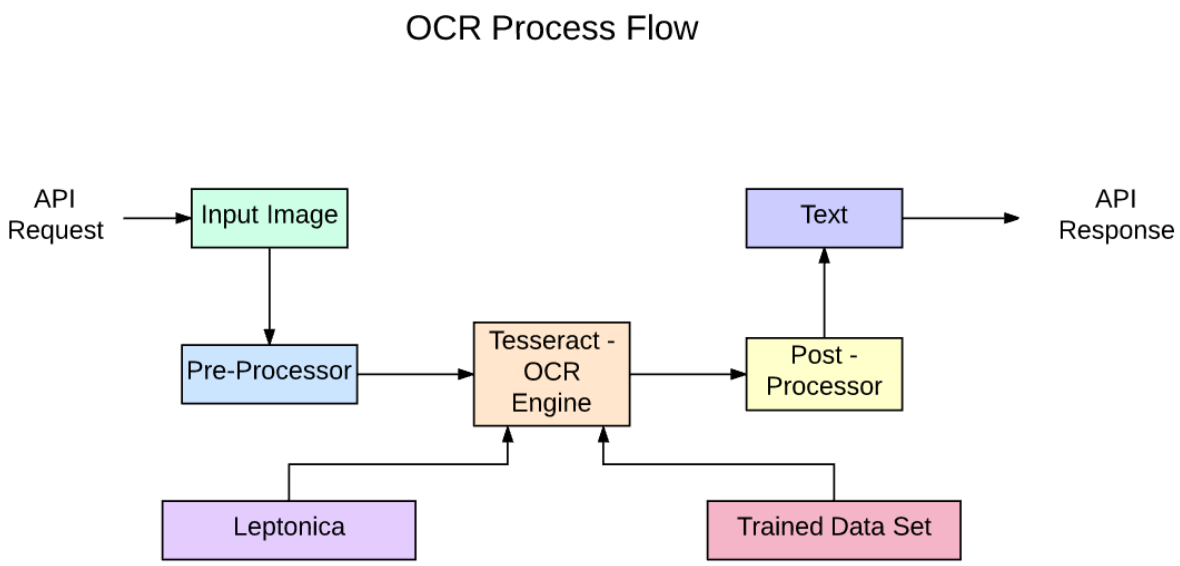

Imagen 38: Tesseract OCR

Fuente: Nanonets (2020)

Tesseract usa normalmente redes neuronales convolucionales, sin embargo también se puede utilizar redes neuronales recurrentes.

2.6.5. Yolo

Yolo viene del inglés You Only Look Once, Yolo es un framework de detección de objetos en tiempo real. Este nos proporciona una red neuronal convolucional ya estructurada para la detección de objetos en imágenes o videos. Hoy en día existen 4 versiones de Yolo, las primeras tres fueron creadas por Joseph Redmon una vez lanzada la tercera versión se informó que el trabajo seria continuado por Alexey Bochkovskiy, todas las versiones desarrolladas son con licencia gratuita por lo que se puede usar para cualquier motivo. En abril del 2020 salió una nueva versión YoloV5 creada por Glenn Jocher, dentro de este se encuentran 4 diferentes modelos de CNN, YOLOV5S, YOLOV5M, YOLOV5L, YOLOV5X donde el primero es el más pequeño e impreciso y el ultimo entrega una gran precisión

2.6.6. ResNet

ResNet es una CNN ya diseñada la cual nos da buenos resultados, existen tres tipos de ResNet, ResNet-50, ResNet-101, ResNet-152. ResNet nos puede servir al igual que Yolo para la detección de objetos.

2.6.7. Roboflow

Roboflow es una herramienta que ayuda a los desarrolladores con tareas de visión artificial. Este nos da las herramientas necesarias para construir modelos de visión computacional rápidos y precisos.

Una de las herramientas de Roboflow es “Roboflow Organize” esta nos permite agregar imágenes con sus anotaciones (coordenadas), para posteriormente poder navegar de manera sencilla a través de nuestra base de datos, y realizar un preprocesamiento a las imágenes.

2.6.8. GitHub

GitHub es una plataforma que nos permite tener un mejor control sobre diferentes proyectos, este nos da la capacidad de trabajar con colaboradores y/o compartir nuestro trabajo con otros desarrolladores. GitHub también se considera uno de los repositorios más grandes del mundo.

2.6.9. Python vs. MatLab

Python posee ciertas ventajas sobre MatLab:

- Python es un entorno de programación gratuito sin importar si es para industria o no.
- Python al ser OpenSource cuenta con librerías como TensorFlow que simplifican el trabajo

- Un código en Python se puede ejecutar desde una página web usando Google Colab mientras que MatLab debe de tener el programa instalado en el otro computador y este debe ser la misma versión.
- Algunas funciones en Python son más sencillas como por ejemplo la imagen.

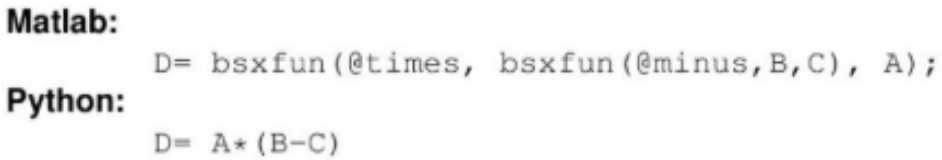

Imagen 39: MatLab vs. Python

Fuente: NGuerrero (2020)

3. PLANTEAMIENTO PRACTICO

3.1. ESQUEMA DE TRABAJO

Podemos dividir el procedimiento practico en tres, el primer paso es hacer un sistema de detección de la ubicación de las placas.

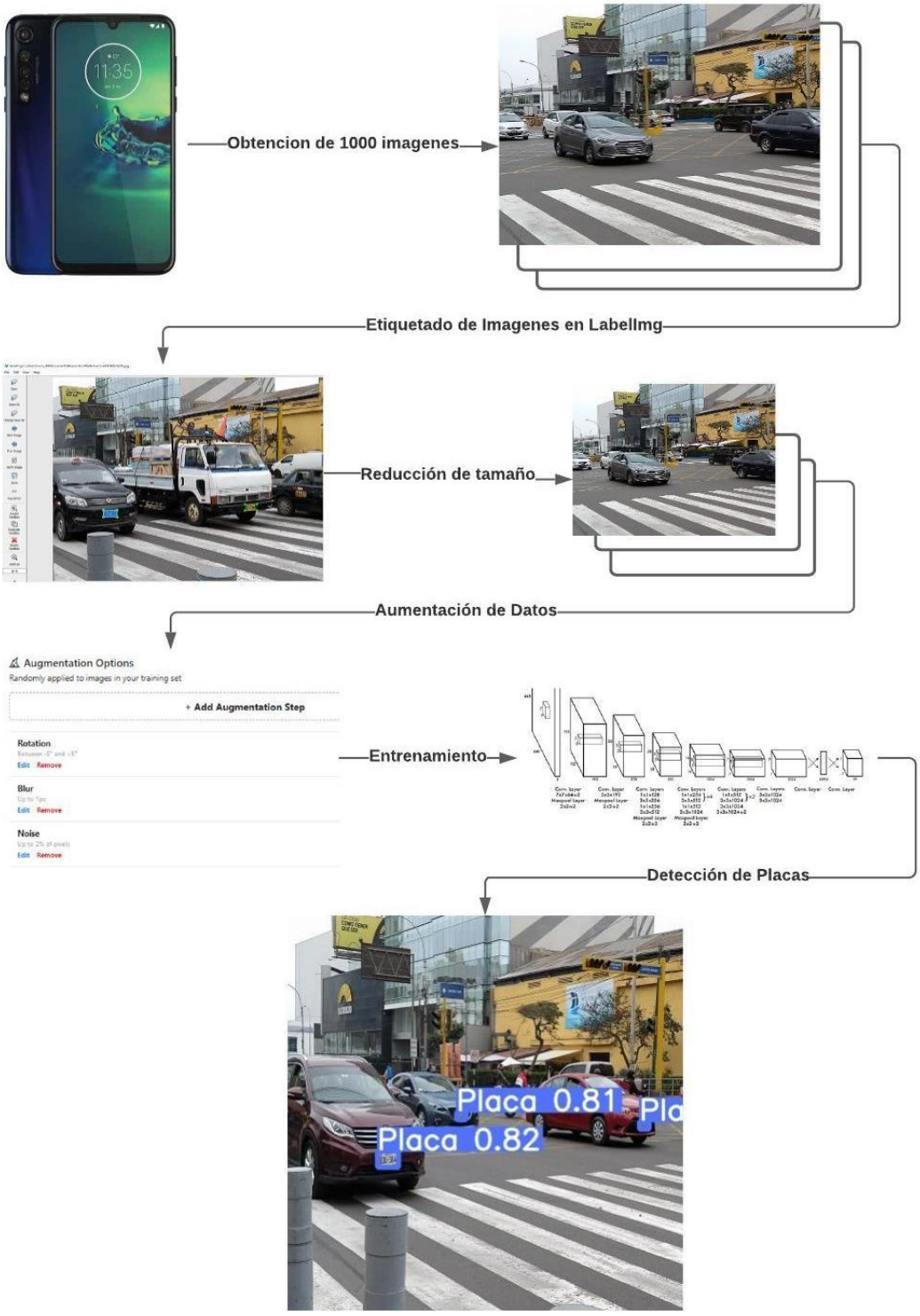

Imagen 40: Diagrama de flujo Placas

El segundo paso es reconocer el texto dentro de las placas detectadas.

Imagen 41: Diagrama de Flujo Texto

Finalmente debemos llevar esta información a una hoja de cálculo de Excel.

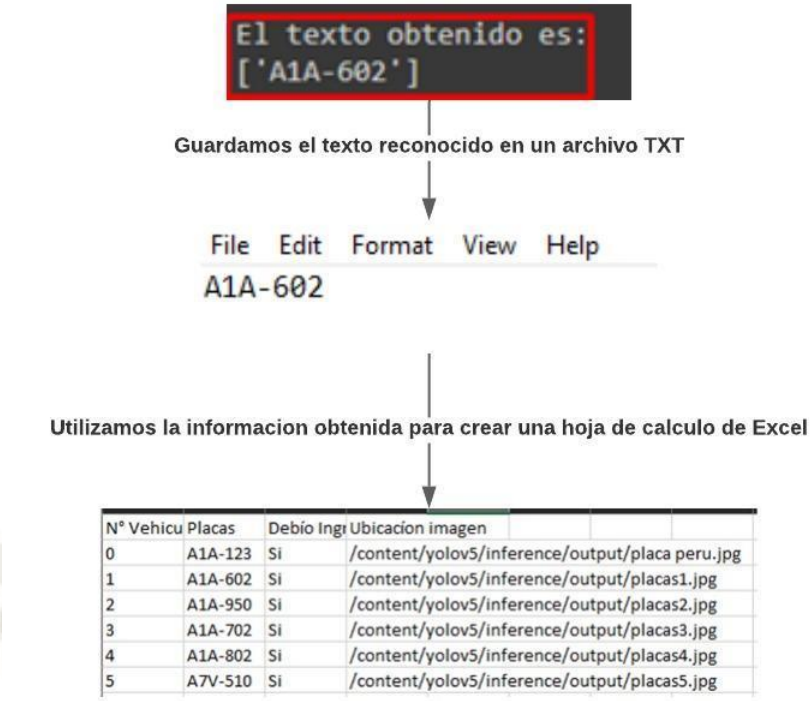

Imagen 42: Diagrama de Flujo Excel

3.2. ESTRATEGIAS DE RECOLECCIÓN DE DATOS

3.2.1. Organización

Las imágenes fueron tomadas en la avenida Javier Prado Este, esto debido a la gran afluencia de vehículos, lo cual facilito la velocidad con la cual se tomaron las fotografías, así como mayor cantidad de recursos para el entrenamiento. Las fotografías fueron tomadas con un teléfono celular Motorola G8 Plus el cual cuenta con una resolución de 48MP

3.2.2. Etiquetado de las imágenes

Para el etiquetado de las imágenes se usó LabelIMG el cual es recomendado por el mismo desarrollador de Yolo.

Imagen 43: IMGlabel

Este programa crea un archivo .txt por cada imagen, este archivo.txt contiene la información de las coordenadas del bounding box que se etiquetaron de manera manual.

Imagen 44: Archivo TXT con JPG correspondiente

```
<annotation>
        <folder>DATASET</folder>
        <filename>(1).jpg</filename>
        <path>C:\Users\bruno_000\Documents\Bruno\Tesis\Parte Practica\DATASET\(1).jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>4000</width>
                <height>3000</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>Placa</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>1594</xmin>
                        <ymin>1777</ymin>
                        <xmax>1725</xmax>
                        <ymax>1869</ymax>
                </bndbox>
        </object>
        <object>
                <name>Placa</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>163</xmin>
                        <ymin>1573</ymin>
                        <xmax>240</xmax>
                        <ymax>1615</ymax>
                </bndbox>
        </object>
</annotation>
```

Imagen 45: Archivo TXT creado por IMGLabel

3.2.3. Preprocesamiento

Para el preprocesamiento se utilizó Roboflow este nos permite cambiar el tamaño de las imágenes a uno solo, en este caso se usó el formato de (416 x 416), a su vez permite aumentar el tamaño de nuestro dataset haciendo pequeñas modificaciones a nuestras imágenes. Las modificaciones que se utilizaron son el efecto de inclinación con 5°, desenfoque y ruido.

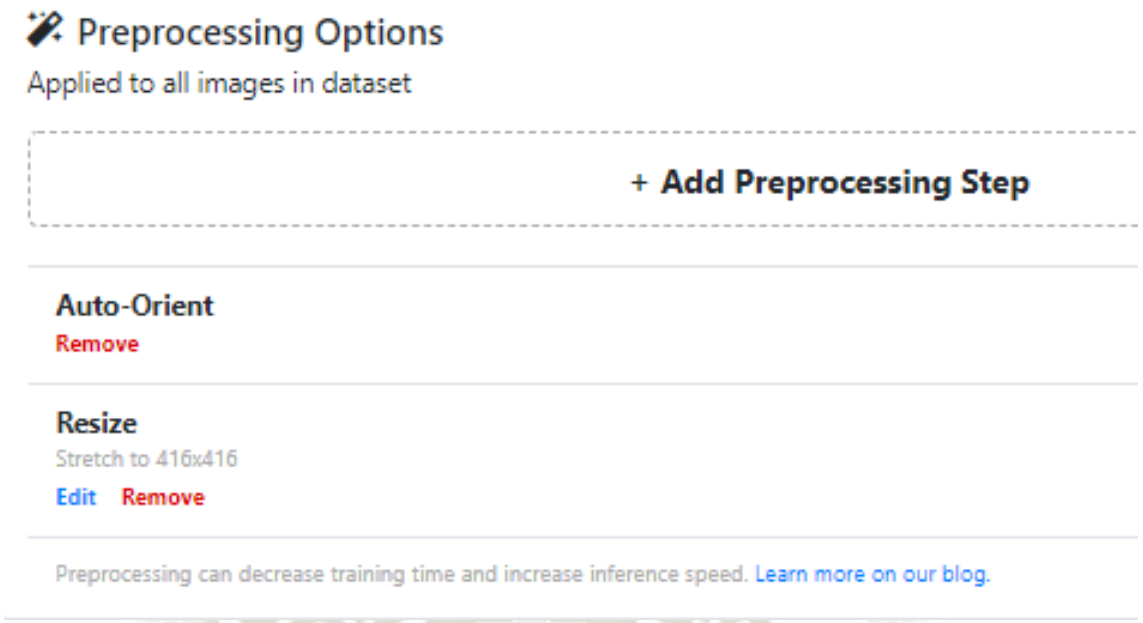

Imagen 46: Preprocesamiento Dataset

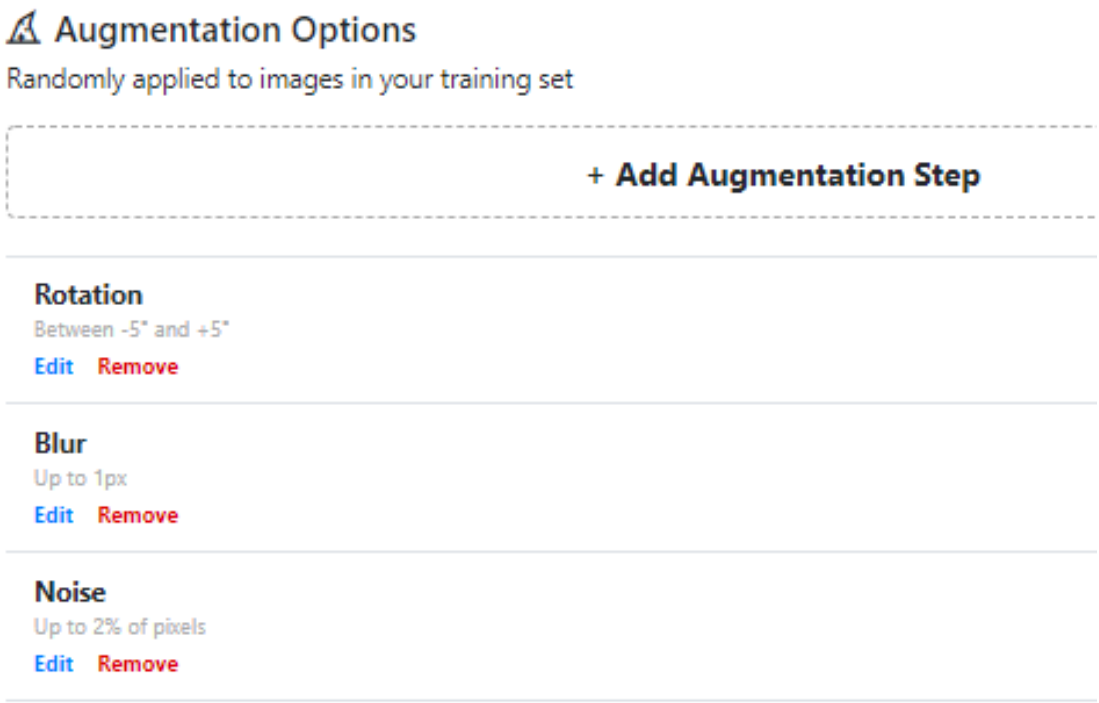

Imagen 47: Modificación para aumentar tamaño dataset

3.3. DETECCIÓN DE LA UBICACIÓN DE LAS PLACAS

Esta parte del algoritmo está inspirada por Roboflow una empresa la cual nos ayuda con el procesamiento de imágenes. Es por ello que nos da las herramientas necesarias para la detección

de imágenes con Yolo V5, cabe resaltar que su notebook está inspirado en el notebook de los creadores de Yolo V5.

3.3.1. Importamos Yolo V5

El primer paso en el notebook es clonar el repositorio de GitHub de Yolo V5, para ello basta con unas líneas de código las cuales crean una carpetas “YOLOV5”, dentro de las cual contamos con los modelos de redes neuronales que utilizaremos para detectar la ubicación de placas.

```
!git clone https://github.com/ultralytics/yolov5
!pip install -qr yolov5/requirements.txt
%cd yolov5
import torch
from IPython.display import Image, clear_output
from utils.google_utils import gdrive_download
print('Setup complete. Using torch %s %s' %
(torch.__version__, torch.cuda.get_device_properties(0) if
torch.cuda.is_available() else 'CPU'))
clear_output()
print('Setup complete. Using torch %s %s' %
(torch.__version__, torch.cuda.get_device_properties(0) if
torch.cuda.is_available() else 'CPU'))
```

Dentro de los archivos creados al clonar el GitHub, se encuentra el archivo “detect.py” el cual contiene todo el algoritmo necesario para usar las CNN y devolvernos los resultados.

Este archivo se modificará de manera que entre a línea 102 a 108 quede de la siguiente manera, siendo la única modificación la integración de “save_txt = True”

```
# Write results
for *xyxy, conf, cls in det:
save_txt = True
 if save_txt:  # Write to file
     xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
gn).view(-1).tolist()
     with open(txt_path + '.txt', 'a') as f:
          f.write(('%g ' * 5 + '\n') % (cls, *xywh))
```

Esto lo hacemos para poder obtener como salida archivos .txt donde se incluya las coordenadas de los bounding boxes detectados por la red neuronal.

3.3.2. Importamos las imágenes

Una vez contamos con el GitHub de Yolo V5 procedemos a importar nuestras imágenes ya preprocesadas en Roboflow.

```
%cd /content
!curl - "Link" > roboflow.zip; unzip roboflow.zip; rm
roboflow.zip
```

El enlace será dado por Roboflow en su página. Con este descargaremos un archivo .zip el cual será descomprimido, dentro de este archivo hay tres carpetas que contienen las imágenes de entrenamiento, prueba y validación.

3.3.3. Definimos modelo

Los modelos que encontramos dentro del repositorio clonado como se puede ver en la imagen 50 son:

- Yolov5l
- Yolov5m
- Yolov5s
- Yolov5x

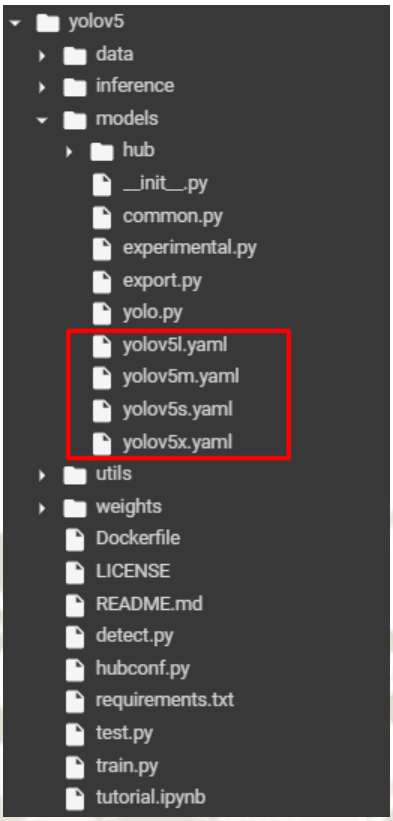

Imagen 48 CNN Importadas

Para ver el modelo que utilizaremos escribimos la siguiente línea de código donde se puede variar cual red neuronal queremos observar.

```
%cat /content/yolov5/models/yolov5s.yaml
```

Se obtiene como salida:

```
# parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
```

```
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  # 9
  ]
# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13
   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]],  # 17 (P3/8-small)
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, BottleneckCSP, [512, False]],  # 20 (P4/16-medium)
   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]],  # 23 (P5/32-large)
   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4,
P5)
  ]
```

Las redes neuronales convolucionales desarrolladas en Yolo cuentan con 24 capas convolucionales seguidas por dos capas totalmente conectadas.

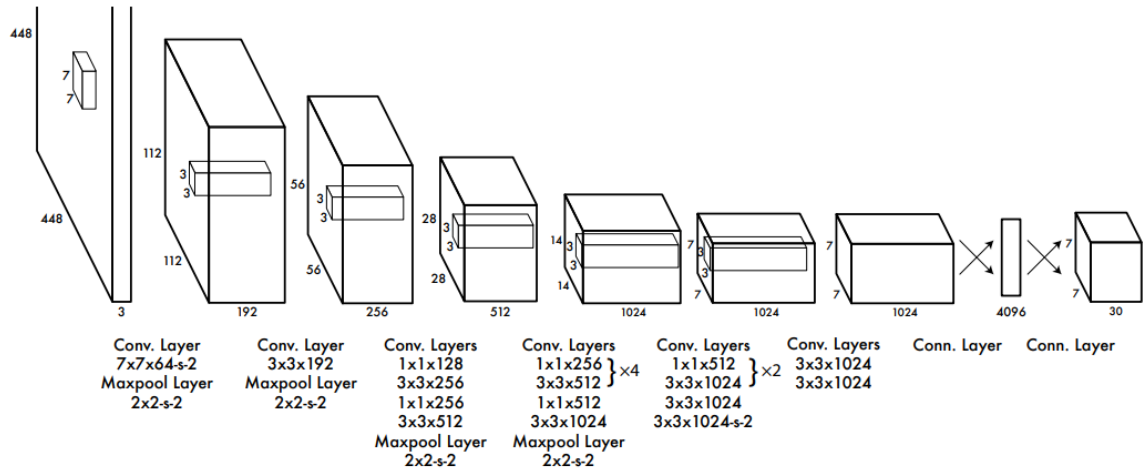

Imagen 49: Arquitectura Red neuronal convolucional

Fuente: Redmon, Divvala, Girshick, & Farhadi (2016)

3.3.4. Entrenamiento

Una vez definido nuestro modelo procedemos a entrenar con nuestras imágenes.

```
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 50 --data
'../data.yaml' --cfg ./models/yolov5s.yaml --weights '' --name
yolov5s_results  --cache
```

Este proceso puede demorar un tiempo. Una vez terminado el entrenamiento, este creara un archivo, el cual contiene almacenado los pesos que utilizaremos para detectar la ubicación de las placas.

Los parámetros utilizados fueron un batch size de 16 y un número de épocas igual a 50, batch size hace referencia al número de imágenes que toma del set de entrenamiento para entrenar la red neuronal y actualizar los valores de los pesos; el número de épocas hace referencia a cuantas veces se correrá todo el set de entrenamiento. La ventaja de usar un batch size es que podemos entrenar nuestra red neuronal más rápido.

Finalmente obtenemos un archivo best.pt el cual contiene la información de los pesos como se ve en la siguiente imagen (en esta imagen no se muestran todos los pesos).

```
00000000  50 4b 03 04 00 00 08 08 00 00 00 00 00 00 00 00
00000010  00 00 00 00 00 00 00 00 00 00 10 00 12 00 61 72
00000020  63 68 69 76 65 2f 64 61 74 61 2e 70 6b 6c 46 42
00000030  0e 00 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
00000040  80 02 7d 71 00 28 58 05 00 00 00 65 70 6f 63 68
00000050  71 01 4a ff ff ff ff 58 0c 00 00 00 62 65 73 74
00000060  5f 66 69 74 6e 65 73 73 71 02 63 6e 75 6d 70 79
00000070  2e 63 6f 72 65 2e 6d 75 6c 74 69 61 72 72 61 79
00000080  0a 5f 72 65 63 6f 6e 73 74 72 75 63 74 0a 71 03
00000090  63 6e 75 6d 70 79 0a 6e 64 61 72 72 61 79 0a 71
000000a0  04 4b 00 85 71 05 63 5f 63 6f 64 65 63 73 0a 65
000000b0  6e 63 6f 64 65 0a 71 06 58 01 00 00 00 62 71 07
000000c0  58 06 00 00 00 6c 61 74 69 6e 31 71 08 86 71 09
000000d0  52 71 0a 87 71 0b 52 71 0c 28 4b 01 4b 01 85 71
000000e0  0d 63 6e 75 6d 70 79 0a 64 74 79 70 65 0a 71 0e
000000f0  58 02 00 00 00 66 38 71 0f 4b 00 4b 01 87 71 10
00000100  52 71 11 28 4b 03 58 01 00 00 00 3c 71 12 4e 4e
00000110  4e 4a ff ff ff ff 4a ff ff ff ff 4b 00 74 71 13
00000120  62 89 68 06 58 0c 00 00 00 36 c2 88 53 40 c2 9e
00000130  c2 ac c3 a2 3f 71 14 68 08 86 71 15 52 71 16 74
00000140  71 17 62 58 10 00 00 00 74 72 61 69 6e 69 6e 67
00000150  5f 72 65 73 75 6c 74 73 71 18 4e 58 05 00 00 00
00000160  6d 6f 64 65 6c 71 19 63 6d 6f 64 65 6c 73 2e 79
00000170  6f 6c 6f 0a 4d 6f 64 65 6c 0a 71 1a 29 81 71 1b
00000180  7d 71 1c 28 58 08 00 00 00 74 72 61 69 6e 69 6e
00000190  67 71 1d 89 58 0b 00 00 00 5f 70 61 72 61 6d 65
000001a0  74 65 72 73 71 1e 63 63 6f 6c 6c 65 63 74 69 6f
000001b0  6e 73 0a 4f 72 64 65 72 65 64 44 69 63 74 0a 71
000001c0  1f 29 52 71 20 58 08 00 00 00 5f 62 75 66 66 65
000001d0  72 73 71 21 68 1f 29 52 71 22 58 1b 00 00 00 5f
000001e0  6e 6f 6e 5f 70 65 72 73 69 73 74 65 6e 74 5f 62
000001f0  75 66 66 65 72 73 5f 73 65 74 71 23 63 5f 5f 62
00000200  75 69 6c 74 69 6e 5f 5f 0a 73 65 74 0a 71 24 5d
00000210  71 25 85 71 26 52 71 27 58 0f 00 00 00 5f 62 61
00000220  63 6b 77 61 72 64 5f 68 6f 6f 6b 73 71 28 68 1f
00000230  29 52 71 29 58 0e 00 00 00 5f 66 6f 72 77 61 72
00000240  64 5f 68 6f 6f 6b 73 71 2a 68 1f 29 52 71 2b 58
00000250  12 00 00 00 5f 66 6f 72 77 61 72 64 5f 70 72 65
00000260  5f 68 6f 6f 6b 73 71 2c 68 1f 29 52 71 2d 58 11
00000270  00 00 00 5f 73 74 61 74 65 5f 64 69 63 74 5f 68
00000280  6f 6f 6b 73 71 2e 68 1f 29 52 71 2f 58 1a 00 00
00000290  00 5f 6c 6f 61 64 5f 73 74 61 74 65 5f 64 69 63
000002a0  74 5f 70 72 65 5f 68 6f 6f 6b 73 71 30 68 1f 29
000002b0  52 71 31 58 08 00 00 00 5f 6d 6f 64 75 6c 65 73
000002c0  71 32 68 1f 29 52 71 33 68 19 63 74 6f 72 63 68
000002d0  2e 6e 6e 2e 6d 6f 64 75 6c 65 73 2e 63 6f 6e 74
```

Imagen 50: Pesos entrenados

3.3.5. Exportamos pesos

Para poder aplicar nuestro código de manera diaria es necesario guardar el entrenamiento de nuestra red, si no hacemos esto tendríamos que entrenar nuestra red diariamente. Para ello aprovechamos las ventajas de almacenamiento en la nube de Google Drive.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

3.3.6. Cortar Bounding Box

Por último, con las coordenadas halladas recortamos las imágenes de manera que solo queden las partes donde se detectó la placa. Para ello utilizaremos los archivos txt que se crean al detectar un bouding box en la imagen. Estos archivos se obtienen gracias a la modificación que se hizo en el archivo “detect.py”.

```
img1 = sorted(glob.glob('/content/yolov5/inference/output/*.jpg'))
coor1 = sorted(glob.glob('/content/yolov5/inference/output/*.txt'))
b = 0
c = 0
lista1 = []
lista2 = []
for a in range(0,len(img1)):
  lis = img1[a].replace(".jpg","")
  lista1.append(lis)
for x in range(0,len(coor1)):
  lis = coor1[x].replace(".txt","")
  lista2.append(lis)
image_no = 0
for a in range(0,len(img1)):
  if lista1[a] == lista2[b]:
    img = img1[a]
    coor = coor1[b]
    myfile=open(coor,'r')
    lines=myfile.readlines()
    i=0
    while i < len(lines):
      Cord = lines[i].split(' ')
      print(Cord)
      x_min=(float(Cord[1])*416)-(float(Cord[3])*416/2) #416 2880
      x_max=(float(Cord[1])*416)+(float(Cord[3])*416/2)
      y_min=(float(Cord[2])*416)-(float(Cord[4])*416/2)
      y_max=(float(Cord[2])*416)+(float(Cord[4])*416/2)
      imageObject = Image.open(img)
      cropped = imageObject.crop((x_min,y_min,x_max,y_max)) #X1, Y1, X2
, Y2
      name = '/content/yolov5/inference/Corte/' + str(image_no) + '.jpg
'
      cropped.save(name, 'JPEG')
      image_no += 1
      plt.figure()
      plt.imshow(cropped)
      plt.show()
      i += 1
```

```
    b += 1
c += 1
```

En las lineas de código podemos ver que hacemos una conversión de las coordenadas.

(float(Cord[1])*416)-(float(Cord[3])*416/2)

Esto se debe a que los archivos txt devueltos al detectar una placa se encuentran escalados de 0 a 1. Podemos observar un ejemplo de las coordenadas devueltas en la imagen.

```
0 0.745192 0.543269 0.0288462 0.0288462
0 0.336538 0.467548 0.0192308 0.0168269
0 0.0384615 0.463942 0.0192308 0.0144231
```

Imagen 51: TXT devuelto por YOLO

La existencia de tres líneas se debe a que, en esa imagen en específico, detecto tres placas. El primer digito es 0 debido a que este indica el objeto detectado, en este estudio solo detecta placas por lo que siempre será 0. El segundo número y tercer número nos dan las coordenadas del centro del bounding box. Finalmente, el tercer y cuarto número nos dan las distancias hasta el borde en X y Y correspondientemente.

3.4. DETECCIÓN DE TEXTO EN IMÁGENES

Para la detección de texto en las imágenes utilizaremos la librería tesseract la cual proporciona la herramientas necesarias para hacer una detección rápida utilizando redes neuronales convolucionales.

3.4.1. Filtros y escalas

Experimentado con diferentes imágenes se determinó que una de las formas de mejorar la detección de texto es convertir la imagen a escala de grises y alargar la imagen. Para posteriormente realizar el reconocimiento de texto donde se eliminará todo carácter que sea diferente a una letra o número.

Otra técnica encontrada que funciona mejor con imágenes inclinadas o que no fueron tomadas de frente, es la de usar gaussiana que da un aspecto borroso seguido del efecto canny para detectar bordes.

Imagen 52: Placa original

Imagen 53: Placa escala de grises

Imagen 54: Placa inclinada original

Imagen 55: Placa inclinada escala de grises

Imagen 56: Placa inclinada Gaussiana

Imagen 57: Imagen inclinada escala de grises

Imagen 58: Resultado final placa inclinada

Una vez obtenida nuestra imagen procesada corremos la siguiente línea de código.

```
text=str(((pytesseract.image_to_string(imageOut,config='--psm 11')))
```

De esta manera la librería pytesseract utiliza redes neuronales convolucionales que ya fueron entrenadas para detectar el texto dentro de las placas. Finalmente, Como resultado nos

devolverá un archivo .txt por cada imagen dentro del cual se encuentra la información del número de placa.

El atributo psm hace referencia de como se desea segmentar el texto a detectar dentro de la imagen, existen 14 tipos de detección de texto en pytesseract los cuales se mostrarán en la siguiente imagen.

```
Page segmentation modes:
  0    Orientation and script detection (OSD) only.
  1    Automatic page segmentation with OSD.
  2    Automatic page segmentation, but no OSD, or OCR.
  3    Fully automatic page segmentation, but no OSD. (Default)
  4    Assume a single column of text of variable sizes.
  5    Assume a single uniform block of vertically aligned text.
  6    Assume a single uniform block of text.
  7    Treat the image as a single text line.
  8    Treat the image as a single word.
  9    Treat the image as a single word in a circle.
 10    Treat the image as a single character.
 11    Sparse text. Find as much text as possible in no particular order.
 12    Sparse text with OSD.
 13    Raw line. Treat the image as a single text line,
                        bypassing hacks that are Tesseract-specific.
```

Imagen 59: Segmentación según pytesseract

Se determino que el atributo 11 es el mejor para este caso ya que nos importa detectar todos los caracteres y al ser una placa este no tiene un significado más de ser un número de serie dado por un humano. Cabe resaltar que si fuera un texto donde importa el orden de las palabras pytesseract utilizaría redes neuronales recurrentes.

3.5. CREACIÓN DE ARCHIVO EN EXCEL

Para culminar con el trabajo se crean carpetas las cuales nos permitirán tener de manera organizada nuestros documentos. Se crean carpetas donde la primera lleva por nombre el año en el que nos encontramos, dentro de esta se crea otra carpeta que lleva por nombre el mes.

Finalmente se crea un archivo en Excel que lleva por nombre la fecha en la cual se corrió el código. Este archivo cuenta con la la fecha en la cual se tomó la fotografía, la cantidad de

vehículos que entraron al centro histórico, el N° de placa, la información si este debió entrar o no al centro y finalmente la dirección de la ubicación de la imagen. El archivo mostrado en la imagen se realizó el día 24 de septiembre del año 2020 que cae sábado, motivo por el cual se muestra que todos los vehículos pueden transitar.

N° Vehicu	Placas	Debío Ingr	Ubicacíon imagen
0	A1A-123	Si	/content/yolov5/inference/output/placa peru.jpg
1	A1A-602	Si	/content/yolov5/inference/output/placas1.jpg
2	A1A-950	Si	/content/yolov5/inference/output/placas2.jpg
3	A1A-702	Si	/content/yolov5/inference/output/placas3.jpg
4	A1A-802	Si	/content/yolov5/inference/output/placas4.jpg
5	A7V-510	Si	/content/yolov5/inference/output/placas5.jpg

Imagen 60: Archivo en Excel

4. RESULTADOS

4.1. DETECCIÓN DE PLACAS

4.1.1. Procesamiento y análisis de los datos

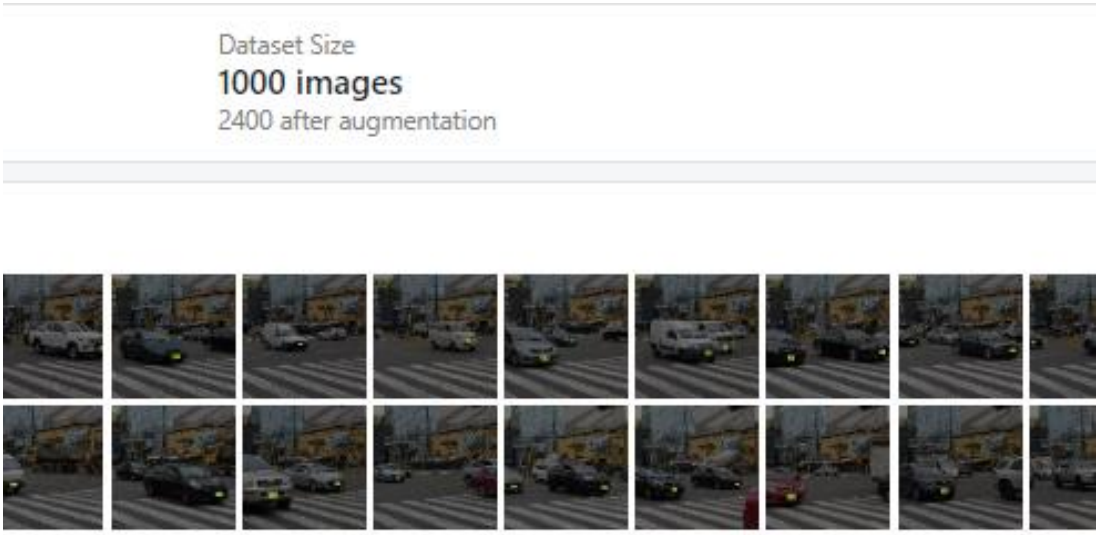

Imagen 61: Tamaño Dataset

Como se ve en la imagen 61 el tamaño original de nuestro set de datos es de 1000 imágenes, que después de añadir ediciones como inclinación, blur y ruido aumenta a 2400.

Imagen 62: División de nuestro Dataset

Nuestro set de datos se divide en 700 imágenes de entrenamiento 200 de validación y 100 de prueba. Las imágenes de entrenamiento se utilizarán para entrenar nuestra red obteniendo así los pesos, las imágenes de validación se utilizarán para saber qué tan preciso es nuestro modelo, y las imágenes de test se utilizarán para observar los resultados ya que estos nunca fueron utilizados para el entrenamiento.

4.1.2. Entrenamiento y validación

Se realizo el entrenamiento con los diferentes modelos dados por Yolo V5, se utilizaron los modelos YoloV5s, YoloV5m y YoloV5l. Para ver los resultados y poder compararlos se utilizó Tensorboard.

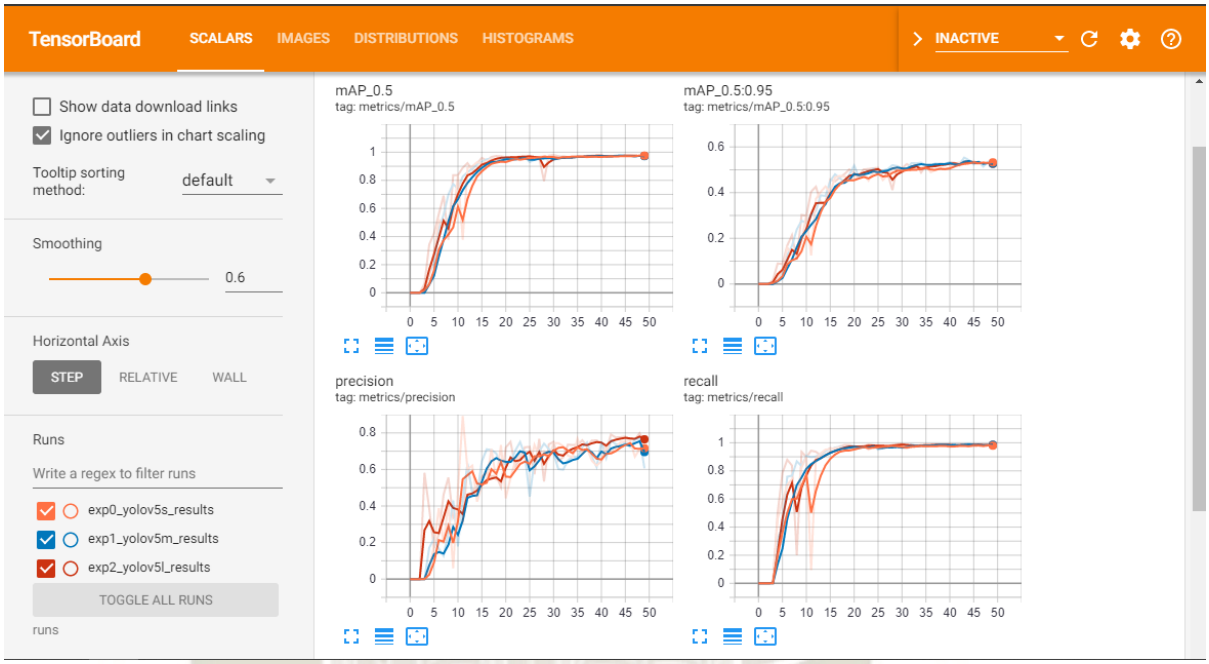

Imagen 63: Resultados Tensorboard

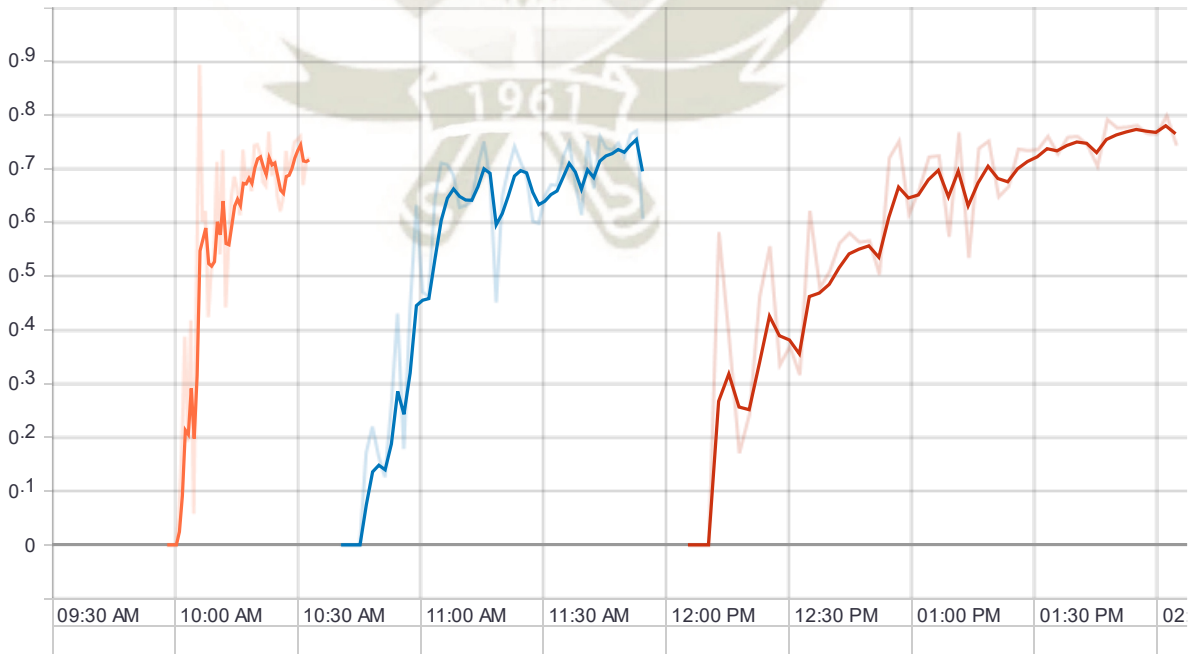

Imagen 64: Precisión Tensorboard

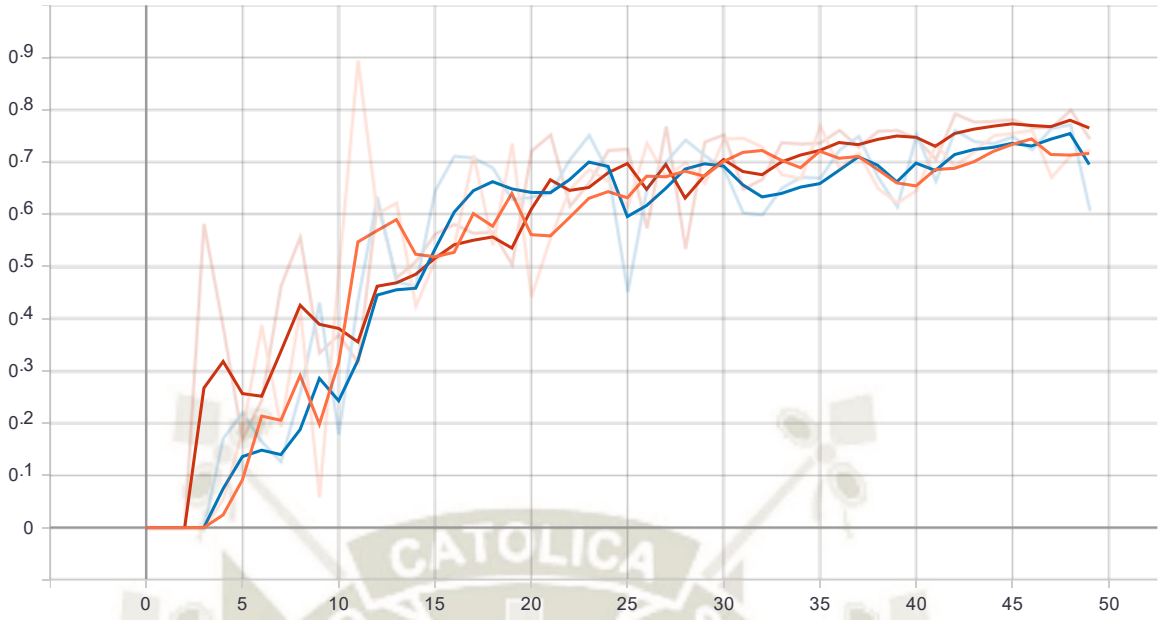

Imagen 65: Precisión Tensorboard

En las imágenes mostradas se puede observar las tablas comparativas de los tres modelos ejecutados obteniendo el mejor resultado con el modelo YoloV5l y el más bajo con el modelo YoloV5m.

La precisión de los modelos YoloV5l (rojo), YoloV5m (azul) y YoloV5l (naranja) como se observa en las imágenes, es de 0.7651, 0.6951 y 0.7163 respectivamente o en porcentaje 76.51%, 69.51% y 71.63%. Como podemos observar los resultados obtenidos en los tres modelos son buenos y con muy poca diferencia entre ellos.

Sin embargo, cabe resaltar los tiempos que tomo entrenar estos modelos, estos demoraron 2h con 2min y 31s en el modelo YoloV5l, 1h con 15min con 51s en el modelo YoloV5m y 36min y 6s en el modelo YoloV5s.

Imagen 66: Ejemplo Reconocimiento

4.2. DETECCION DE TEXTO

Como se observa en la imagen la detección de texto resulto eficiente utilizando una vez se eliminaron caracteres especiales

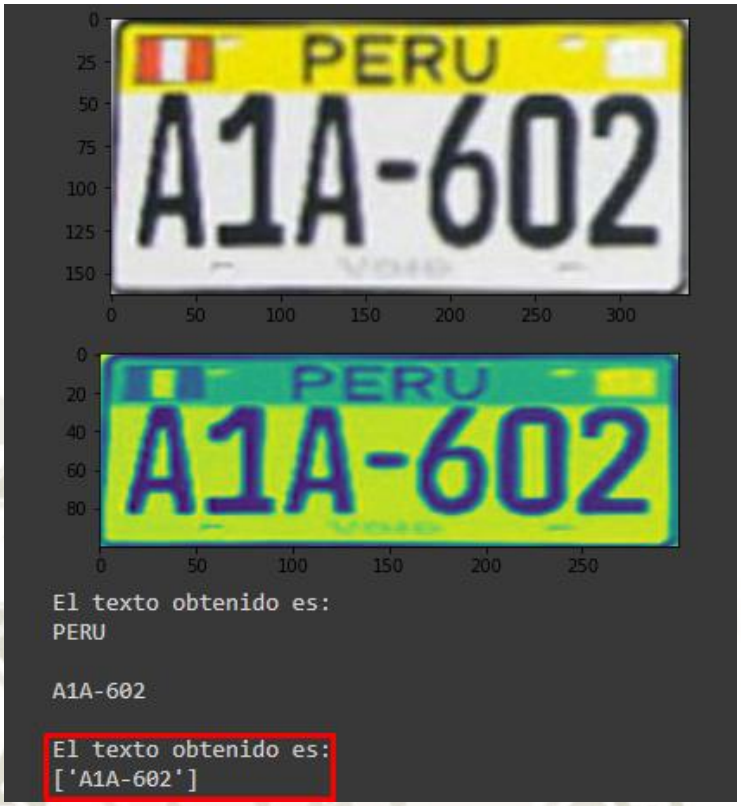

Imagen 67: Detección de texto

Imagen 68: Resultados texto

4.3. Archivo XLS

Se obtiene un archivo que tiene por nombre la fecha en que se corrió el programa, este se guardara dentro de una carpeta con el año y mes correspondiente. Esta imagen fue capturada el día 22 de octubre del 2020 que cae jueves motivo por el cual placas que finalicen en 6 y 7 no deben ingresar.

N° Vehicu	Placas	Debío Ingr	Ubicacíon imagen
0	A1A-123	Si	/content/yolov5/inference/output/placa peru.jpg
1	A1A-602	Si	/content/yolov5/inference/output/placas1.jpg
2	A1A-956	No	/content/yolov5/inference/output/placas2.jpg
3	A1A-702	Si	/content/yolov5/inference/output/placas3.jpg
4	A1A-802	Si	/content/yolov5/inference/output/placas4.jpg
5	A7V-510	Si	/content/yolov5/inference/output/placas5.jpg

Imagen 69: Resultado Excel

4.4. Alcance y limitaciones

Una vez finalizado el trabajo se puede observar que se obtienen buenos resultados con una precisión de 76.51% esto quiere decir de cada 100 placas se reconocerán entre 76 a 77 placas, motivo por el cual no se puede decir que el algoritmo sea perfecto.

También nosotros debemos tener en consideración que esta precisión se puede ver afectada por condiciones externas como por ejemplo el clima, el conjunto de datos que se utilizo se realizo bajo un clima soleado el cual es predominante en Arequipa sin embargo esto no lo hace ajeno a climas lluviosos o neblineros, es por ello que es necesario agregar al set de entrenamiento imágenes bajo diferentes climas y horas.

Para lograr el objetivo de controlar los vehículos que ingresan siempre se puede optar por un método convencional muy parecido al que se realiza hoy en día, revisar los datos (images) de manera manual este a su vez incluye la ventaja al método actual en el sentido que el personal policía no estará expuesto al extenuante sol característico de la ciudad de

Arequipa, así como también podrá revisar de manera exhaustiva de manera que detectará a todos los infractores.

A pesar de no contar con una precisión perfecta se puede decir que el sistema funciona dado que, si bien no se sancionara a todos los infractores es muy probable que si violo la ordenanza 1 vez lo vuelva a hacer y probablemente esa segunda vez que la infrinja se le reconocerá, también es muy probable que la mismo vehículo pase por otra cámara por lo cual su placa aparecerá no solo 1 imagen sino 2 lo cual aumentaría las probabilidades de atrapar al infractor.

CONCLUSIONES

- Finalmente se logró realizar el algoritmo en Python utilizando redes neuronales, el cual nos devuelve una tabla de Excel junto con las imágenes de prueba de los vehículos que infringieron la ordenanza municipal N° 927. Se anexa código en el CD
- Se estudiaron los diferentes tipos de redes neuronales, como las redes neuronales multicapa, redes neuronales convolucionales o redes neuronales recurrentes; además vimos que no solo se pueden dividir según su estructura, sino también se pueden dividir según su método de aprendizaje. Se profundizo en las redes neuronales convolucionales ya que estas fueron más utilizadas en este trabajo.
- Se creo una base de datos amplia usando como recurso básico la cámara de un celular Motorola G8 Plus el cual cuenta con una cámara de 48MP, Asu vez también se utilizó la plataforma de Roboflow para aumentar la información obteniendo una mejora de 1000 imágenes a un total de 2400, Estos se dividieron de manera que 70% fue usado para entrenamiento, 20% para prueba y 10% para validación. Estos se anexan en el CD
- Para el entrenamiento de nuestra red se seleccionaron parámetros como Batch size igual a 16 el cual permite seleccionar un conjunto de imágenes que serán entrenadas al mismo tiempo, también se utilizó un numero de épocas igual a 50 el cual indica cuantas veces se utilizara toda la data de entrenamiento.
- Se realizo un algoritmo en Python utilizando la arquitectura proporcionada por Yolo V5, específicamente se utilizaron los modelos S, M, y L los cuales cuentas con 24 capas de convolución, 6 de maxpoolin y dos capas totalmente conectadas.

- Se demostró las ventajas de utilizar leguaje Python esto debido a su gran versatilidad, cantidad librerías, y capacidad de utilizar plataformas como Google Colaboratory.

RECOMENDACIONES

- Realizar una comparación con el mismo set de datos entre una arquitectura de Yolo y una arquitectura ResNet.
- Encontrar más herramientas para facilitar el trabajo.
- Hacer un estudio para encontrar la mejor ubicación de cámaras, así como de los principales ingresos al centro histórico de Arequipa.
- Hacer uso de algoritmos neuro-evolutivos para reducir el tamaño del dataset.

Durán, 2014; Bravo & Villegas, 2017; Delgado, n.d.; Hinoz
Oheja, 2018; Shang, Liu, & Shang, 2020; Marín
Marín, 2018; Muñoz et al., 2019; Ng & Kiranfuros
; Roja Salazar Márquez, 2015; Tarkhov & Vasilye
& Moriakov, 2019; Theodoridis, 2020; Villalba, 2014)

REFERENCIA BIBLIOGRAFÍA

Álvarez Durán, M. A. (2014). Análisis, diseño e implementación de un sistema de control de ingreso de vehículos basado en visión artificial y reconocimiento de placas en el parqueadero de la Universidad Politécnica Salesiana - Sede Cuenca, 144.

AMPTech. (2019). Redes neuronales convolucionales CNN (Clasificación de imagenes).

Bravo, B., & Villegas, J. (2017). Diseño e Implementación de un Prototipo de Brazo Robótico (4gl) Teleoperado para Manipulación de Sustancias Tóxicas Asistido con Visión Artificial y Redes Neuronales para Laboratorios Farmacéuticos. *Universidad Católica de Santa María - UCSM*, 445.

Calvo, D. (2017). Tipos de redes neuronales. *July*.

Delgado, A. (n.d.). Software de Reconocimiento De Placas Vehiculares con Lógica Difusa en Matlab, 2–5.

DotCSV. (2018a). *¿Qué es una Red Neuronal? Parte 1 : La Neurona | DotCSV*.

DotCSV. (2018b). *¿Qué es una Red Neuronal? Parte 2 : La Red | DotCSV*.

Espinoza Vásquez, G. A. J. (2014). Sistema de reconocimiento de patrones en placas vehiculares para el acceso automático de visitas a un edificio. *Pontificia Universidad Católica Del Perú*.

Fathi, E., & Maleki Shoja, B. (2018). *Deep Neural Networks for Natural Language Processing*. *Handbook of Statistics* (1st ed., Vol. 38). Elsevier B.V. https://doi.org/10.1016/bs.host.2018.07.006

hewlett packard enterprice. (n.d.). HPE ARTIFICIAL INTELLIGENCE.

Jiao, L., Shang, R., Liu, F., & Zhang, W. (2020). *The models and structure of neural*

networks. *Brain and Nature-Inspired Learning Computation and Recognition*. https://doi.org/10.1016/b978-0-12-819795-0.00002-5

Marín Diazaraque, J. M. (2007). Introducción a las redes neuronales aplicadas. *Manual Data Mining*, 1–31.

Mosquera, A., & Martínez, J. (2018). Reconocimiento Optico de Caracteres en Placas Vehiculares haciendo uso de Redes Neuronales Convolucionales.

Mundaca-vidarte, G. (2016). DETECCIÓN DE CARACTERES DE PLACAS DE AUTOMÓVILES MEDIANTE TÉCNICAS DE VISIÓN ARTIFICIAL.

Muñoz, E. G., Cedeño, ;, O, F., Ruiz, ;, M, S., & Cruz, J. C. (2019). Aplicación de redes neuronales para predecir el éxito de la compra de deuda a una entidad financiera Application of neural networks to predict the success of the purchase of debt to a financial institution Contenido. *Espacios*, *40*, 6.

Nanonets. (2020). A comprehensive guide to OCR with Tesseract, OpenCV and Python. Retrieved from https://nanonets.com/blog/ocr-with-tesseract/

Ng, A., & Katanforoosh, K. (2000). Lecture notes Deep Learning, 1–30.

NGuerrero. (2020). Python vs. Matlab. Retrieved from https://www.programaenlinea.net/python-vs-matlab/

Nuñez, M. P. B. (2019). Redes Neuronales Convolucionales + Ejemplo usando Keras-Tensorflow.

Peñafiel, C., & Ávila, R. (2007). *Inteligencia Artificial*. *Inteligencia Artificial* (Vol. 2).

Ponce Cruz, P. (2010). *Inteligencia Artificial con aplicaciones a la ingeniería*. *Alfaomega, México*.

RAE Inteligencia Artificial. (n.d.).

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 779–788. https://doi.org/10.1109/CVPR.2016.91

Rojas, K. E. D. (2006). Localización y reconocimiento automático del número de la placa de un automovil.

Rusell, S., & Norvig, P. (2004). *Inteligencia Artificial. Un Enfoque Moderno*. *Inteligencia Artificial*. https://doi.org/10.4114/ia.v2i6.614

Salazar Márquez, M. B. (2015). Desarrollo de una algoritmo para la localización automática de placas vehiculares peruanas usando técnicas de procesamiento de imágenes. *Pontificia Universidad Católica Del Perú*.

Tarkhov, D., & Vasilyev, A. (2020). *Methods for the selection of parameters and structure of the neural network model*. *Semi-Empirical Neural Network Modeling and Digital Twins Development*. https://doi.org/10.1016/b978-0-12-815651-3.00003-1

Teuwen, J., & Moriakov, N. (2019). *Convolutional neural networks*. *Handbook of Medical Image Computing and Computer Assisted Intervention*. Elsevier Inc. https://doi.org/10.1016/B978-0-12-816176-0.00025-9

Theodoridis, S. (2020). *Neural Networks and Deep Learning*. *Machine Learning*. https://doi.org/10.1016/b978-0-12-818803-3.00030-1

Toledo Muñoz, L. E. (2005). Reconocimiento automático de matrículas de automóvil, 168.

Villalba, J. E. P. (2014). Reconocimiento de placas vehiculares mediante procesamiento de imágenes para optimizar el acceso a los parqueaderos de la UTA, Campus Huachi". *UNIVERSIDAD TÉCNICA DE AMBATO*.

ANEXOS

Anexo A Archivo "Tesis.iybnb":

```
from google.colab import drive

drive.mount('/content/gdrive')

#Importamos librerias

import glob

from IPython.display import Image, display

import yaml

import os

from datetime import date

from datetime import datetime


from PIL import Image

import sys

import cv2

from matplotlib import pyplot as plt

import numpy as np

import openpyxl

!git clone https://github.com/ultralytics/yolov5  # clone repo

!pip install -qr yolov5/requirements.txt  # install dependencies (ignore errors)

%cd yolov5


import torch

from IPython.display import Image, clear_output  # to display images
```

```
from utils.google_utils import gdrive_download  # to download models/datasets


print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
!curl -L "https://app.roboflow.com/ds/FybbOq9TKl?key=CXa9UJbgZb" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip #416*416
# define number of classes based on YAML
with open("data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])
%cat /content/yolov5/models/yolov5l.yaml
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 50 --data '../data.yaml' --cfg ./models/yolov5l.yaml --weights '' --name yolov5l_results  --cache
# Start tensorboard
%load_ext tensorboard
%tensorboard --logdir runs
Image(filename='/content/yolov5/runs/exp2_yolov5l_results/results.png', width=1000)
# view results.png
directorio = "/content/yolov5/inference/Corte"
try:
  os.mkdir(directorio)
```

```
except OSError:
    print("")
else:
    print("")
img1 = sorted(glob.glob('/content/yolov5/inference/output/*.jpg'))
coor1 = sorted(glob.glob('/content/yolov5/inference/output/*.txt'))
b = 0
c = 0
lista1 = []
lista2 = []
for a in range(0,len(img1)):
  lis = img1[a].replace(".jpg","")
  lista1.append(lis)
for x in range(0,len(coor1)):
  lis = coor1[x].replace(".txt","")
  lista2.append(lis)
image_no = 0
for a in range(0,len(img1)):
  if lista1[a] == lista2[b]:
    img = img1[a]
    coor = coor1[b]
    myfile=open(coor,'r')
    lines=myfile.readlines()
    i=0
    while i < len(lines):
```

```
        Cord = lines[i].split(' ')
        print(Cord)
        x_min=(float(Cord[1])*416)-(float(Cord[3])*416/2) #416 2880
        x_max=(float(Cord[1])*416)+(float(Cord[3])*416/2)
        y_min=(float(Cord[2])*416)-(float(Cord[4])*416/2)
        y_max=(float(Cord[2])*416)+(float(Cord[4])*416/2)
        imageObject = Image.open(img)
        cropped = imageObject.crop((x_min,y_min,x_max,y_max)) #X1, Y1, X2, Y2
        name = '/content/yolov5/inference/Corte/' + str(image_no) + '.jpg'
        cropped.save(name, 'JPEG')
        image_no += 1
        plt.figure()
        plt.imshow(cropped)
        plt.show()
        i += 1
      b += 1
    c += 1
!sudo apt install tesseract-ocr
!pip install pytesseract
# Import libraries
from PIL import Image
import pytesseract
import sys
import os
import cv2
```

```
from matplotlib import pyplot as plt
import numpy as np
import glob
import openpyxl
filename1 = sorted(glob.glob('/content/yolov5/inference/Corte/*.jpg'))
%cd /content/yolov5/inference/Corte/
a = 0
image_no = 0
for a in range(0,len(filename1)):
  filename = filename1[a]
  outfile = 'placa'+ str(a) +'.txt'
  f = open(outfile,"a")
  img = cv2.imread(filename)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  plt.imshow(img)
  plt.show()
  # Escalando una imagen
  gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  imageOut = cv2.resize(gris,(300,100), interpolation=cv2.INTER_CUBIC)
  # Recognize the text as string in image using pytesserct
  text = str(((pytesseract.image_to_string(imageOut, config='--psm 11'))))
  text = text.replace('-\n', '')
  if text.isspace() == True:
    gaussiana = cv2.blur(gris, (3,3))
    canny = cv2.Canny(gaussiana, 50, 150)
```

```
    #Creando contornos
    (contornos,_) = cv2.findContours(canny.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(img,contornos,-1,(0,0,255), 2)
    imageOut = cv2.resize(img,(300,100), interpolation=cv2.INTER_CUBIC)
    text = str(((pytesseract.image_to_string(imageOut, config='--psm 11'))))
    text = text.replace('-\n', '')
  plt.imshow(imageOut)
  plt.show()
  # Finally, write the processed text to the file.
  f.write(text)
  print('El texto obtenido es:')
  print(text)
  # Close the file after writing all the text.
  f.close()
  f=open(outfile,'r')
  lines=f.readlines()
  f.close()
  output = []
  for line in lines:
    line = line.replace(' ','')
    line = line.replace('\n','')
    line = line.replace("","")
    line = line.replace("""","")
    line = line.replace(')','')
```

```
    line = line.replace('(',")
    line = line.replace('/',")
    line = line.replace('\\',")
    if len(line) == 7:
        output.append(line)
  f = open(outfile, 'w')
  f.writelines(output)
  print('El texto obtenido es:')
  print(output)
  f.close()
%cd /content/
wb = openpyxl.Workbook()
hoja = wb.active
today = date.today()
formato1 = "%d-%m-%Y"
Fecha = today.strftime(formato1)
hoja.title = str(Fecha)
hoja["A1"] = "N° Vehiculo"
hoja["B1"] = "Placas"
hoja["C1"] = "Debío Ingresar"
hoja["D1"] = "Ubicacíon imagen"
a = 0
i=2
ubic = sorted(glob.glob('/content/yolov5/inference/output/*.jpg'))
for txt in sorted(glob.glob('/content/yolov5/inference/Corte/*.txt')):
```

```
f = open(txt,"r")

dia_semana = datetime.weekday(today)

placa = f.readlines()

f.close()

hoja["A" + str(i)] = str(a)

hoja["B" + str(i)] = placa[0]

b2 = hoja["B" + str(i)]

hoja["D" + str(i)] = ubic[a]

hoja["C" + str(i)] = "Si"

#Lunes

if dia_semana == 0 and b2.value[6] == "0" or  b2.value[6] == "1" :

  hoja["C" + str(i)] = "No"

#Martes

if dia_semana == 1 :

  if b2.value[6] == "2" or  b2.value[6] == "3" :

    hoja["C" + str(i)] = "No"

#Miercoles

if dia_semana == 2 and b2.value[6] == "4" or  b2.value[6] == "5" :

  hoja["C" + str(i)] = "No"

#Jueves

if dia_semana == 3 and b2.value[6] == "6" or  b2.value[6] == "7" :

  hoja["C" + str(i)] = "No"

#Viernes

if dia_semana == 4 and b2.value[6] == "8" or  b2.value[6] == "9" :

  hoja["C" + str(i)] = "No"
```

```
  #Sabado
  if dia_semana == "5":
    hoja["C" + str(i)] = "Si"
  #Domingo
  if dia_semana == "6":
    hoja["C" + str(i)] = "Si"
  i += 1
  a += 1
directorio = "/content/" + today.strftime("%Y")
try:
  os.mkdir(directorio)
except OSError:
    print("")
else:
    print("")
directorio = "/content/" + today.strftime("%Y") + '/' + today.strftime("%B")
try:
  os.mkdir(directorio)
except OSError:
    print("")
else:
    print("")
wb.save('/content/'+ today.strftime("%Y") +'/' + today.strftime("%B") + '/' + Fecha + '.xl
sx')
```

Anexo B Set de entrenamiento y validación (dataset):

Se encuentra dentro de CD