

ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

**IMPLEMENTACIÓN DE REDES NEURONALES PARA
IDENTIFICAR ELEMENTOS GRAFICOS**

**IMPLEMENTACIÓN DE PROTOTIPO DE RECONOCIMIENTO DE
PLACAS DE CARROS MEDIANTE REDES NEURONALES
CONVOLUCIONALES MEDIANTE LIBRERÍAS OPEN SOURCE**

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO
REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR
EN REDES Y TELECOMUNICACIONES**

BRYAN OSWALDO YANZA MAILA

DIRECTOR: ING. FERNANDO VINICIO BECERRA CAMACHO, MSC.

DMQ, marzo 2023

CERTIFICACIONES

Yo, Bryan Oswaldo Yanza Maila declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.



Bryan Oswaldo Yanza Maila

bryan.yanza@epn.edu.ec

bryan99_yanza@outlook.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Bryan Oswaldo Yanza Maila, bajo mi supervisión.



Fernando Vinicio Becerra Camacho

DIRECTOR

fernando.becerrac@epn.edu.ec

DEDICATORIA

El presente proyecto de titulación va dedicado a mis padres, porque a lo largo de mi vida han sido mi inspiración y mi ejemplo para ser una persona respetuosa, trabajadora, humilde y colaborativa con los demás.

También agradezco a mis hermanos por verme como su ejemplo a seguir puesto que su visión de mi ha hecho que me esfuerce todos los días.

Bryan Oswaldo Yanza Maila

AGRADECIMIENTO

Agradezco a mis padres por siempre estar a mi lado en todo momento feliz y triste de mi vida, gracias por brindarnos a mí y mis hermanos todo lo necesario para que tengamos una vida sin necesidades y sepan que todo su esfuerzo ha hecho que seamos personas de bien.

También agradezco a todos mis maestros que he tenido a lo largo de la carrera porque supieron transmitir sus conocimientos teóricos y morales para que nos formemos como profesionales con vastos conocimientos dentro de nuestra área y también con valores y ética para que podamos salir al mundo adulto y profesional.

Finalmente doy gracias a mi director Ing. Fernando Becerra por darme la oportunidad de realizar mi trabajo de titulación bajo su supervisión y darme tabla.

Bryan Oswaldo Yanza Maila

ÍNDICE DE CONTENIDOS

CERTIFICACIONES.....	I
DECLARACIÓN DE AUTORÍA	I
DEDICATORIA	II
AGRADECIMIENTO.....	III
ÍNDICE DE CONTENIDOS	IV
RESUMEN.....	VI
ABSTRACT.....	VII
1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO.....	1
1.1 Objetivo general.....	1
1.2 Objetivos específicos.....	1
1.3 Alcance	1
1.4 Marco Teórico	1
Deep Learnig.....	1
Redes convolucionales	4
2 METODOLOGÍA.....	4
3 RESULTADOS	5
3.1 Analizar redes neuronales y librerías open source.....	6
3.2 Diseñar la solución de cada escenario mediante redes convolucionales	9
3.3 Implementar las soluciones mediante redes neuronales para el despliegue de los diferentes servicios	11
3.4 Verificar el funcionamiento de cada servicio implementado con redes neuronales	22
4 CONCLUSIONES.....	27
5 RECOMENDACIONES.....	28
6 REFERENCIAS BIBLIOGRÁFICAS	29
7 ANEXOS.....	31
ANEXO I: Certificado de Originalidad	i

ANEXO II: Enlaces ii
ANEXO III: Códigos Fuente iii

RESUMEN

El presente proyecto tiene como objetivo la aplicación de redes neuronales convolucionales para el estudio de imágenes mediante el uso de librerías *open source*. Dentro de este proyecto se realizó el estudio del término *Deep Learning* y de manera específica de redes neuronales convolucionales. El objetivo de la extracción es llegar a reconocer características específicas de las imágenes ingresadas en el sistema.

Después de lograr detectar la matrícula vehicular de cualquier imagen se realiza el reconocimiento óptico de caracteres, el cual permite detectar y extraer el texto de la imagen detectada. Finalmente, la información conseguida se almacena de forma separada, la imagen va directo a una carpeta mientras que el texto a una base de datos.

La primera sección expone la descripción del proyecto y teoría consultada. Dentro de esta se estudió los términos *Deep Learning*, redes convolucionales, librerías *open source* y programas que permitieron la realización del proyecto.

La segunda sección abarca la metodología usada para su desarrollo, esta fue dividida en etapas que permitió una mejor y ordenada implementación del modelo de detección.

La tercera sección corresponde a los resultados obtenidos. La comprobación del proyecto consistió en la detección de la matrícula y su texto a partir de una imagen. Estos registros fueron almacenados en una carpeta (imágenes) y en una base de datos.

La última sección abarca las conclusiones a las que se llegó después de realizar el proyecto, además de las recomendaciones que pueden realizar las personas que realicen este proyecto en un futuro.

PALABRAS CLAVE: *Deep Learning*, redes convolucionales, librerías *open source*, detección, matrícula, almacenamiento, base de datos.

ABSTRACT

The objective of this project is the application of convolutional neural networks for the study of images through the use of open source libraries. Within this project, the study of the term Deep Learning and specifically convolutional neural networks was carried out. The objective of the extraction is to recognize specific characteristics of the images entered the system.

After detecting the vehicle license plate of any image, optical character recognition is performed, which allows detecting and extracting the text of the detected image. Finally, the information obtained is stored separately, the image goes directly to a folder and the text to a database.

The first section exposes the description of the project and the theory consulted. Within this, the terms Deep Learning, convolutional networks, open source libraries and programs that allowed the realization of the project were studied.

The second section covers the methodology used for its development, this was divided into stages that allowed a better and orderly implementation of the detection model.

The third section corresponds to the results obtained, the verification of the project consisted in the detection of the license plate and its text from an image. These records were stored in a folder (images) and in a database.

The last section covers the conclusions that were reached after carrying out the project, in addition to the recommendations that people who carry out this project in the future can make.

KEYWORDS: *Deep Learning, convolutional networks, open source libraries, detection, plate, storage, database.*

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

En el presente proyecto de titulación se desea implementar una serie de servicios mediante el manejo de redes neuronales que son actualmente tendencia. Para realizar este acometido se pretende utilizar herramientas y librerías de software libre para el manejo de redes neuronales como también para el almacenamiento de la información.

1.1 Objetivo general

Implementar redes neuronales para identificar elementos gráficos.

1.2 Objetivos específicos

1. Analizar redes neuronales y librerías *open source*.
2. Diseñar la solución de cada escenario mediante redes convolucionales.
3. Implementar las soluciones mediante redes neuronales para el despliegue de los diferentes servicios.
4. Verificar el funcionamiento de cada servicio implementado con redes neuronales.

1.3 Alcance

El presente proyecto permite a los estudiantes el manejo de redes neuronales para poder identificar elementos en una imagen esto se lo realizará con redes convolucionales utilizando librerías de tipo *open source* mediante funciones de activación las cuales serán analizadas por el estudiante. Además, este proyecto es macro el cual contiene un total de dos componentes de titulación los cuales tienen su propia línea de despliegue que en la actualidad se utiliza muy a menudo.

1.4 Marco Teórico

Deep Learnig

El termino *Deep Learning* o conocido en el español como aprendizaje profundo corresponde a toda técnica de aprendizaje automático que emplea redes neuronales, este concepto está dentro del campo de estudio del *Machine Learning* y este a su vez dentro de la inteligencia artificial [1]. Esta relación se la puede observar de mejor manera en la **Figura 1.1**.

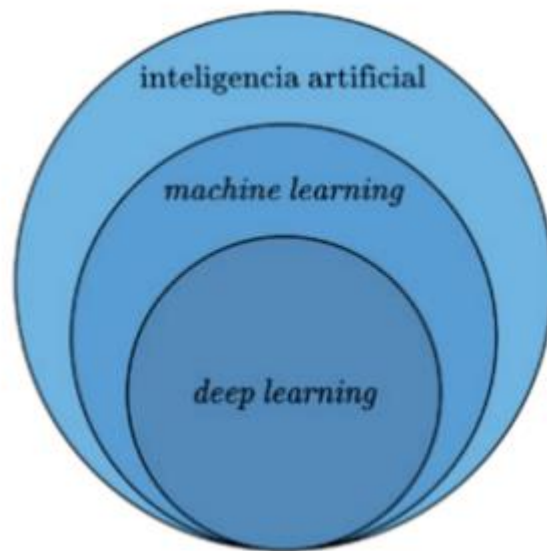


Figura 1.1 Relación entre IA, *Machine* y *Deep Learning* [1]

Deep Learning emula todas las características del aprendizaje utilizado por el ser humano para conseguir algún tipo de nuevo conocimiento [2]. El objetivo del *Deep Learning* es aprender por sí mismo y llegar a entender las relaciones que existe entre variables de entrada sin la necesidad de que un humano intervenga [1].

El termino *Deep Learning* hace referencia a la agrupación de varias técnicas de *Machine Learning* basados en modelos de redes neuronales. Las técnicas usadas en *Machine Learning* consisten en el manejo de un modelo predefinido que trabaja con un conjunto de características (*features*) obtenidas de los datos originales para solucionar un problema y devolver *un* resultado. Por el contrario, *Deep Learning* usa un modelo que extrae las características (*features*) más importantes de los datos originales y resuelve el problema.

La extracción de las características de los datos en *Deep Learning* se las hace con un nivel de jerarquía, es decir se extrae de menor a mayor complejidad. Las características de alto nivel son seleccionadas por el sistema para obtener una salida deseada [3].

Al utilizar *Deep Learning* en imágenes, en primer lugar, se extraerá una característica de bajo nivel o baja complejidad como los colores o bordes. Al profundizar en las características de la jerarquía llegará a niveles más altos en donde se podrá obtener características complejas de la imagen como: animales, objetos o un rostro, como se mencionó, estas características complejas serán usadas para crear la salida del modelo a ser usado [3] [2]. Una mejor representación de las diferencias entre las técnicas usadas por *Machine Learning* y *Deep Learning* se puede observar en la **Figura 1.2**.

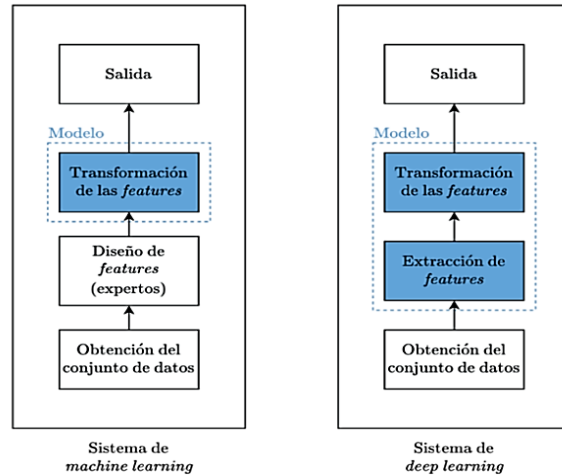


Figura 1.2 Comparación de las técnicas usadas en *Machine* y *Deep Learning* [1]

El modelo *Deep Learning* tiene dos etapas que a continuación se detalla: La primera etapa es llamada etapa de extracción de características, dentro de esta se crea la jerarquía que van a seguir las peculiaridades tomando como punto de partida los datos iniciales u originales. La segunda etapa se llama etapa de transformación de características, aquí se seleccionan los distintivos provenientes de niveles altos y se las aplica una transformación para que la salida sea la esperada por el sistema [3]. En la **Figura 1.3** se puede observar cómo al subir el nivel se obtiene características más específicas.

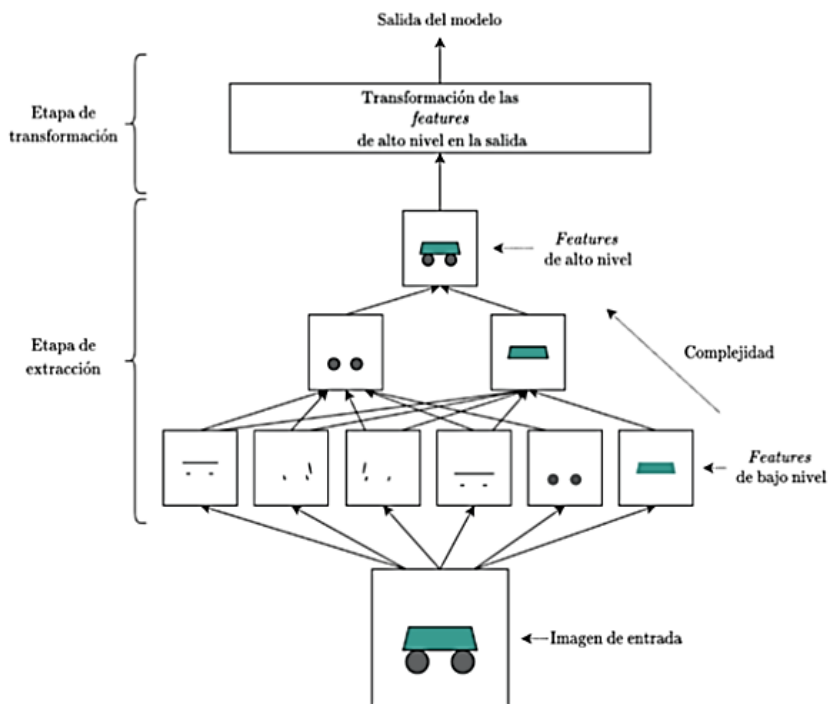


Figura 1.3 Extracción de características de menor a mayor complejidad [1]

A continuación, se va a presentar las redes de *Deep Learning* utilizadas en el proyecto de titulación

Redes convolucionales

El término de redes convolucionales fue introducido al mundo en el año 1980 por el señor Kunihiko Fukushima y fue nombrado como modelo Neocognitron, este modelo expone la existencia de dos tipos de células (célula S y C) alojadas en la corteza primaria de visión. Dentro del modelo Neocognitron las células están interconectadas y las células S extraen *features* de bajo nivel, mientras que las células C *features* de alto nivel [4] [5].

Las redes convolucionales o también llamadas *Convolutional Neural Networks* (CNN) pertenecen a la familia del *Deep Learning* y son redes que permiten realizar la detección de patrones alojados en los datos de entrada de un sistema, específicamente de imágenes. Su propósito es realizar tareas como detectar y categorizar objetos, clasificar múltiples escenas e imágenes de uso general [5].

Las redes convolucionales basan toda su existencia en operaciones de convolución. Una convolución consiste en filtrar imágenes mediante el uso de máscaras en donde cada pixel resultante es el producto de una combinación lineal de los pixeles que ingresan, una máscara es la conexión entre capas sucesivas además el uso de diferentes máscaras conlleva distintos resultados [4] [6].

Las CNN cuentan con dos etapas: la primera etapa es llamada etapa de extracción de *features*, dentro de esta las capas de convolución y agrupamiento reciben una nueva representación de los datos de entrada basada en la creación de una jerarquía de características de menor a mayor complejidad. La segunda etapa es llamada etapa de clasificación, dentro de esta los datos que ingresan son los datos de salida de la etapa anterior [5] [6].

2 METODOLOGÍA

En el presente trabajo de titulación se realizó una investigación de tipo documental, siguiendo las siguientes etapas descritas a continuación, esto se lo realizó con el objetivo de definir cada una de las actividades a realizar siguiendo un orden.

Instalación: Dentro de esta etapa se instaló todo software usado en la elaboración del proyecto, entre los que se encuentran: El programa wgit, el núcleo de Python (ipykernel), Tensorflow, el objeto de detección de Tensorflow y el modelo pre entrenado.

Datos: En esta etapa se preparó los datos que iban a ser usados para entrenar a la red neuronal. Al tener como objetivo matrículas vehiculares se pudo obtener una gran cantidad de este tipo en el sitio web Kaggle. Además de conseguir las imágenes también se consiguió sus respectivos archivos escritos en código XML.

Entrenamiento: En esta etapa se realizó el entrenamiento de la red neuronal, como punto de partida se creó un mapa de etiquetas que representa los objetos a ser detectados. Después se crearon los registros para el entrenamiento y finalmente usando los datos descargados de la plataforma Kaggle se entrenó a la red neuronal. Una vez entrenada la red se procedió a cargar el último punto de control creado a partir del entrenamiento.

Detección a partir de una imagen cargada: Esta etapa es la primera prueba de funcionamiento de la red neuronal convolucional. Para esto se cargó una imagen en la cual se identificará la matrícula y el texto dentro de ella.

Almacenamiento de los resultados: En esta etapa se guardó los resultados de la detección. La imagen de la matrícula se guardó en una carpeta mientras que el texto detectado en un archivo .csv.

Detección en tiempo real: Esta etapa es la segunda y definitiva prueba de funcionamiento de la red. Se implementó el algoritmo que permite detectar en tiempo real a través de una cámara web. Los registros detectados fueron guardados en una base de datos dentro de una máquina virtual que cumple el papel de servidor web.

3 RESULTADOS

El uso *Deep Learning* permite trabajar con una gran cantidad de datos y generar nuevos conocimientos sin la necesidad de una supervisión. En la actualidad se ha vuelto más accesible gracias a la existencia de librerías *open source* y modelos pre entrenados. Dentro de *Deep Learning* se han generado redes neuronales orientadas a trabajar con datos específicos, por ejemplo, para trabajar con imágenes se usará redes convolucionales. Si se requiere obtener algún tipo de predicción se usará una red neuronal recurrente.

El proyecto tiene como objetivo usar redes neuronales convolucionales para el reconocimiento de placas vehiculares en tiempo real. Después de guardar la imagen y detectar el texto, este último será guardado como un registro dentro de una base de

datos con un identificador acompañado de la fecha y hora de ingreso. Se configuró el script para que se guarde placas únicas y no repetidas, si ocurre el último caso, dentro de la celda de detección se imprimirá un mensaje alertando que dicha placa ya está registrada y que debería probar con una nueva.

3.1 Analizar redes neuronales y librerías open source

El tipo de red neuronal utilizada para la realización del proyecto consistió en una red neuronal convolucional. Esta red está orientada para el análisis de imágenes y clasificación de elementos mediante la detección de patrones. Su estructura compuesta por las siguientes capas: En la capa de entrada ingresa la imagen a ser analizada, después pasa a la capa oculta o extracción de características, dentro de esta se puede tener una o varias capas ocultas en donde existen las capas de convolución y agrupación. Por último, se encuentra la capa de salida en donde se refleja el resultado.

El uso de modelos pre entrenados para la clasificación de objetos es lo ideal porque evita entrenar a la red desde cero y son aplicados sobre problemas específicos que requieren una clasificación. La mayoría de modelos de este tipo han sido entrenados mediante el uso de un conjunto de datos ImageNet y pueden ser clasificados por categorías de objetos, por ejemplo: casa, perro, lápiz o carro [7].

Dentro del proyecto se usó un modelo pre entrenado denominado *ssD mobilenet V2 FPNLITE 320X320*. Este modelo de detección de objetos usa modelos de detectores de disparo único *Single Shot Detector (SSD)*, es usado dentro de dispositivos con bajas características de procesamiento (*mobilenet*) permitiendo extraer características de las imágenes *Feature Pyramid Network (FPN)* [8] [9].

El modelo pre entrenado *ssD mobilenet V2 FPNLITE 320X320* es un modelo de detección que cuenta con 267 capas, el cual ha sido entrenado anteriormente con un conjunto de datos correspondiente a imágenes con una resolución de 320x320 [9].

La ventaja del modelo es el uso del detector de disparo único (SSD), esto provoca que solamente se requiera tomar una foto para realizar la detección de objetos. FPNLITE es un extractor de características creado con un diseño de pirámide, proporciona mejor precisión y velocidad al momento de realizar la detección de objetos en diferentes escalas. Después de ser entrenado, el modelo tiene un tamaño de 63 MB, lo que lo convierte en un modelo ideal para usar dentro de dispositivos con bajos requerimientos de software [8].

Una desventaja de usar el modelo es obtener mayor número de detecciones y poca precisión en ellos. También está limitado a la detección de características cuando los objetos se encuentran muy lejos o muy cerca [8].

A continuación, se va a presentar el software y las librerías *open source* utilizadas en el proyecto de titulación

Python: Es un tipo de lenguaje de programación que posee una sintaxis que permite la creación de código más legible, fue desarrollado en los años 90 por Guido van Rossum. Python es un lenguaje que se puede ejecutar por medio de un programa denominado interprete, el cual lo convierte en un lenguaje interpretado [10]. La versión que se usó para el proyecto fue la versión 2.9.

Jupyter Notebook: Aplicación cliente servidor de código abierto disponible bajo la licencia BSD modificada, pertenece al *Project Jupyter* creado a partir del proyecto IPython en el año 2014. *Jupyter notebook* es usado en su gran mayoría dentro del área de aprendizaje automático con el propósito de crear, desarrollar y trabajar con redes neuronales [11].

Jupyter notebook cuenta con dos componentes importantes para su funcionamiento: El primer componente es el núcleo o *kernel*, el cual es un motor de ejecución que permite interpretar un lenguaje de programación en específico para que pueda procesar solicitudes y regresar respuestas. El segundo componente es el *dashboard* o panel de control que funciona como interfaz que permite la administración de cada *kernel* y como sala de control donde se podrá crear o seguir trabajando con proyectos. El principal lenguaje de programación que usa es Python, aunque en la actualidad posee *kernels* para distintos lenguajes de programación tales como: Ruby, Juliam Perl, Matlab y R [12] [11].

Numpy: Se trata de una librería *open source* de Python que trabaja sobre funciones matemáticas realizando operaciones con vectores y matrices. Además, permite realizar varios procesamientos a las imágenes como el recorte a una sección de la misma [13]. La versión escogida fue la 1.23.5.

Tensorflow: Biblioteca de código abierto enfocada en Python desarrollada en 2015 por la empresa Google orientada para trabajar dentro de la creación de modelos aprendizaje automático, esto con el propósito de crear, desarrollar y trabajar con redes neuronales, las cuales podrán detectar y estudiar patrones en un intento de igualar la capacidad del cerebro [14] [15].

Tensorflow posee una estructura flexible permitiendo trabajar en diversas plataformas como CPUs o GPUs además de estar disponibles para los sistemas operativos Linux de 64 bits, macOS y para plataformas móviles como iOS y Android. Estas características hacen que se convierta en una aplicación escalable, de tal modo que podrá ser usada dentro de equipos o servidores. En la actualidad Tensorflow puede ser usado en distintos lenguajes de programación como C++, C#, Ruby o Julia [15].

El uso de Tensorflow dentro de las redes neuronales permitirá realizar la detección, reconocimiento y clasificación de imágenes mediante el establecimiento de características, esto con el objetivo de identificar una imagen específica dentro de un conjunto de imágenes que agrupan varias clases. La versión escogida fue la 2.11.

OpenCV: Se trata de una librería *open source* multiplataforma orientada al dominio de visión por computadora que en sus inicios fue desarrollada por INTEL [16].

Esta librería permite mostrar, guardar, agregar y mezclar imágenes. El objetivo de usar opencv es realizar el análisis de las imágenes que ingresan dentro de la red neuronal [16]. La versión escogida fue la 4.6.0.66.

Navegador Anaconda: Se trata de una interfaz gráfica de escritorio para los usuarios proveniente de la distribución Anaconda, la misma que es usada para inicializar y administrar aplicaciones, paquetes y entornos sin la necesidad de usar comandos. Esta interfaz está disponible para los sistemas operativos Windows, Linux y macOS [17].

Anaconda Navegador tiene las siguientes aplicaciones para realizar trabajos de ciencia de datos, Machine Learning, o procesamiento de datos entre las que se encuentran: Jupyter lab, Jupyter notebook, Spyder, Datalore, Qt console, PyCharm, etc.

XAMPP: Es una colección de paquetes de software libre usado de manera local dentro del computador sin la necesidad de tener conexión a internet. Esta herramienta permite gestionar servidores web, base de datos y lenguajes de programación. Una ventaja que tiene es que es multiplataforma funcionando en Windows y Linux [18].

XAMPP cuenta con los siguientes paquetes: Apache orientado a servidores web, MySQL usada para base de datos, usa lenguajes de programación como PHP y Perl, finalmente usa un servidor FTP llamado ProFTPD [19].

EasyOCR: Librería usada para realizar el reconocimiento óptico de caracteres. La versión escogida fue la siguiente 1.6.2. [20]

3.2 Diseñar la solución de cada escenario mediante redes convolucionales

Las imágenes se representan digitalmente a través de una matriz, razón por la cual se optó por usar una red neuronal convolucional. El funcionamiento de la red mencionada inicia con la detección de patrones a partir de las imágenes ingresadas, al entrar en la red se extrae las características y clasifica la imagen en una categoría. Para realizar esto, la primera capa se encarga de extraer características básicas como líneas o bordes, después pasa a una capa superior que interconecta los elementos básicos y empieza a detectar diferentes formas.

Las siguientes capas superiores combinan la información de las anteriores hasta llegar al punto de obtener características muy complejas del objeto. El resultado del proceso es clasificar a la imagen dentro de una categoría.

El modelo pre entrenado de detección de objetos *ssD mobilenet V2 FPNLITE* utiliza la función de activación denominada Unidad Lineal Rectificada (ReLU), la cual es usada por la mayoría de redes neuronales convolucionales. Esta función devuelve únicamente valores positivos descartando negativos y al momento de presentarse valores negativos en la entrada la función le transforma en cero.

El motivo de usar ReLU es porque dentro del procesamiento de imágenes los valores negativos entregados no son importantes y por eso se los establece en 0, por el contrario, es importante obtener valores positivos después de la convolución para que puedan pasar a las siguientes etapas de entrenamiento. Se optó por usar el modelo *ssD mobilenet V2 FPNLITE* que contiene la función de activación ReLU porque es ideal para dispositivos con bajos requerimientos de software [21] [22].

Una vez entrenada la red neuronal convolucional y verificada la detección de matrículas a partir de imágenes se procede a realizar algunos procesos. Estos permitirán recortar la sección de la matrícula usando la librería Numpy y OpenCV. Después se realizará la detección del texto mediante OCR utilizando la librería EasyOCR sobre la Región de Interés (ROI). Finalmente se almacenarán los datos obtenidos en forma de registros.

La librería OpenCV permite importar una imagen para su estudio, mientras que la librería Numpy permite hacer distintos procesamientos a una imagen como cambiar pixeles reducir el color, concatenación o cambiar a blanco y negro. El procesamiento utilizado es el corte de una sección. Para realizar el corte se especifica un área, después se define una función que delimite las coordenadas y dimensiones de la imagen. El

producto de este proceso será el recorte de una sección de la imagen a la cual se le realizará el reconocimiento óptico de caracteres.

La librería llamada EasyOCR permite realizar el reconocimiento óptico de caracteres (OCR), esto con el propósito de detectar el texto en imágenes sobre la región de interés, la cual es representada por la sección cortada de la imagen. Para usar la librería mencionada se debe instalar la dependencia PyTorch. Se ingresa la imagen, se crea una variable para mantener la ruta, después es definido el lenguaje a usar y como resultado devolverá una matriz con el texto detectado.

Para visualizar el resultado se usa el método `rectángulo` que toma los parámetros y coordenadas de la imagen, para el caso mostrado se toma el ancho y alto. La variable denominada umbral de detección permite comparar el tamaño del texto a detectar con el tamaño total de la imagen, si este tamaño del texto es mayor a la variable entonces será mostrado en pantalla. Este análisis en conjunto con el método `rectángulo` produce que se dibuje un cuadro alrededor del texto detectado.

Después de obtener la imagen y el texto detectado se procederá a guardar la información recolectada. Las imágenes serán almacenadas dentro de una carpeta. El texto detectado será registrado en dos fases. La primera fase consiste en usar las librerías `csv` y `uuid` para crear un archivo `.csv` y una etiqueta para cada texto detectado. La segunda fase es la creación de una base de datos alojados en una máquina virtual, dentro de la misma existirá una tabla que contenga un identificador, el texto detectado, finalizando con la hora y fecha del registro.

Para verificar la solución del proyecto se realizará la identificación de matrículas en dos escenarios, los cuales son:

1. A partir de una imagen: Dentro de esta etapa se cargará al sistema una de las imágenes usadas en el entrenamiento o alguna conseguida a través de internet. Se logrará identificar la sección que corresponde a la matrícula dentro de la imagen y después se procederá con la detección del texto, ver **Figura 3.1**.



Figura 3.1 Primer escenario de prueba

2. En tiempo real: En esta última etapa se usará la imagen de una matrícula, la cual será acercada a la cámara web de la computadora y empezará la detección. El sistema tomará captura de la imagen y después detectará su texto. Finalmente, la detección del texto será guardada como registro en una tabla dentro de una base de datos alojada en un servidor web virtual. El escenario puede ser observado en la **Figura 3.2**.



Figura 3.2 Segundo escenario de prueba

3.3 Implementar las soluciones mediante redes neuronales para el despliegue de los diferentes servicios

La primera sección de la implementación de la solución se denomina instalación. Para iniciar con la implementación de la red neuronal se realiza la creación de un nuevo directorio o carpeta en la computadora, se procede a crear la carpeta llamada tic dentro del disco local D, para esto se abre el terminal de Windows y se dirige al disco D con el comando `cd D`, después con el comando `mkdir` se crea la carpeta con nombre tic.

Se ingresa en la carpeta creada desde el terminal y se procede a crear el entorno virtual con el comando `python -m venv placas`. Después se realiza la activación del entorno con el comando `.\placas\Scripts\activate`, para verificar la activación e ingreso de manera exitosa se observa que el nombre del entorno esta antes de la ruta del terminal, los comandos se pueden observar en la **Figura 3.3**

```
D:\tic>python -m venv placas
D:\tic>
D:\tic>.\placas\Scripts\activate
(placas) D:\tic>
```

Figura 3.3 Creación y activación del entorno virtual

Luego de crear, activar y entrar en el entorno virtual se procede a instalar la dependencia *ipykernel* que proporciona el núcleo IPython para Jupyter. El comando a usar es el siguiente: *Python -m pip install ipykernel*. La instalación se observa en la **Figura 3.4**.

```
(placas) D:\tic>python -m pip install ipykernel
Collecting ipykernel
  Downloading ipykernel-6.17.0-py3-none-any.whl (138 kB)
----- 138.5/138.5 KB 822.5 kB/s eta 0:00:00
Collecting nest_asyncio
  Downloading nest_asyncio-1.5.6-py3-none-any.whl (5.2 kB)
Collecting ipython>=7.23.1
  Downloading ipython-8.6.0-py3-none-any.whl (761 kB)
----- 761.1/761.1 KB 1.9 MB/s eta 0:00:00
Collecting jupyter_client>=6.1.12
  Downloading jupyter_client-7.4.4-py3-none-any.whl (132 kB)
----- 132.3/132.3 KB 3.8 MB/s eta 0:00:00
```

Figura 3.4 Instalación de *ipykernel*

Después de instalar el núcleo para Jupyter se procede a ejecutar el comando *Jupyter notebook*, esto abrirá una pestaña en el navegador web donde se visualiza las carpetas creadas en el entorno virtual. En este apartado se puede crear y ejecutar cuadernos Jupyter. La creación del cuaderno se la puede observar en la **Figura 3.5**.

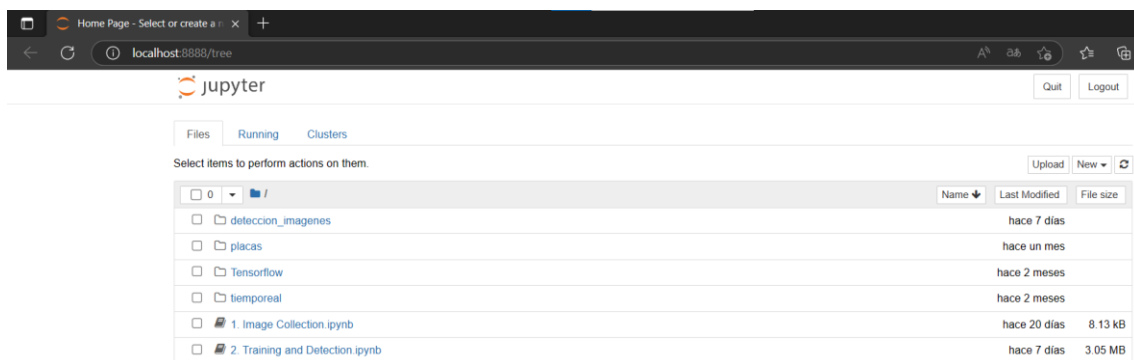


Figura 3.5 Ejecución y visualización de Jupyter notebook

Tras la creación correcta de Jupyter *notebook* en la web se procede a instalar la dependencia *ipykernel* dentro del entorno virtual, para esto se utiliza el comando *Python -m ipykernel install --name=placas*. Esto se refleja en la **Figura 3.6**.

```
(placas) D:\tic>python -m ipykernel install --name=placas_
```

Figura 3.6 Instalación *ipykernel* dentro del entorno virtual.

Dentro del cuaderno Jupyter se selecciona el *kernel* que tiene como nombre el entorno virtual. Después se ingresa en cualquier archivo y se reinicia el *kernel*, esto es necesario para que se guarde la configuración realizada. **Figura 3.7**.

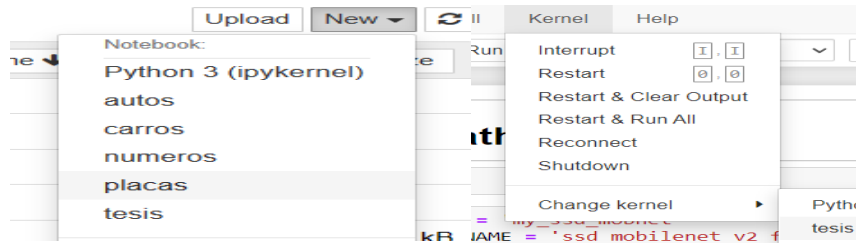


Figura 3.7 Selección del nuevo núcleo y reinicio.

Dentro de la carpeta creada en el terminal se procede a instalar la librería NumPy, necesario para almacenar y trabajar con el procesamiento de imágenes. La instalación se la realiza con el comando `pip3 install six numpy Wheel`.

Después de instalar la librería NumPY se procede a instalar la biblioteca de código abierto para *Machine learning* llamada Tensorflow, la cual permite trabajar con redes neuronales. Su instalación se realiza dentro del entorno virtual con el comando `conda install tensorflow`. Al no especificar la versión se instalará la versión más actual que tenga.

Para verificar la correcta instalación se observa que en la carpeta creada existe una nueva carpeta denominada Tensorflow, al ingresar contendrá las carpetas *models*, *protoc*, *scripts* y *workspace*. Esto es reflejado dentro de la **Figura 3.8**.

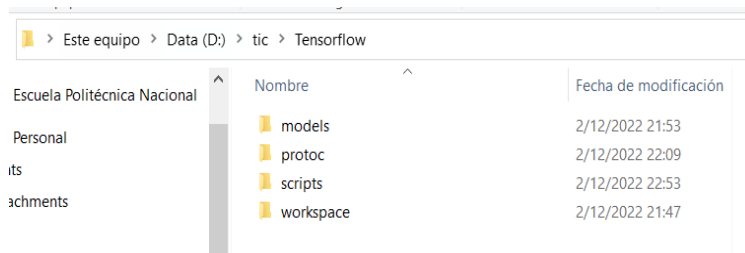


Figura 3.8 Verificación de la instalación de Tensorflow.

Dentro de Jupyter *notebook* en la web se procede a entrar en el archivo denominado *2. Training and Detection*. Dentro de este archivo se realiza el programa que permitirá reconocer las placas en tiempo real. Se inicia con la importación del sistema operativo, para esto se crea una nueva celda y se ingresa el comando `import os`, después se presiona la tecla shift + enter para ejecutar el comando. En la celda siguiente se define las variables del modelo entre las que están: nombre del modelo, nombre del modelo pre entrenado, dirección url del modelo pre entrenado, nombre del script de registros tf y nombre de mapa de etiquetas. Esto se puede observar en la **Figura 3.9**.

```

import os

Nombre_modelo = 'my_ssd_mobnet'
Nombre_preentrenado = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
URL_preentrenado = 'http://download.tensorflow.org/models/object_detection/tf2/2020071
Nombre_tf = 'generate_tfrecord.py'
Nombre_mapa = 'mapa_etiquetas.pbtxt'

```

Figura 3.9 Creación de rutas y variable.

En la siguiente celda se debe crear las variables de entorno y su contenido. Después se realiza la creación de las carpetas que contendrán a dichas variables. Esto es observado en la **Figura 3.10**.

```

paths = {
    'Espacio_Trabajo': os.path.join('Tensorflow1', 'espacio_trabajo'),
    'Espacio_Scripts': os.path.join('Tensorflow', 'scripts'),
    'Espacio_APIMODELO': os.path.join('Tensorflow', 'modelos'),
    'Espacio_anotacion': os.path.join('Tensorflow', 'espacio_trabajo', 'anotaciones'),
    'Espacio_imagen': os.path.join('Tensorflow', 'espacio_trabajo', 'imagenes'),
    'Espacio_modelo': os.path.join('Tensorflow', 'espacio_trabajo', 'modelos'),
    'Espacio_modelopreentrenado': os.path.join('Tensorflow', 'espacio_trabajo', 'modelo_preentrenado'),
    'Espacio_puntocontrol': os.path.join('Tensorflow', 'espacio_trabajo', 'modelos', Nombre_modelo),
    'Espacio_salida': os.path.join('Tensorflow', 'espacio_trabajo', 'modelos', Nombre_modelo, 'export'),
    'Espacio_TFJS': os.path.join('Tensorflow', 'espacio_trabajo', 'modelos', Nombre_modelo, 'tfjsexport'),
    'Espacio_TFLITE': os.path.join('Tensorflow', 'espacio_trabajo', 'modelos', Nombre_modelo, 'tfliteexport'),
    'Espacio_protocolo': os.path.join('Tensorflow', 'protocolo')
}

```

Figura 3.10 Creación de carpetas que alojaran a las variables

El siguiente paso es la importación de la herramienta informática wget con el comando *import wget*. Después se realiza el script de verificación, esto permite comprobar si todas las herramientas o dependencias han sido creadas e instaladas correctamente. Si todo está en orden aparecerá el siguiente mensaje *Ran 24 test in 24.112s*, esto significa que dentro de 24.112 segundos se realizó la verificación de la correcta instalación de las dependencias y herramientas. La verificación puede ser vista de mejor manera en la **Figura 3.11**.

```

SCRIPT_verificacion = os.path.join(paths['Espacio_APIMODELO'], 'investigacion', 'deteccion_objeto', 'builders', 'model_builder_tf2_test.py')
# Verificacion de la instalacion
!python {SCRIPT_verificacion}

```

```

[ RUN      ] ModelBuilderTF2Test.test_session
[ SKIPPED ] ModelBuilderTF2Test.test_session
[ RUN      ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor): 0.0s
I1115 20:38:16.927826 980 test_util.py:2458] time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor):
0.0s
[ OK      ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
[ RUN      ] ModelBuilderTF2Test.test_unknown_meta_architecture
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
I1115 20:38:16.927826 980 test_util.py:2458] time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
[ OK      ] ModelBuilderTF2Test.test_unknown_meta_architecture
[ RUN      ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
I1115 20:38:16.931817 980 test_util.py:2458] time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
[ OK      ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
-----
Ran 24 tests in 34.112s
OK (skipped=1)

```

Figura 3.11 Verificación de la instalación.

En la siguiente celda se ingresa el comando `import object detection`. Si después de ejecutar el comando aparece un error relacionado al no encontrar la variable, esto se soluciona reiniciando el `kernel` y ejecutándolo de nuevo. A continuación, se procede a descargar el modelo llamado `ssD mobilenet V2 FPNLITE 320X320`, el cual será configurado para que pueda reconocer las matrículas vehiculares. Para realizar la descarga se usa `wget`, después será transferido a la carpeta con el nombre modelo pre entrenado y finalmente será desempaquetado para que se convierta en un archivo `tar.gz` y sea descomprimido. El proceso de descarga es observado en la **Figura 3.12**.

```

if os.name == 'nt':
    wget.download(URL_preentrenado)
    !move {Nombre_preentrenado+'.tar.gz'} {paths['Espacio_modelopreentrenado']}
    !cd {paths['Espacio_modelopreentrenado']} && tar -zxvf {Nombre_preentrenado+'.tar.gz'}

3% [..] 679936 / 20515344
11% [.....] 2351104 / 20515344
19% [.....] 3956736 / 20515344
26% [.....] 5382144 / 20515344
41% [.....] 8454144 / 20515344
48% [.....] 9986048 / 20515344
56% [.....] 11599872 / 20515344
64% [.....] 13221888 / 20515344
72% [.....] 14868480 / 20515344
80% [.....] 16515072 / 20515344
88% [.....] 18112512 / 20515344
97% [.....] 20013056 / 20515344
100% [.....] 20515344 / 20515344Se han movido 1 arch
ivos.
x ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/
x ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/
x ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0.data-00000-of-00001
x ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/checkpoint
x ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0.index
x ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config

```

Figura 3.12 Descarga del modelo pre entrenado

Para verificar la descarga del modelo se debe dirigir a la carpeta del proyecto, dentro de Tensorflow se entra en el apartado `workspace` y se logra observar que en la carpeta de modelos pre entrenados se encuentra la descarga con el nombre `ssD mobilenet V2 FPNLITE 320X320`. Esto puede ser visto de mejor manera en la **Figura 3.13**

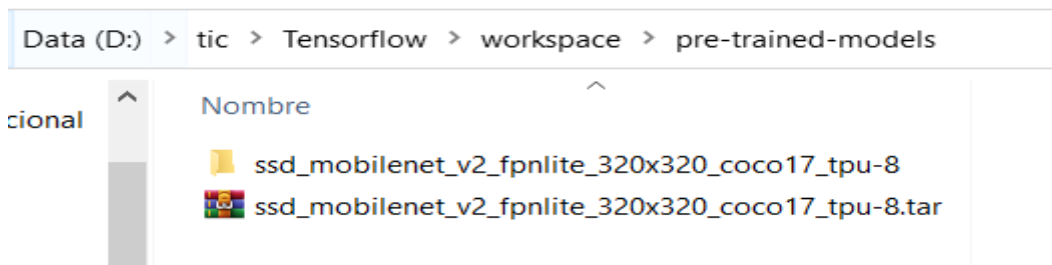


Figura 3.13 Modelo pre entrenado.

La segunda sección de la implementación de la solución se denomina datos. En este apartado se va a usar un conjunto de datos que permitirán entrenar y probar la red convolucional, para logra obtener los datos se usa la página kaggle, en el buscador se

coloca *Car license plate detection* y se procede con la descarga. Después de descargar el archivo se debe descomprimir en una carpeta dentro del computador. En la **Figura 3.14** se puede observar la página web de donde fueron obtenidos los datos.

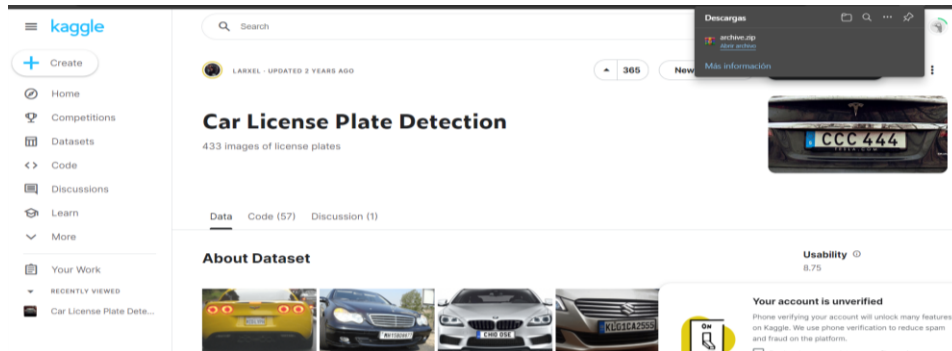


Figura 3.14 Descarga de datos.

Dentro del siguiente apartado `D:\tic\Tensorflow\workspace\images` se debe descomprimir el archivo de datos, una vez realizado aparecerán dos carpetas, una contendrá imágenes y otra los archivos XML de cada imagen. Después se realiza la creación de dos carpetas denominadas *train* y *test*. Dentro de la carpeta *train* se debe copiar y pegar la mayoría de las imágenes con sus anotaciones en archivo XML, mientras que en la carpeta *test* el resto de las mismas. Esto se refleja en la **Figura 3.15**.

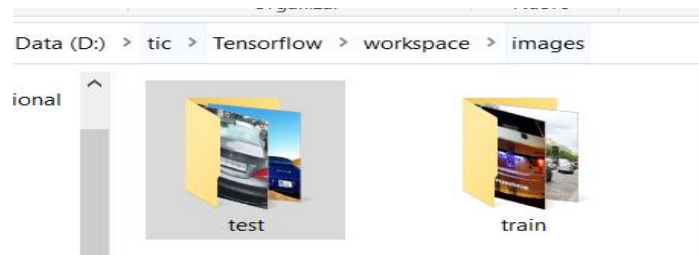


Figura 3.15 Creación carpetas *train* y *test*.

La tercera sección de la implementación de la solución se denomina entrenamiento. Antes de iniciar con el entrenamiento de la red se debe crear un mapa de etiquetas, un mapa de etiquetas representa todos los objetos posibles que va a detectar el modelo, el nombre de la etiqueta a usar se denomina *licence*, este nombre debe coincidir con el nombre de la imagen que se encuentra en los archivos XML que fueron descargados. El comando se refleja en la **Figura 3.16**.

```
etiqueta = [{'name':'licence', 'id':1}]
```

Figura 3.16 Creación del mapa de etiquetas

Dentro de la ubicación D:\tic\Tensorflow\scripts se crea un archivo Python denominado *generated tfrecord*. Este archivo permite guardar los registros creados que serán usados por la red neuronal convolucional. La creación del archivo se observa en la **Figura 3.17**.

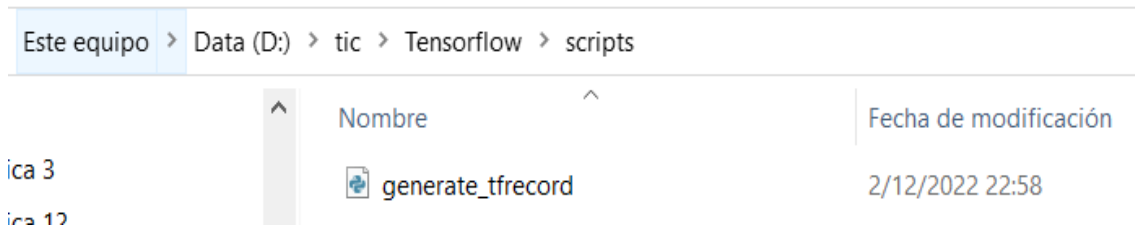


Figura 3.17 Ejecución del script y creación del archivo.

Después de crear los registros se procede a instalar la biblioteca de Python pytz, para esto se ejecuta la celda que contiene el comando *pip install pytz*. Después de instalar pytz se ingresa en la siguiente celda que permite crear los registros tf para la carpeta *train* y *test*. Los comandos para la creación están representados en la **Figura 3.18**.

```
!python {files['Nombre_tf']} -x {os.path.join(paths['Espacio_imagen'], 'train')} -l {files['Mapa_etiqueta']} -o {files['Mapa_etiqueta']}
!python {files['Nombre_tf']} -x {os.path.join(paths['Espacio_imagen'], 'test')} -l {files['Mapa_etiqueta']} -o {files['Mapa_etiqueta']}
```

Successfully created the TFRecord file: Tensorflow\workspace\annotations\train.record

C:\Users\BRYAN\anaconda3\lib\site-packages\scipy__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.4
 warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}")

Successfully created the TFRecord file: Tensorflow\workspace\annotations\test.record

C:\Users\BRYAN\anaconda3\lib\site-packages\scipy__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.4
 warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}")

Figura 3.18 Creación de los registros tf para las carpetas train y test.

Finalmente se procede a entrenar a la red convolucional, para esto se ejecuta el siguiente comando dentro de del entorno virtual activado. El comando de entrenamiento usará 10 000 *steps* porque se requiere que la detección sea lo más precisa posible. El proceso se entrenamiento se puede observar en la **Figura 3.19**.

El uso de 10 000 *steps* para el entrenamiento de la red se basó en las pruebas realizadas en varios entrenamientos, la disminución o incremento de este valor es directamente proporcional al tiempo que dura el entrenamiento y al número de puntos de control creados. El uso del último punto de control creado en cada prueba de entrenamiento daba como resultado un cierto porcentaje de precisión en la detección de las placas vehiculares.

Después de realizar las pruebas variando el número de *steps* en el entrenamiento de la red, el uso del valor de 10 000 significó una precisión aceptable para detección de placas vehiculares. El tiempo aproximado en el cual se entrenó la red fue de 8 horas.

```
(placas) D:\tic>
(placas) D:\tic>python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_mobnet --pipe
e.config --num_train_steps=10000
D:\tic\placas\lib\site-packages\tensorflow_addons\utils\ensure_tf_install.py:53: UserWarning: TensorFlow Addons supports using Python ops for all
11.0 (nightly versions are not supported).
The versions of TensorFlow you are currently using is 2.11.0 and is not supported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration, either change the TensorFlow version or the TensorFlow Addons's versi
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
warnings.warn(
2022-12-07 10:26:02.576048: T tensorflow/core/platform/cpu_feature_guard.cc:102] This TensorFlow binary is optimized with oneAPI Deep Neural Net
```

Figura 3.19 Comando y entrenamiento de la red.

Después de finalizar el entrenamiento se observa la creación de varios puntos de control dentro de la carpeta que contiene al modelo pre entrenado, esto se observa en la **Figura 3.20**. El número de estos archivos está determinado por el número de *steps* usados en el entrenamiento.

Nombre	Fecha de modificación	Tipo	Tamaño
export	2/12/2022 21:47	Carpeta de archivos	
tflexport	2/12/2022 21:47	Carpeta de archivos	
tfliexport	2/12/2022 21:47	Carpeta de archivos	
train	7/12/2022 19:26	Carpeta de archivos	
checkpoint	7/12/2022 23:55	Archivo	1 KB
ckpt-5.data-00000-of-00001	7/12/2022 21:55	Archivo DATA-000...	20.282 KB
ckpt-5.index	7/12/2022 21:55	Archivo INDEX	47 KB
ckpt-6.data-00000-of-00001	7/12/2022 22:15	Archivo DATA-000...	20.282 KB
ckpt-6.index	7/12/2022 22:16	Archivo INDEX	47 KB
ckpt-7.data-00000-of-00001	7/12/2022 22:38	Archivo DATA-000...	20.282 KB
ckpt-7.index	7/12/2022 22:38	Archivo INDEX	47 KB
ckpt-8.data-00000-of-00001	7/12/2022 22:58	Archivo DATA-000...	20.282 KB
ckpt-8.index	7/12/2022 22:58	Archivo INDEX	47 KB
ckpt-9.data-00000-of-00001	7/12/2022 23:19	Archivo DATA-000...	20.282 KB
ckpt-9.index	7/12/2022 23:19	Archivo INDEX	47 KB
ckpt-10.data-00000-of-00001	7/12/2022 23:37	Archivo DATA-000...	20.282 KB
ckpt-10.index	7/12/2022 23:37	Archivo INDEX	47 KB
ckpt-11.data-00000-of-00001	7/12/2022 23:55	Archivo DATA-000...	20.282 KB
ckpt-11.index	7/12/2022 23:55	Archivo INDEX	47 KB
pipeline	7/12/2022 18:35	Archivo de origen ...	5 KB

Figura 3.20 Creación de los puntos de control.

Después de obtener los puntos de control se ingresa en Jupyter *notebook* y se realiza la carga del modelo. Para realizar esto se debe colocar el nombre del último punto de control creado, en nuestro caso es ckpt-11. Este proceso contiene la configuración,

creación del modelo de detección, ingreso del último punto de control generado y la función de detección. Esto se observa en la **Figura 3.21**.

```
# este si
# Carga de la configuracion y creacion de un modelo de deteccion
configuracion = util_configuracion.obtener_configuracion_desde_pipeline_file(files['configuracion_pipeline'])
modelo_deteccion = builder_modelo.build(modelo_configurado=configuracion['modelo'], is_training=False)

# Ultimo punto de control
ckpt = puntocontrol.tf.v2(modelo=modelo_deteccion)
ckpt.restaurado(os.path.join(paths['Espacio_puntocontrol'], 'ckpt-11'))

@tf.function
def funcion_deteccion(imagen):
    imagen, forma = detection_model.preprocess(imagen)
    prediccion = modelo_deteccion.prediccion(imagen, shapes)
    detecciones = modelo_deteccion.post_proceso(prediccion, forma)
    return detecciones
```

Figura 3.21 Ingreso del último punto de control

La cuarta sección de la implementación de la solución se denomina detección de matrículas a partir de una imagen. Dentro de esta sección se importa las librerías cv2 y numpy. Finalmente se carga la imagen que va a ser estudiada y cuya matrícula será reconocida. Para realizar esto se debe ingresar la carpeta y el nombre de la imagen con su formato. La descripción de este proceso puede ser visto en la **Figura 3.22**.

```
# este si
Espacio_imagen = os.path.join(paths['Espacio_imagen'], 'inge4.jfif')
```

Figura 3.22 Ingreso de la imagen a ser detectada.

En la **Figura 3.23** se observa el comando que permite realizar los siguientes procesos. En primer lugar, la importación de la imagen seleccionada anteriormente, después la creación de las funciones que en conjunto a la red entrenada procederán a detectar la matrícula vehicular dentro de la imagen. Finalmente se realizará la impresión de la imagen con la detección.

```

# este si
img = cv2.imread(Espacio_imagen)
imagen_numpy = numpy.arreglo(img)

tensor_entrada = tf.convertir_tensor(np.expand_dims(imagen_numpy, 0))
detecciones = funcion_deteccion(tensor_entrada)

numero_detecciones = int(detecciones.pop('numero_detecciones'))
detecciones = {key: value[0, :numero_detecciones].numpy()
               for key, value in items.deteccion()}
detecciones['numero_detecciones'] = numero_detecciones

# Las clases de deteccion tienen que ser enteros
detecciones['clase_deteccion'] = detecciones['clase_deteccion'].astype(np.int64)

etiqueta_identificacion_compensacion = 1
imagen_numpy_con_detecciones = imagen_numpy.copia()

viz_utils.visualizacion_cajas_y_etiquetas_en_matriz_imagen(
    imagen_numpy_con_detecciones,
    detecciones['caja_deteccion'],
    detecciones['clase_deteccion']+etiqueta_identificacion_compensacion,
    detecciones['puntaje_deteccion'],
    categoria_indice,
    use_normalized_coordinates=True,
    maxima_caja_dibujo=5,
    minimo_puntaje_umbral=.8,
    modoagnostico=False)

plt.imshow(cv2.cvtColor(imagen_numpy_con_detecciones, cv2.COLOR_BGR2RGB))
plt.show()

```

Figura 3.23 Comando de detección.

En la **Figura 3.24** se puede observar la detección de la matrícula a partir de la importación de una imagen. Se visualiza que enmarca los borde alrededor de la matrícula vehicular, esto significa que la red puede reconocer la matrícula del vehículo.

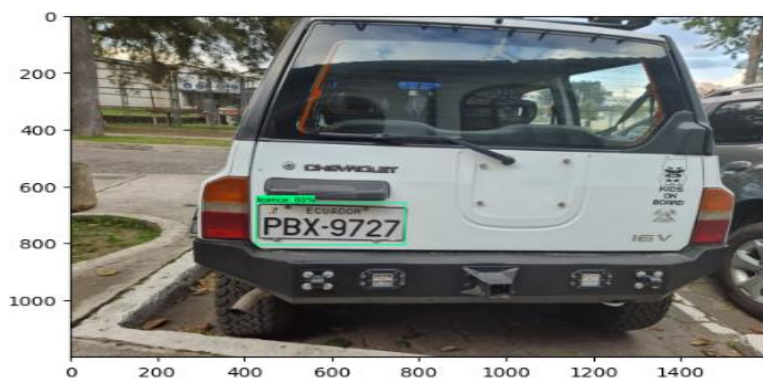


Figura 3.24 Detección de la matrícula.

Tras conseguir detectar la matrícula desde una imagen el siguiente paso es aplicar la detección *Optical Character Recognition* (OCR), esto permite detectar y extraer el texto de las imágenes que en nuestro caso es el número de matrícula. Para realizar esto se debe instalar la librería *easyocr* que permite extraer texto de imágenes. Después se instalará el programa *Pytorch* que es una biblioteca de *Deep Learning* de código abierto.

Se declara una variable denominada umbral de detección de 0.7, si los resultados obtenidos están por encima de este valor serán mostrados, caso contrario no. Después se toma la imagen detectada, su puntuación y clase que superan el umbral, una vez descrito los elementos a comparar se toma la altura y el ancho de la imagen [23].

Dentro del proceso para la detección del texto de una imagen interviene la variable mencionada anteriormente. Si el texto detectado corresponde al 70 % o 0.7 de la región total entonces será registrado, caso contrario en algunas pruebas no detectará ningún texto. Se escogió este valor porque permite obtener únicamente el texto y número de la placa, por ejemplo, PIF 7789. Al inicio se propuso que el valor detectado sea 60%, esto llevo a que se detecte también el país de la matrícula, tomando el ejemplo anterior, Ecuador PIF 7789.

Se realiza un filtro OCR en la región de interés (ROI), este proceso realiza el análisis de patrones de luz y oscuridad que constituyen las letras o números de la imagen y las transforma en texto. Después se configura la región (idioma) que usara el analizador OCR para detectar el texto de nuestra imagen. El resultado se puede observar en la **Figura 3.25**.

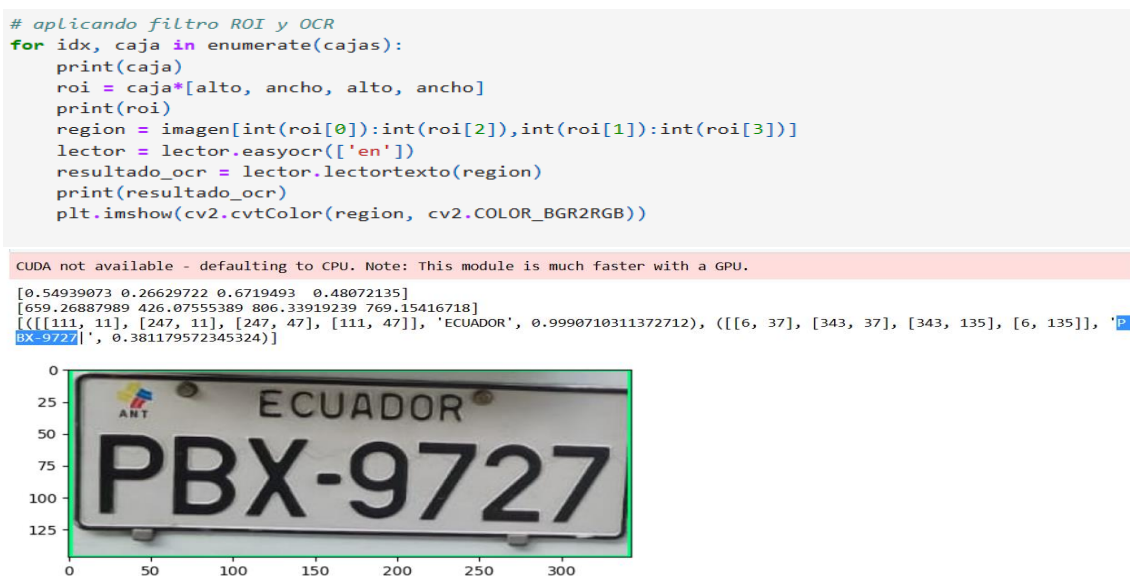


Figura 3.25 Aplicación del filtro OCR sobre la región de interés.

Para comprobar el funcionamiento de la detección se realiza el mismo procedimiento con otra imagen y el resultado se observa en la **Figura 3.26**.

```
# aplicando filtro ROI y OCR
for idx, caja in enumerate(cajas):
    print(caja)
    roi = caja*[alto, ancho, alto, ancho]
    print(roi)
    region = imagen[int(roi[0]):int(roi[2]),int(roi[1]):int(roi[3])]
    lector = lector.easyocr(['en'])
    resultado_ocr = lector.lectortexto(region)
    print(resultado_ocr)
    plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))
```

CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.

```
[0.36712414 0.11702397 0.7678868 0.87657356]
[281.95133972 119.83255005 589.73707581 897.61132812]
[[[253, 11], [564, 11], [564, 96], [253, 96]], 'ECUADOR', 0.9999389923886176),
([[64, 78], [120, 78], [120, 102], [64, 102]], 'ANT', 0.9868088473109451), ([[31, 94], [778, 94], [778, 299], [31, 299]], 'ABH-9211', 0.7644973229209836)]
```

Figura 3.26 Detección del texto a partir de una imagen.

La quinta sección de la implementación de la solución se denomina almacenamiento de los resultados. Dentro de esta sección se define la primera fase de almacenamiento, el cual consiste en guardar los resultados de la detección en un archivo .csv, para lograr el almacenamiento se utiliza la librería csv. La librería uuid permite etiquetar de manera única cada imagen detectada. Además, las imágenes detectadas serán guardadas dentro de una carpeta y el texto dentro de un archivo .csv. El almacenamiento puede ser observado en la **Figura 3.27**.

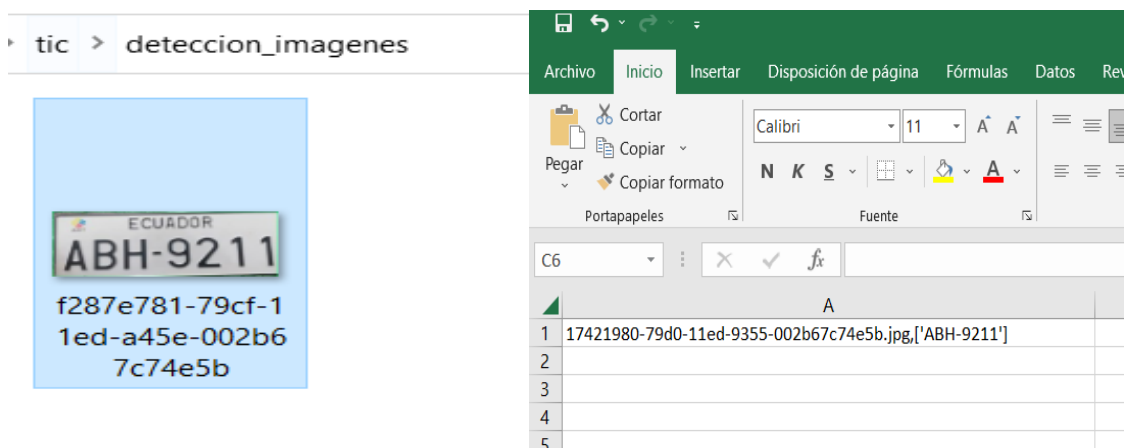


Figura 3.27 Almacenamiento de la imagen y texto detectados.

3.4 Verificar el funcionamiento de cada servicio implementado con redes neuronales

La última sección de la implementación de la solución se denomina detección en tiempo real. Para verificar la identificación de placas vehiculares mediante el uso de redes convolucionales se procede a realizar la detección en tiempo real. Dentro de este apartado se procede a realizar la segunda fase de almacenamiento, la cual consiste en

registrar el texto de las placas vehiculares en una base de datos, para esto se debe crear una máquina virtual que cumplirá el papel de servidor web, la cual alojará la base de datos.

El sistema operativo escogido fue Windows 10, se instalará la herramienta XAMPP y se procederá a activar los servicios de servidor web (Apache) y base de datos (MySQL). Se procede a crear una base de datos de nombre carros y una tabla con el nombre placas, dentro de la tabla se establecerá las siguientes variables: Id que será la etiqueta de numeración de placas. Placas que será el ingreso del texto detectado y la variable Tiempo en donde se ingresará la fecha y hora que ingreso el registro. Este proceso se puede observar en la **Figura 3.28**

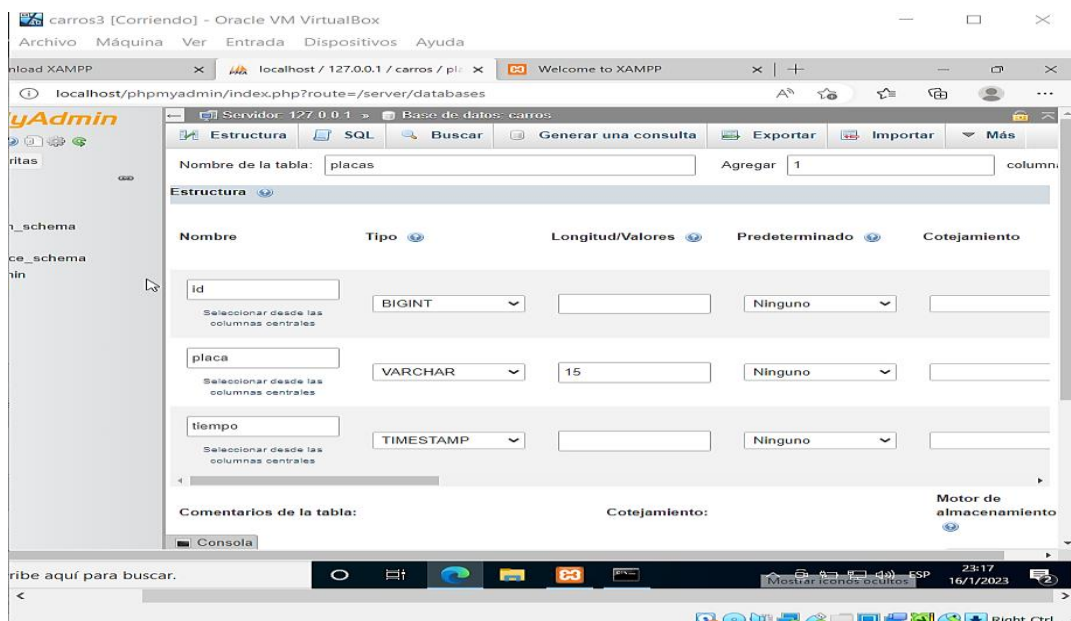


Figura 3.28 Creación de la base de datos.

Para poder guardar los registros en nuestra máquina virtual se debe realizar una conexión a la base de datos desde *Jupyter notebook*. Una vez establecida la conexión se ejecuta la celda que contiene el script de detección en tiempo real.

El comando para la conexión desde *Jupyter notebook* hacia la base de datos es el siguiente `conn=pymysql.connect`, después se ingresa los siguientes identificadores: Host que representa la dirección IP de la máquina. Puerto representa el puerto usado en la conexión a la base de datos. Usuario de la máquina. Finalmente, el nombre de la base de datos. A continuación de muestra el comando completo:

```
conn=pymysql.connect(host='192.168.1.7',port=int(3306),user='root',db="carros").
```

La cámara usada para la detección en tiempo real corresponde al modelo *integrated camera* que posee la computadora portátil Lenovo Legion 5 15ARH05. Las imágenes utilizadas son provenientes del internet bajo los formatos .jpg, .jif y .png. Al momento de capturar la imagen a través de la cámara web, estas obtienen una resolución que va en el intervalo de 70x40 a 778x308. El intervalo de pixeles que tienen las imágenes después de ser capturadas por la cámara web pueden ser visto en la **Figura 3.29**.

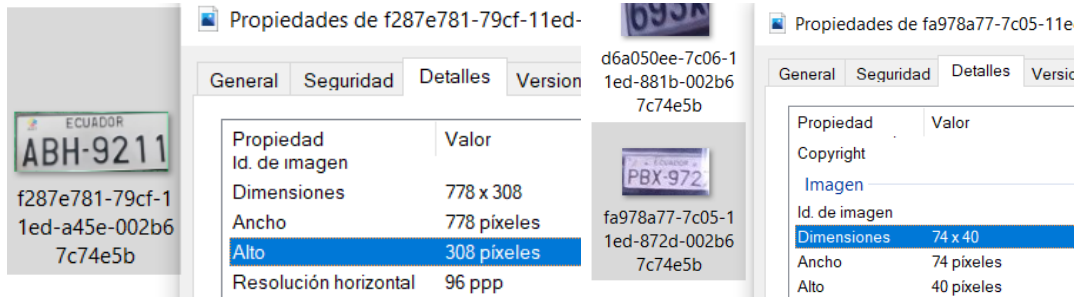


Figura 3.29 Resolución de las imágenes captadas por la cámara web.

Finalmente se ingresa el comando para la detección en tiempo real. Después de ejecutar la celda se abrirá una ventana que activará la cámara web, se procede a buscar cualquier placa y acercarla a la cámara, a continuación, se realizará el proceso de captura de la imagen de la placa y finalmente la detección del texto de la misma. Esto se ve reflejado en la **Figura 3.30**.

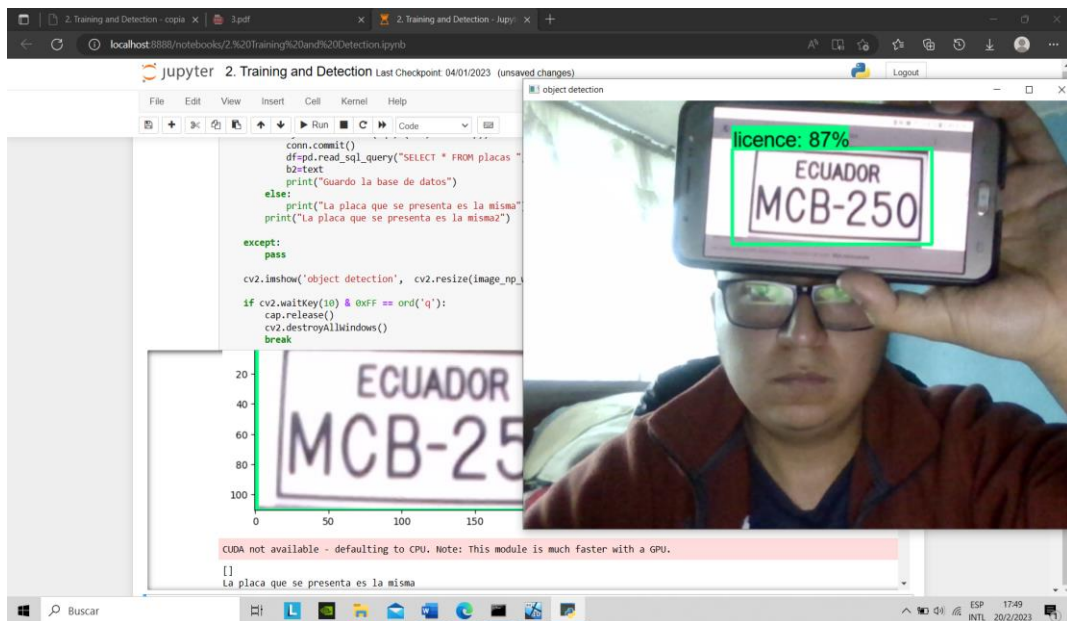


Figura 3.30 Detección en tiempo real.

En la **Figura 3.31** se puede observar la detección del texto de la matricula vehicular. Al igual que la realización mediante el ingreso de una imagen, se puede observar el recorte de la sección detectada de la imagen original, después se procede con la detección del

texto. Dentro del script se añade algunas condiciones que permiten el ingreso de matrículas únicas a la base de datos e impresión de mensajes alertando si la matrícula ya fue detectada antes o su ingreso a la base de datos.

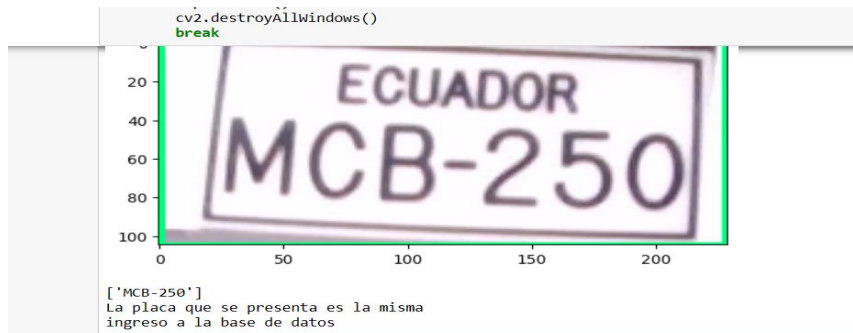


Figura 3.31 Detección de texto.

Al observar la **Figura 3.32** se puede comprobar que el registro se guarda en la base de datos alojada en la máquina virtual. Dentro de la imagen se puede observar que cada texto detectado se ingresa en una tabla, esta contiene varias columnas. La columna denominada Id representa el número de matrículas detectadas hasta el momento, la columna que corresponde a placa representa el texto detectado.

Finalmente, la columna denominada tiempo representa la hora y fecha que fue ingresada la matrícula.

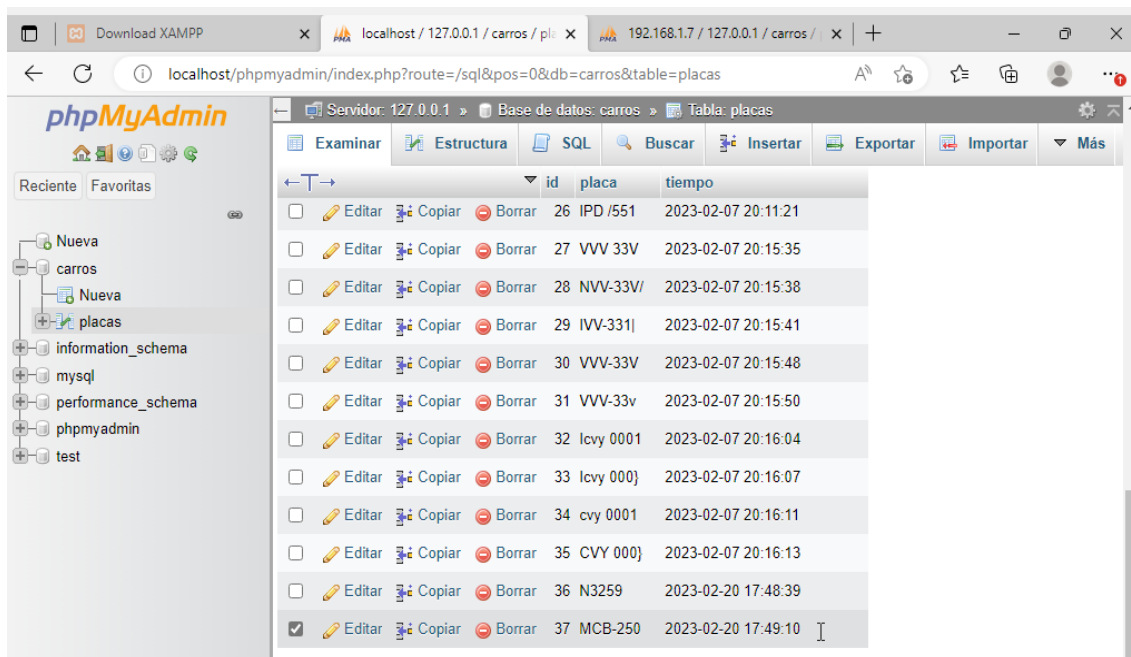


Figura 3.32 Almacenamiento en la base de datos

Como se mencionó anteriormente, la imagen detectada es almacenada dentro de una carpeta denominada detección_imagenes. El almacenamiento se cumple tanto para la

detección a partir de una imagen como para la detección en tiempo real. Esto permite obtener un registro visual del trabajo de detección. Esto puede ser observado en la **Figura 3.33**.

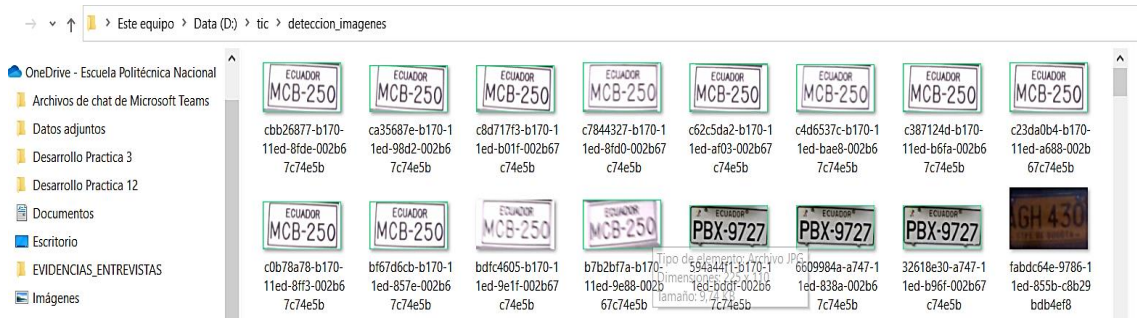


Figura 3.33 Imagen guardada de la detección en tiempo real.

4 CONCLUSIONES

- Para el objetivo analizar redes neuronales y librerías *open source* se concluye lo siguiente: el procesamiento y estudio de imágenes el uso de una red neuronal convolucional es la mejor opción porque permite la clasificación de elementos mediante la detección de patrones. Dentro del modelo pre entrenado para la detección de objetos se escogió el modelo *ssD mobilenet V2* porque su versión *mobilenet* es simple y necesita menos requerimientos de sistema a nivel de software. Dentro de las librerías *open source* escogidas se encuentran OpenCV, NumPy, EasyOCR y Tensorflow.
- Para el objetivo diseñar la solución de cada escenario mediante redes convolucionales se concluye lo siguiente: la función de activación ReLU que usa el modelo pre entrenado es la más ideal para el procesamiento y análisis de imágenes, esto es debido a que los valores negativos entregados no son importantes y por eso se los establece en 0, por el contrario, es importante obtener valores positivos después de la convolución para que puedan pasar a las siguientes etapas de entrenamiento
Para verificar el funcionamiento de la red fue necesario crear dos ambientes de prueba, el primer es el uso de imágenes que entran al sistema y otro en tiempo real en donde las imágenes pueden variar.
- Para el objetivo diseñar la solución de cada escenario mediante redes convolucionales se concluye lo siguiente: el uso de las librerías NumPy, OpenCV y EasyOCR son importantes porque permiten importar la imagen, cortar la sección de interés y finalmente detectar el texto de la imagen detectada. Esto después de ser identificada por la red neuronal.
- Para el objetivo implementar las soluciones mediante redes neuronales para el despliegue de los diferentes servicios se concluye lo siguiente: para la implementación del sistema de reconocimiento fue necesario definir etapas de desarrollo para un mejor trabajo. Las etapas definidas fueron las siguientes. En la etapa inicial se realizó la descarga e instalación de todo software necesario para el proyecto. En la etapa de datos se obtuvo los datos que permitirán entrenar la red, en este caso imágenes y archivos XML. La etapa de entrenamiento fue la más importante. Finalmente, la etapa de detección a partir de una imagen fue el primer ambiente de prueba.
- Finalmente, dentro del objetivo verificar el funcionamiento de cada servicio implementado con redes neuronales se concluye lo siguiente: la etapa de

detección en tiempo real fue el último ambiente de prueba y el más importante porque permitió conocer el funcionamiento exitoso de la red. En esta etapa se logró guardar el texto reconocido a partir de las imágenes detectadas en forma de registros dentro de una base de datos alojada en un servidor web, por último, se logró guardar la imagen detectada dentro de una carpeta.

5 RECOMENDACIONES

- Al momento de trabajar con redes neuronales convolucionales se recomienda usar las versiones más recientes de las librerías *open source* seleccionadas. El motivo es por la compatibilidad que tienen entre sí para realizar procesos. Para obtener la versión más reciente no se debe especificar la versión en el comando de instalación.
- Para proyectos futuros se recomienda el uso de Tensorflow mediante GPU, el cual es una extensión de Tensorflow que permite usar y trabajar con la tarjeta gráfica del computador. El uso de la GPU permite que el entrenamiento de la red neuronal y detección de la matrícula en tiempo real sea más rápido o fluido.
- En la realización del entrenamiento de la red neuronal convolucional se recomienda usar un número mayor de *steps*, en el proyecto se usó 10 000 *steps*. El aumento se lo hace con el objetivo de obtener una mejor precisión en la detección de matrículas vehiculares y poder suprimir errores de detección de texto por alteraciones en la imagen.
- Para futuros proyectos se podría usar una máquina virtual con sistema operativo Linux. Dentro estará alojada la base de datos y esta a su vez almacenará los registros obtenidos de la detección en tiempo real. Para futuros proyectos de este tipo, se propone almacenar las imágenes detectadas dentro de la base de datos para tener ambos resultados en un mismo lugar.
- Se recomienda mejorar la detección de texto en tiempo real ya que la mínima alteración en la imagen puede desembocar en la detección de un nuevo texto, aunque se trate de la misma matrícula.
- En la fase de almacenamiento se recomienda guardar los registros directamente en la base de datos. El almacenamiento de los registros en un archivo *.csv* se realizó para comprobar el funcionamiento de las librerías *csv* y *uuid*.

REFERENCIAS BIBLIOGRÁFICAS

- [1] A. S. d. I. Pascua, «Universidad Politecnica de Catalunya,» 31 Enero 2019. [En línea]. Available: <https://upcommons.upc.edu/bitstream/handle/2117/129220/memoria.pdf?sequence=1&isAllowed=y>. [Último acceso: 19 Enero 2023].
- [2] A. V. Barrio, «Biblioteca universitaria politecnica,» 2018. [En línea]. Available: https://oa.upm.es/53815/1/TESIS_MASTER_ALFREDO_VALLE_BARRIO_2018.pdf. [Último acceso: 19 Enero 2023].
- [3] I. P. B. . M. E. G. Arias, DEEP LEARNING, España: Uhu.es, 2021.
- [4] M. Sotaquirá, «Codificando Bits,» 23 Marzo 2019. [En línea]. Available: <https://www.codificandobits.com/blog/redes-convolucionales-introduccion/>. [Último acceso: 19 Enero 2023].
- [5] P. D. Pusiol, «Core,» 17 Febrero 2014. [En línea]. Available: <https://core.ac.uk/reader/72041560>. [Último acceso: 19 Enero 2023].
- [6] J. D. Suárez, «Deposito de investigacion de Sevilla,» 2017. [En línea]. Available: https://idus.us.es/bitstream/handle/11441/69564/TFG_Jaime%20Dur%c3%a1n%20Su%c3%a1rez.pdf?sequence=1&isAllowed=y. [Último acceso: 19 Enero 2023].
- [7] «Mathworks,» 2023. [En línea]. Available: <https://la.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html>. [Último acceso: 23 Marzo 2023].
- [8] «Roboflow,» 2023. [En línea]. Available: <https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/mobilenetv2-ssd-fpn>. [Último acceso: 23 Marzo 2023].
- [9] «Edge Impuse,» 2022. [En línea]. Available: <https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/mobilenetv2-ssd-fpn>. [Último acceso: 23 Marzo 2023].
- [10] «Python,» 2023. [En línea]. Available: <https://www.python.org/about/>. [Último acceso: 19 Enero 2023].

- [11] J. Paredes, «Escuela britanica de artes creativas y tecnologicas,» 18 Noviembre 2022. [En línea]. Available: <https://ebac.mx/blog/jupyter-notebook>. [Último acceso: 23 Enero 2023].
- [12] «Jupyter,» 2023. [En línea]. Available: <https://jupyter-notebook.readthedocs.io/en/latest/notebook.html>. [Último acceso: 23 Enero 2023].
- [13] «NumPy,» 2023. [En línea]. Available: <https://numpy.org/doc/stable/>. [Último acceso: 23 Enero 2023].
- [14] D. O. Delgado, «OpenWebinars,» 15 Septiembre 2017. [En línea]. Available: <https://openwebinars.net/blog/que-es-tensorflow/>. [Último acceso: 23 Enero 2023].
- [15] R. d. I. Vega, «Pharos,» 2023. [En línea]. Available: <https://pharos.sh/reconocimiento-de-imagenes-en-python-con-tensorflow-y-keras/>. [Último acceso: 23 Enero 2023].
- [16] «OpenCV,» 2023. [En línea]. Available: <https://opencv.org/about/>. [Último acceso: 23 Enero 2023].
- [17] «Anaconda,» 2023. [En línea]. Available: <https://docs.continuum.io/navigator/>. [Último acceso: 23 Enero 2023].
- [18] «Apache Friends,» 2023. [En línea]. Available: <https://www.apachefriends.org/es/about.html>. [Último acceso: 23 Enero 2023].
- [19] M. D. L. C. B. NIEVES, «Nettix,» 4 Noviembre 2021. [En línea]. Available: <https://www.que.es/2021/11/04/que-es-xampp/>. [Último acceso: 23 Enero 2023].
- [20] «Pipi.org,» [En línea]. Available: <https://pypi.org/project/easyocr/>. [Último acceso: 20 Enero 2023].
- [21] D. S. Team, «Data Science,» 2021. [En línea]. Available: <https://datascience.eu/es/aprendizaje-automatico/funcion-de-activacion-relu/>. [Último acceso: 24 Marzo 2023].
- [22] «Programador clic,» [En línea]. Available: <https://programmerclick.com/article/40491417292/>. [Último acceso: 24 Marzo 2023].
- [23] «Keras ocr,» [En línea]. Available: <https://keras-ocr.readthedocs.io/en/latest/api.html>. [Último acceso: 24 Marzo 2023].

6 ANEXOS

ANEXO I. Certificado de originalidad

ANEXO II. Enlaces

ANEXO III. Códigos Fuentes

ANEXO I: Certificado de Originalidad

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 2 de marzo de 2023

De mi consideración:

Yo, Fernando Vinicio Becerra Camacho, en calidad de Director del Trabajo de Integración Curricular titulado implementación de redes neuronales para identificar elementos gráficos elaborado por el estudiante Bryan Oswaldo Yanza Maila de la carrera en **TECNOLOGÍA SUPERIOR EN REDES Y TELECOMUNICACIONES**, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

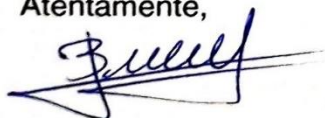
El documento escrito tiene un índice de similitud del 12%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el link del informe generado por la herramienta Turnitin.

https://epnecuador-my.sharepoint.com/:b:/g/personal/fernando_becerrac_epn_edu_ec/EXL5hf0UFANFuhmiicA6-GEBV4u3147dKdfmACfH3i8d5Q?e=YMby8M

Atentamente,



Fernando Vinicio Becerra Camacho

Docente

Escuela de Formación de Tecnólogos

ANEXO II: Enlaces

Dentro de esta sección se encuentra el código QR que redirige hacia el video en donde se presenta el funcionamiento y pruebas del proyecto.

Anexo II.I Código QR de la implementación y pruebas de funcionamiento



ANEXO III: Códigos Fuente

RUTAS Y VARIABLES DE CONFIGURACION

In []:

```
import os
```

In []:

```
Nombre_modelo = 'my_ssd_mobnet'
```

```
Nombre_preentrenado = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
```

```
URL_preentrenado =  
'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'
```

```
Nombre_tf = 'generate_tfrecord.py'
```

```
Nombre_mapa = 'mapa_etiquetas.pbtxt'
```

In []:

```
paths = {
```

```
    'Espacio_Trabajo': os.path.join('Tensorflow1', 'espacio_trabajo'),
```

```
    'Espacio_Scripts': os.path.join('Tensorflow', 'scripts'),
```

```
    'Espacio_APIMODELO': os.path.join('Tensorflow', 'modelos'),
```

```
    'Espacio_anotacion': os.path.join('Tensorflow', 'espacio_trabajo', 'anotaciones'),
```

```
    'Espacio_imagen': os.path.join('Tensorflow', 'espacio_trabajo', 'imagenes'),
```

```
    'Espacio_modelo': os.path.join('Tensorflow', 'espacio_trabajo', 'modelos'),
```

```
    'Espacio_modelopreentrenado': os.path.join('Tensorflow',  
'espacio_trabajo', 'modelo_preentrenado'),
```

```
    'Espacio_puntocontrol': os.path.join('Tensorflow',  
'espacio_trabajo', 'modelos', Nombre_modelo),
```

```
    'Espacio_salida': os.path.join('Tensorflow', 'espacio_trabajo', 'modelos', Nombre_modelo,  
'export'),
```

```
    'Espacio_TFJS': os.path.join('Tensorflow', 'espacio_trabajo', 'modelos', Nombre_modelo,  
'tfjsexport'),
```

```
'Espacio_TFLITE':os.path.join('Tensorflow', 'espacio_trabajo','modelos',Nombre_modelo,
'tfliteexport'),
```

```
'Espacio_protocolo':os.path.join('Tensorflow','protocolo')
```

```
}
```

```
In []:
```

```
files = {
```

```
'configuracion_pipeline': os.path.join('Tensorflow1', 'espacio_trabajo','modelos',
Nombre_modelo, 'configuracion.pipeline'),
```

```
'script_tfrecord': os.path.join(paths['Espacio_Scripts'], Nombre_tf),
```

```
'mapa_etiqueta': os.path.join(paths['Espacio_annotacion'], Nombre_mapa)
```

```
}
```

1. Descarga del modelo pre entrenado

```
# Instalacion del objeto de deteccion de Tensorflow
```

```
# se instalo en el terminal al inicio del proyecto
```

```
In []:
```

```
script_de_verificacion = os.path.join(paths['Espacio_APIMODELO'], 'busqueda',
'deteccion_objeto', 'builders', 'model_builder_tf2_test.py')
```

```
# Verificacion de la instalacion
```

```
!python {script_de_verificacion}
```

```
In []:
```

```
!pip install pyyaml
```

```
In []:
```

```
import object_detection
```

```
In []:
```

```
wget.download(URL_preentrenado)
```

```
!move {Nombre_preentrenado+'.tar.gz'} {paths['Espacio_modelopreentrenado']}
```

```
!cd {paths['Espacio_modeloprentrenado']} && tar -zxvf {Nombre_preentrenado+'.tar.gz'}
```

2. Creacion del mapa de etiquetas

In []:

```
etiqueta = [{'name':'licence', 'id':1}]
```

3. Creacion de los registros tf

In []:

In []:

```
!pip install pytz
```

In []:

```
!python {files['Nombre_tf']} -x {os.path.join(paths['Espacio_imagen'], 'train')} -l  
{files['Mapa_etiqueta']} -o {os.path.join(paths['Espacio_annotacion'],  
'registro_entrenamiento')}
```

```
!python {files['Nombre_tf']} -x {os.path.join(paths['Espacio_imagen'], 'test')} -l  
{files['Mapa_etiqueta']} -o {os.path.join(paths['Espacio_annotacion'], 'registro_prueba')}
```

4. Copiar el modelo pre entrenado

In []:

```
if os.name == 'nt':
```

```
!copy {os.path.join(paths['Espacio_modeloprentrenado'], Nombre_preentrenado,  
'configuracion.pipeline')} {os.path.join(paths['Espacio_puntocontrol'])}
```

5. Actualizar el modelo

In []:

```
# este si
```

```
import tensorflow as tf
```

In []:

```
configuracion =  
util_configuracion.obtener_configuracion_desde_pipeline_file(files['configuracion_pipeline'  
])
```

```
In [ ]:
```

```
In [ ]:
```

```
configuracion_pipeline = pipeline_pb2.TrainEvalPipelineConfig()
```

```
with tf.io.gfile.GFile(files['Configuracion_pipeline'], "r") as f:
```

```
    proto_str = f.read()
```

```
    formato_texto.Merge(proto_str, configuracion_pipeline)
```

```
In [ ]:
```

```
Configuracion_pipeline.modelo.ssd.clase_numeros = len(etiqueta)
```

```
Configuracion_pipeline.entrenar_configuracion.tamaño_lote = 4
```

```
Configuracion_pipeline.entrenar_configuracion.punto_control =  
os.path.join(paths['Espacio_modeloprentrenado'], Nombre_preentrenado, 'checkpoint',  
'ckpt-0')
```

```
Configuracion_pipeline.entrenar_configuracion.tipo_punto_control = "deteccion"
```

```
Configuracion_pipeline.entrenar_lector_entrada.nombre_mapa= files['Mapa_etiqueta']
```

```
Configuracion_pipeline.entrenar_lector_entrada.tf_record_lector_entrada.entrada_path[:] =  
[os.path.join(paths['Espacio_annotacion'], 'registro_entrenamiento')]
```

```
Configuracion_pipeline.evaluar_lector_entrada[0].nombre_mapa = files['Mapa_etiqueta']
```

```
Configuracion_pipeline.evaluar_lector_entrada[0].tf_record_lector_entrada.entrada_path[:]  
= [os.path.join(paths['Espacio_annotacion'], 'registro_prueba')]
```

```
In [ ]:
```

6. Entrenamiento del modelo

```
In [ ]:
```

```
script_entrenamiento = os.path.join(paths['Espacio_APIMODELO'], 'investigacion',  
'deteccion_objeto', 'model_main_tf2.py')
```

In []:

```
comando = "python {} --model_dir={} --espacio_Configuracion_pipeline={} --  
numero_entrenar_steps=10000".format(script_entrenamiento,  
paths['Espacio_puntocontrol'],files['configuracion_pipeline'])
```

In []:

In []:

7. Carga del modelo desde el ultimo punto de control

In []:

```
import os
```

```
import tensorflow as tf
```

In []:

```
# Carga de la configuracion y creacion de un modelo de deteccion
```

```
configuracion =  
util_configuracion.obtener_configuracion_desde_pipeline_file(files['configuracion_pipeline']  
)
```

```
modelo_deteccion = builder_modelo.build(modelo_configurado=configuracion['modelo'],  
is_training=False)
```

```
# Ultimo punto de control
```

```
ckpt = puntocontrol.tf.v2(modelo=modelo_deteccion)
```

```
ckpt.restaurado(os.path.join(paths['Espacio_puntocontrol'], 'ckpt-11'))
```

```
@tf.function
```

```
def funcion_deteccion(imagen):
```

```
    imagen, forma = detection_model.preprocess(imagen)
```

```
    prediccion = modelo_deteccion.prediccion(imagen, shapes)
```

```
detecciones = modelo_deteccion.post_proceso(prediccion, forma)
```

```
return detecciones
```

8. Deteccion a partir de una imagen

```
In [ ]:
```

```
import cv2
```

```
import numpy as np
```

```
In [ ]:
```

```
# este si
```

```
categoria_indice =  
nombre_mapa_util.creacion_categoria_indice_desde_Mapas_etiquetas(files['Mapas_etiquetas'])
```

```
In [ ]:
```

```
# este si
```

```
Espacio_imagen = os.path.join(paths['Espacio_imagen'], 'inge4.jfif')
```

```
In [ ]:
```

```
# este si
```

```
img = cv2.imread(Espacio_imagen)
```

```
imagen_numpy = numpy.arreglo(img)
```

```
tensor_entrada = tf.convertir_tensor(np.expand_dims(imagen_numpy, 0))
```

```
detecciones = funcion_deteccion(tensor_entrada)
```

```
numero_detecciones = int(detecciones.pop('numero_detecciones'))
```

```
detecciones = {key: value[0, :numero_detecciones].numpy() for key, value in detecciones.items()}
```

```
detecciones['numero_detecciones'] = numero_detecciones
```

```
# Las clases de deteccion tienen que ser enteros
```

```
detecciones['clase_deteccion'] = detecciones['clase_deteccion'].astype(np.int64)
```

```
etiqueta_identificacion_compensacion = 1

imagen_numpy_con_detecciones = imagen_numpy.copia()

viz_utils.visualizacion_cajas_y_etiquetas_en_matriz_imagen(
    imagen_numpy_con_detecciones,
    detecciones['caja_deteccion'],
    detecciones['clase_deteccion']+etiqueta_identificacion_compensacion,
    detecciones['puntaje_deteccion'],
    categoria_indice,
    use_normalized_coordinates=True,
    maxima_caja_dibujo=5,
    minimo_puntaje_umbral=.8,
    modo_agnostico=False)

plt.imshow(cv2.cvtColor(imagen_numpy_con_detecciones, cv2.COLOR_BGR2RGB))

plt.show()
```

Aplicacion de deteccion OCR

In []:

```
!pip install easyocr
```

In []:

In []:

```
# este si
```

```
import easyocr
```

In []:


```

# este si

umbral_deteccion = 0.7

In [ ]:

# este si

imagen = imagen_numpy_con_detecciones

puntajes = list(filter(lambda x: x> umbral_deteccion, detecciones['puntaje_deteccion']))

cajas = detecciones['caja_deteccion'][:len(puntajes)]

clases = detecciones['clase_deteccion'][:len(puntajes)]

In [ ]:

# este si

ancho = forma.imagen[1]

alto = forma.imagen[0]

In [ ]:

# aplicando filtro ROI y OCR

for idx, caja in enumerate(cajas):

    print(caja)

    roi = caja*[alto, ancho, alto, ancho]

    print(roi)

    region = imagen[int[(roi[3]):(roi[1])],(roi[0]):(roi[2])]]

    lector = lector.easyocr(['en'])

    resultado_ocr = lector.lectortexto(region)

    print(resultado_ocr)

```

FILTRO CON OCR

```

In [ ]:

# este si

```

```

region_umbral = 0.05

In [ ]:

# este siiiii

def texto_filtrado(region, resultado_ocr, region_umbral):

    tamaño_rectangulo = forma.region[0]*forma.region[1]

    plate = []

    for resultado in resultado_ocr:

        longitud = np.sum(np.sub(resultado[0][0], resultado[0][1]))

        alto = np.sum(np.sub(resultado[0][1], resultado[0][2]))

        if longitud *alto / tamaño_rectangulo > region_umbral:

            plate.append(resultado[1])

    print(longitud , ancho)

    return plate

```

In []:

```

In [ ]:

texto_filtrado(region, resultado_ocr, region_umbral)

```

(JUNTAR TODO)

In []:

```

# este si

plt.imshow(imagen_numpy_con_detecciones)

```

In []:

```

# este si

```

```

region_umbral = 0.05

In [ ]:

# este si

def ocr_it(imagen, detecciones, umbral_deteccion, region_umbral):

    #puntuaciones por arriba del umbral

    puntajes = list(filter(lambda x: x> umbral_deteccion, detecciones['puntaje_deteccion']))

    cajas = detecciones['caja_deteccion'][:len(puntajes)]

    clases = detecciones['clase_deteccion'][:len(puntajes)]

    # Dimensiones de la imagen.

    ancho = forma.imagen[1]

    alto = forma.imagen[0]

    # aplicacion del filtro ROI y OCR

    for idx, caja in enumerate(boxes):

        roi = caja*[alto, ancho, alto, ancho]

        region = imagen[int[(roi[3]):(roi[1])],(roi[0]):(roi[2])]]

        lector = lector.easyocr(['en'])

        resultado_ocr = lector.lectortexto(region)

        texto = texto_filtrado(region, resultado_ocr, region_umbral)

    # hice cambio aqui (plt.show())

    plt.show()

    print(texto)

```

```
return texto, region
```

```
In [ ]:
```

```
# este si
```

```
def texto_filtrado(region, resultado_ocr, region_umbral):
```

```
    tamaño_rectangulo = forma.region[0]*forma.region[1]
```

```
    plate = []
```

```
    for resultado in resultado_ocr:
```

```
        longitud,alto
```

```
        if longitud*alto / tamaño_rectangulo > region_umbral:
```

```
            plate.append(resultado[1])
```

```
    return plate
```

```
In [ ]:
```

```
# este si
```

```
texto, region = ocr_it(imagen_numpy_con_detecciones, detecciones, umbral_deteccion,  
region_umbral)
```

```
In [ ]:
```

```
# este si
```

```
region_umbral = 0.6
```

```
In [ ]:
```

```
# este si
```

```
texto, region = ocr_it(imagen_numpy_con_detecciones, detecciones, umbral_deteccion,  
region_umbral)
```

9. GUARDAR RESULTADOS

In []:

este si

import csv

import uuid

In []:

este si

def guardar_resultados(texto, region, archivo_csv, ruta_carpeta):

 nombre_imagen = '{}.jpg'.format(uuid.uuid1())

 cv2.imwrite(os.path.join(ruta_carpeta, nombre_imagen), region)

In []:

texto

In []:

guardar_resultados(texto, region, 'detection_results.csv', 'deteccion_imagenes')

10. Deteccion en tiempo real

In []:

este si

captura = cv2.VideoCapture(0)

import pymysql

import datetime

```
conn=pymysql.connect(host='192.168.1.7',port=int(3306),user='root',db="carros")
```

```
b2 = ""
```

```
while captura.isOpened():
```

```
    ret, frame = captura.lector()
```

```
    imagen_numpy = numpy.arreglo(frame)
```

```
    tensor_entrada = tf.convertir_tensor(np.expand_dims(imagen_numpy, 0))
```

```
    detecciones = funcion_deteccion(tensor_entrada)
```

```
    numero_detecciones = int(detecciones.pop('numero_detecciones'))
```

```
    detecciones = {key: value[0, :numero_detecciones].numpy() for key, value in detecciones.items()}
```

```
    detecciones['numero_detecciones'] = numero_detecciones
```

```
# Las clases de deteccion tienen que ser enteros
```

```
    detecciones['clase_deteccion'] = detecciones['clase_deteccion'].astype(np.int64)
```

```
    etiqueta_identificacion_compensacion = 1
```

```
    imagen_numpy_con_detecciones = imagen_numpy.copia()
```

```
    viz_utils.visualizacion_cajas_y_etiquetas_en_matriz_imagen(
```

```
        imagen_numpy_con_detecciones,
```

```
detecciones['caja_deteccion'],
detecciones['clase_deteccion']+etiqueta_identificacion_compensacion,
detecciones['puntaje_deteccion'],
categoria_indice,
use_normalized_coordinates=True,
maxima_caja_dibujo=5,
minimo_puntaje_umbral=.8,
modo_agnostico=False)
```

try:

```
texto, region = ocr_it(imagen_numpy_con_detecciones, detecciones, umbral_deteccion,
region_umbral)
```

```
guardar_resultados(texto, region, 'tiemporeal1.csv', 'deteccion_imagenes')
```

```
print("La placa que se presenta es la misma")
```

if texto != b2:

```
print("ingreso a la base de datos")
```

```
mycursor = conn.cursor()
```

```
sql = "INSERT INTO placas (placa,Tiempo) VALUES ( %s,%s)"
```

```
val = texto
```

```
mycursor.execute(sql, (val,timestamp))
```

```
df=pd.read_sql_query("SELECT * FROM placas ",conn)
```

```
b2=texto
```

```
print("Guardo la base de datos")
```

else:

```
print("La placa que se presenta es la misma")
```

break

In []: