IME

Clasificador de plagas de cultivos utilizando Deep Learning y YOLO

Cabrera Zanabria César

Universidad Nacional Autónoma de México
Facultad de Estudios Superiores Cuautitlán

Asesor: David Tinoco Varela

Introducción

Hoy en día, la inteligencia artificial (IA) tiene un gran impulso debido a la facilidad con la cual se puede integrar no solo en la industria, sino también, sino en cualquier elemento cotidiano alcance de cualquier individuo.

Existen un sinfín de aplicaciones en las que estos esquemas pueden ser implementados, en la misma forma, existen un sinfín de problemáticas de toda índole que puede ser tratado por medio de IA, incluyendo problemas sociales.

Un problema que puede ser tratado es la identificación de plagas en cultivos parcelarios o domésticos, con la finalidad de evitar la pérdida de estos y de hasta un 40% de producción agrícola, según la Organización de las Naciones Unidas para la Alimentación y la Agricultura. La identificación y control de plagas no es un problema pequeño, ya que estas pueden afectar la producción de alimentación en todo el mundo, por lo que, si la plaga que está atacando un cultivo es detectada a tiempo, se podrá tomar el protocolo de protección adecuado.

Para el proyecto desarrollado y aquí presentado, se busca implementar un modelo de detección de objetos mediante el uso de **Yolov3** (*You Only Look Once*) para identificar plagas que se pueden encontrar en plantas y vegetales. Yolov3 utiliza una red neuronal la cual secciona una imagen analizada y permite una clasificación en partes o por zonas, permitiendo formar una relación con el objeto a detectar para después imprimir el resultado de la detección.

Este tipo de modelo de *Deep learning* utiliza una red denominada **Darknet-53**, en nuestro caso se utilizó una red pre entrenada, con el objetivo de facilitar el entrenamiento de un set de imágenes propio, estas nuevas imágenes van en relación a nuestro objetivo particular, el cual es *detección de plagas tipo insecto que afectan plantas* (plagas generales presentes en México), este tipo de red se puede encontrar en el sitio oficial de Yolo.

Originalmente, nosotros tomamos como base una red entrenada para la identificación de cartas de poker, este trabajo lo podemos encontrar en el siguiente enlace:

https://github.com/DavidReveloLuna/Yolov3Custom,

a su vez este trabajo se basa en la información de los siguientes sitios:

https://github.com/eriklindernoren/PyTorch-YOLOv3
https://www.tooploox.com/blog/card-detection-using-yolo-on-android.

El modelo propuesto de detección de plagas está pensado para poder facilitar la identificación de estas ya sea en una parcela afectada y o en plantas que se encuentren en un invernadero o jardín.

La identificación de plagas mediante uso de esta IA facilita, o permite, que cualquier persona interesada, o afectada, pueda tomar una decisión sobre cómo combatir la plaga de manera más efectiva. Este sistema es rápido, sencillo y de fácil acceso por

lo cual no es necesario contar con un experto para poder utilizarlo, lo único indispensable es saber cómo instalarlo y seguir unos cuantos pasos para ponerlo en operación.

Desarrollo del programa

➢ ¿Qué es un detector de objetos?

Un detector de objetos puede distinguir uno o varios elementos en una imagen o entorno, tratando de implementar un modelo que logre emular lo que nosotros hacemos cuando miramos a nuestro alrededor, es decir, nosotros podemos identificar casi todos o la mayoría de elementos que observamos, además de ver características como luz, nitidez de la imagen, tamaño, distancia, etc., tratar de lograr este objetivo es algo difícil ya que entran en juego varias variables al momento de usar una IA puesto que esta no es comparable a las características de una mente humana, sin embargo, se puede utilizar para detectar elementos específicos, logrando resultados sorprendentes.

Algunas variables que entran en juego para una correcta identifiación son las siguientes:

1. Cantidad de objetos a identificar (uno o múltiples).
2. Tamaño y posición del objeto en la imagen o video (regularmente se encierra con un rectángulo).
3. Datos de entrenamiento del modelo (*data set* de imágenes).
4. Cantidad de épocas de entrenamiento (que tan confiable puede ser el modelo).
5. Tipo o redes neuronales usadas en el modelo.
6. Tipo de detección (en tiempo real o por imágenes).

➢ Detector de objetos Yolo

En el año de 2016 se crea un modelo nuevo e innovador llamado YOLO (*You Only Look Once)* la cual hace uso de una red de tipo convolucional denominada *Darknet*, esta permite la detención de objetos en tiempo real muy rápidamente y con la utilización de equipos convencionales.

Lo novedoso de este modelo es que la red neuronal se aplica a toda la imagen, esta pasa a dividir la imagen en una especie de rejilla, asignando regiones con los datos de detección del objeto haciendo el proceso más rápido y sencillo en comparación con modelos anteriores, figura 1.

Los resultados que arrojo este modelo fueron sorprendentes y se pueden checar con el llamado **COCO Dataset** donde se podía identificar hasta 80 clases de objetos distintos y designar su posición.

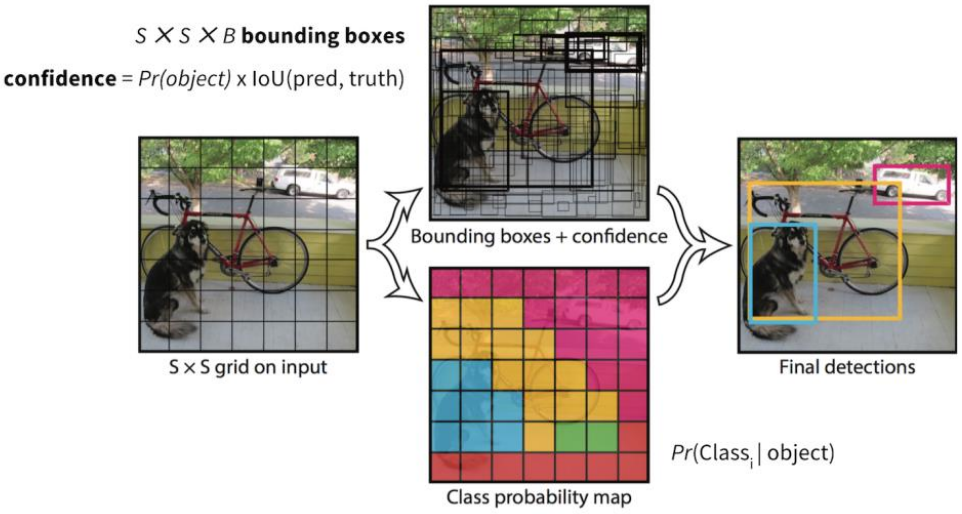

Figura1.- Procesamiento de la red convolucional en el modelo YOLO.

➢ Utilización del modelo Yolo versión 3 para hacer un detector de plagas tipo insecto

Para el desarrollo de nuestro detector de objetos se utilizará la versión 3 de Yolo, ejecutándolo en un entorno de *Anaconda*, se utilizara una red pre entrenada, originalmente utilizada para la detección de cartas de póker, a pesar de la aplicación original, se he utilizado como base para entrenar e identificar otro objeto, en este caso, las plagas.

Debido a que uno como alumno de Ingeniería Mecánica Eléctrica (IME) no posee tantos conocimientos sobre programación de este tipo de modelos, nos basaremos en las instrucciones y recomendaciones de los siguientes autores los cuales proporcionan información sobre como es el funcionamiento de la red, como entrenar la red y la ejecución de esta.

Trabajos consultados:

https://github.com/DavidReveloLuna/Yolov3Custom

https://www.toopploox.com/blog/card-detection-using-yolo-on-android

https://github.com/eriklindernoren/PyTorch-YOLOv3

A continuación indicaremos los pasos e instrucciones a seguir para lograr el desarrollo del proyecto

Paso 1 Instalación y creación de un ambiente en Anaconda *prompt*

Para poder ejecutar el modelo de detección de objetos necesitaremos descargar el entorno *Anaconda* (fig. 2), este nos permite trabajar con *Python* y otras utilizadas en inteligencia artificial.

Procedemos a descargar el programa desde el sitio oficial:

https://www.anaconda.com/products/individual

Figura 2.- Logo característico de la aplicación Anaconda.

Una vez realizada la instalación, la cual es muy sencilla, necesitaremos acceder a la consola de comandos que nos proporciona la aplicación la cual se llama *Anaconda prompt* (Fig. 3), después de iniciar procederemos a crear un ambiente el cual contenga las librerías y versiones de los programas compatibles con el modelo de detección Yolov3, si omitimos alguna librería o la instalación se realiza de manera incorrecta tendremos mensajes de error al momento de tratar de correr nuestro modelo.

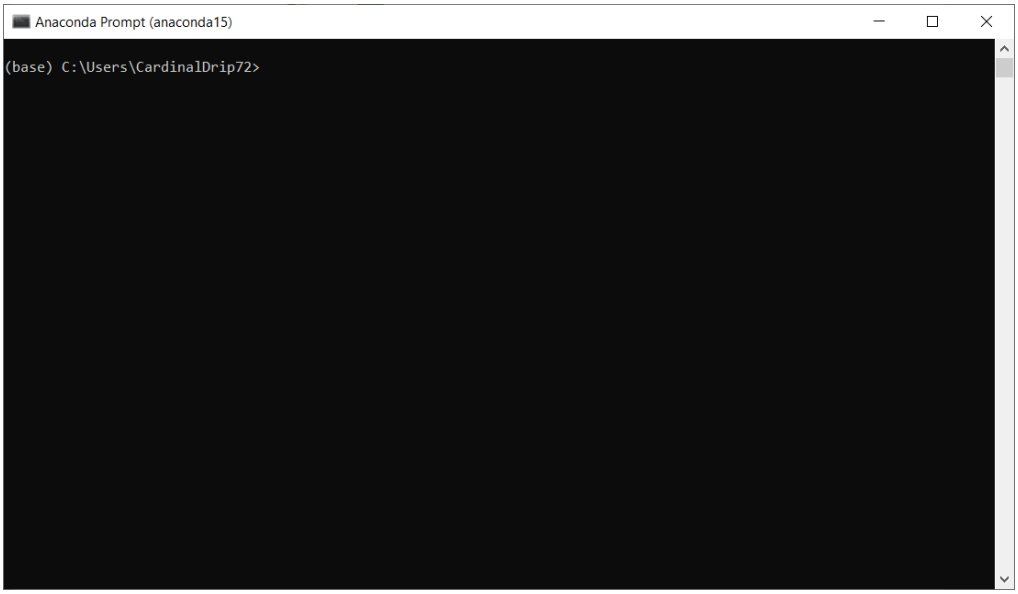

Figura 3.- Consola de comandos de anaconda prompt.

Es necesario mencionar que necesitaremos tener por lo menos una capacidad de memoria de 20 a 30 GB libres, esto debido a que las paqueterías utilizadas son muy pesadas y también necesitan memoria cuando se ejecuta el modelo.

Para poder crear un ambiente procederemos con el siguiente comando:

```
conda create -n YoloCustom anaconda python=3.6
```

Nota: La denominación de *YoloCustom* hace referencia a que usaremos el modelo Yolo, sin embargo, podemos ponerle algún otro nombre a nuestro ambiente siempre y cuando respetemos que se cree usando Python 3.6.

Puede que pida confirmación sobre la instalación de las paqueterías de Python, lo cual permitiremos.

Para poder activar el entorno usamos el siguiente comando:

```
conda activate YoloCustom
```

Una vez activado el entorno procederemos a descargar las paqueterías compatibles necesarias para la ejecución del detector, ingresamos los siguientes comandos:

```
conda install opencv-python numpy matplotlib tensorboard terminaltables pillow tqdm
```

```
conda install pytorch==1.1.0 torchvision==0.3.0 cudatoolkit=10.0 -c pytorch
```

```
conda install tensorflow
```

Paso 2 Descargar la carpeta con los archivos que se ejecutaran en el entorno de anaconda

Necesitaremos entrar en el siguiente *link* (figura 4) y descarga la carpeta comprimida que contiene los archivos de ejecución del modelo (la carpeta es del modelo de cartas de poker previamente mencionado).

https://github.com/DavidReveloLuna/Yolov3Custom

Una vez descargada la carpeta la descomprimimos (recordar la ubicación de esta carpeta).

En esta carpeta podemos encontrar los archivos necesarios para poder desarrollar el modelo de detección de cartas de póker sin embargo nosotros borraremos esa información y la sustituiremos con la de nuestro data set.

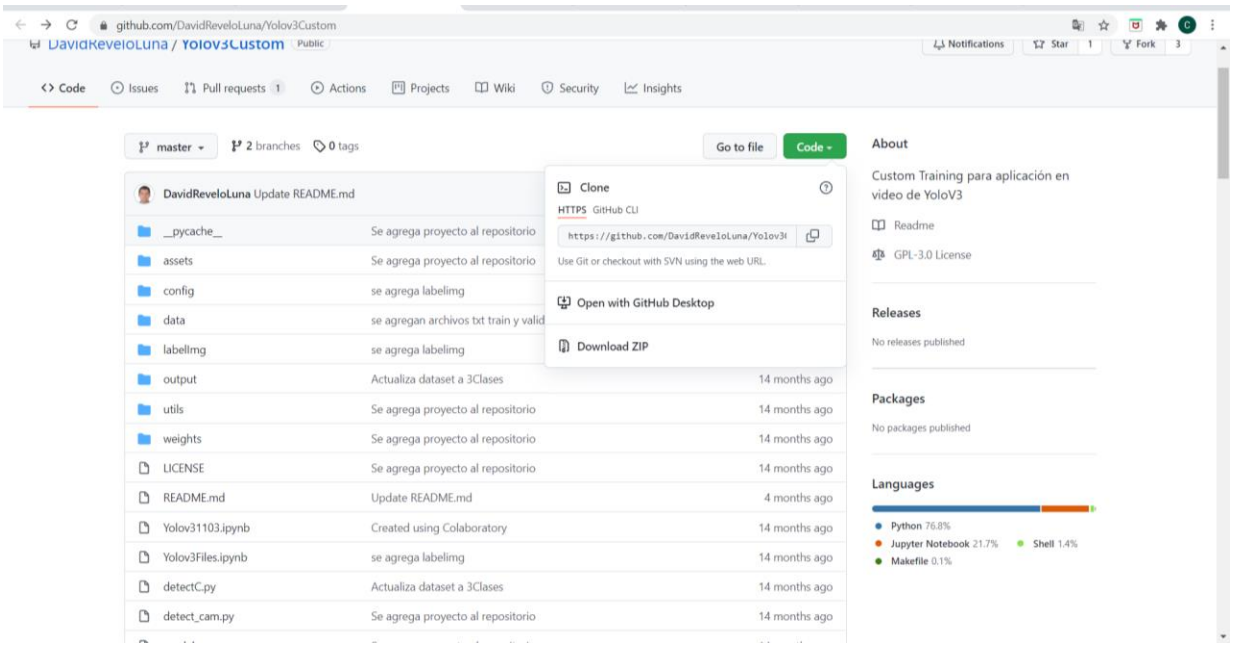

Figura 4.- Ubicación de la carpeta a descargar.

Paso 3 Creación de nuestro data set particular

Para la creación de nuestro data set es necesario primero hacer una recopilación de imágenes, en este caso, se procedio a recopilar entre 50 a 80 imágenes de dos diferentes plagas denominadas *Mosquita Blanca* y *Trips*, estos data sets se pueden ver en las figuras 5 y 6, respectivamente.

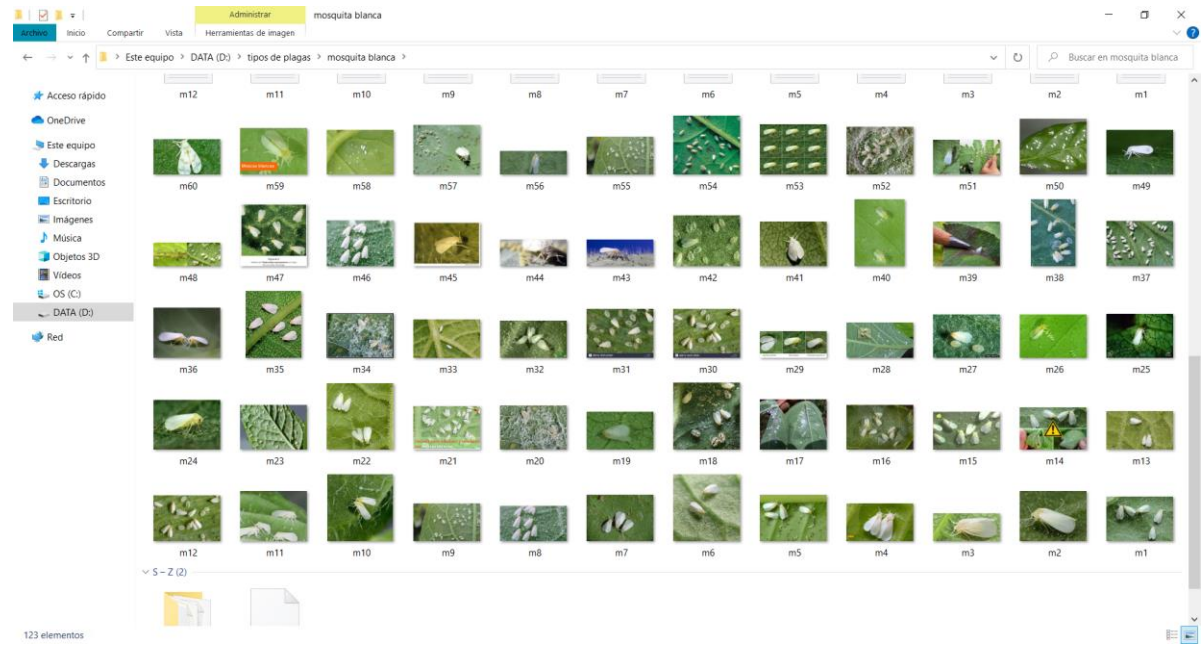

Figura 5.- Imágenes recolectadas sobre la plaga mosquita blanca.

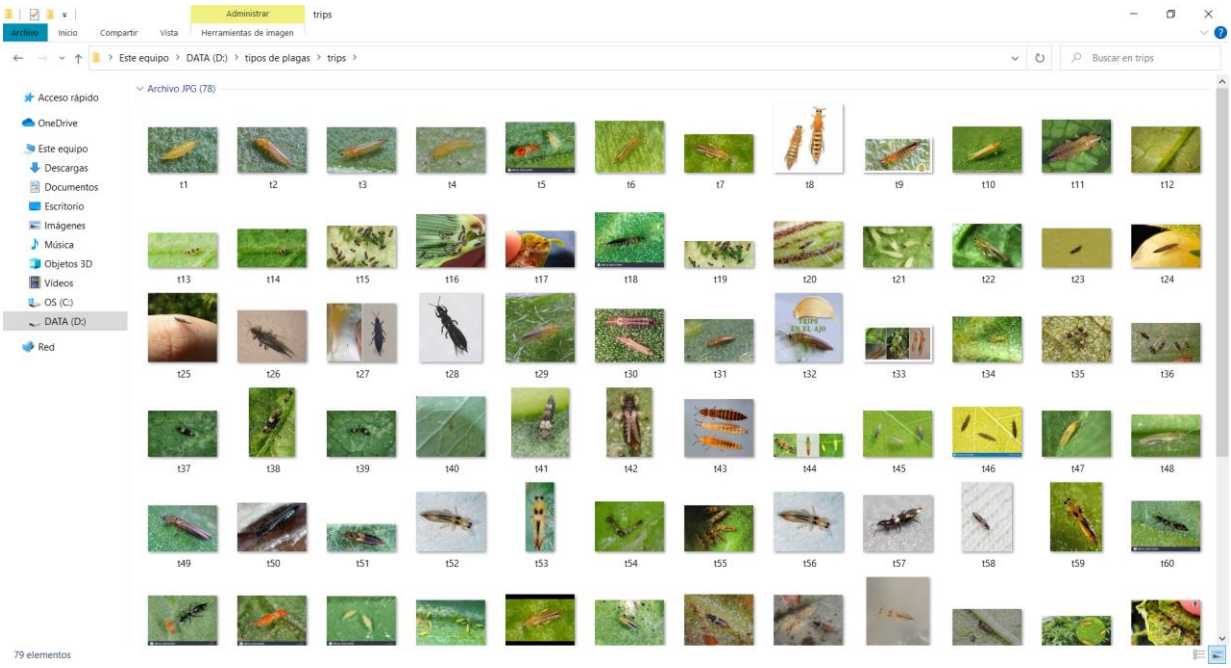

Figura 6.- Imágenes recolectadas sobre la plaga Trips.

Una peculiaridad digna de mencionar, es que las imágenes deben de encontrarse en los formatos de JPG, JPEG o PNG debido que cuando etiquetemos, el programa que nos ayuda a realizar esa tarea solo admite estos formatos sin proceder a errores.

Otra peculiaridad de las imágenes, es que no deben de estar tan saturadas en cuanto a objetivos a identificar. Lo que, si hay que procurar para generar un buen data set es que el objeto a identificar debe tener diferentes poses o ángulos, esto permite al modelo identificar en diferentes situaciones y entornos.

Por último, hay que cuidar la calidad de la imagen, en esto nos ayudara a que el modelo relacione mejor la etiquetación del objeto.

Una vez obtenidas las imágenes regresaremos al entorno que creamos en anaconda, necesitamos acceder a la carpeta anteriormente descargada y ejecutar los siguientes comandos:

```
cd labelImg
```

comando que nos permite acceder a la carpeta donde se ejecuta el programa necesario para la etiquetación de las imágenes.

```
python labelImg.py
```

Por otro lado, este otro comando ejecuta el programa (el programa tienen por nombre *Label Img*), como se ve en la figura 7.

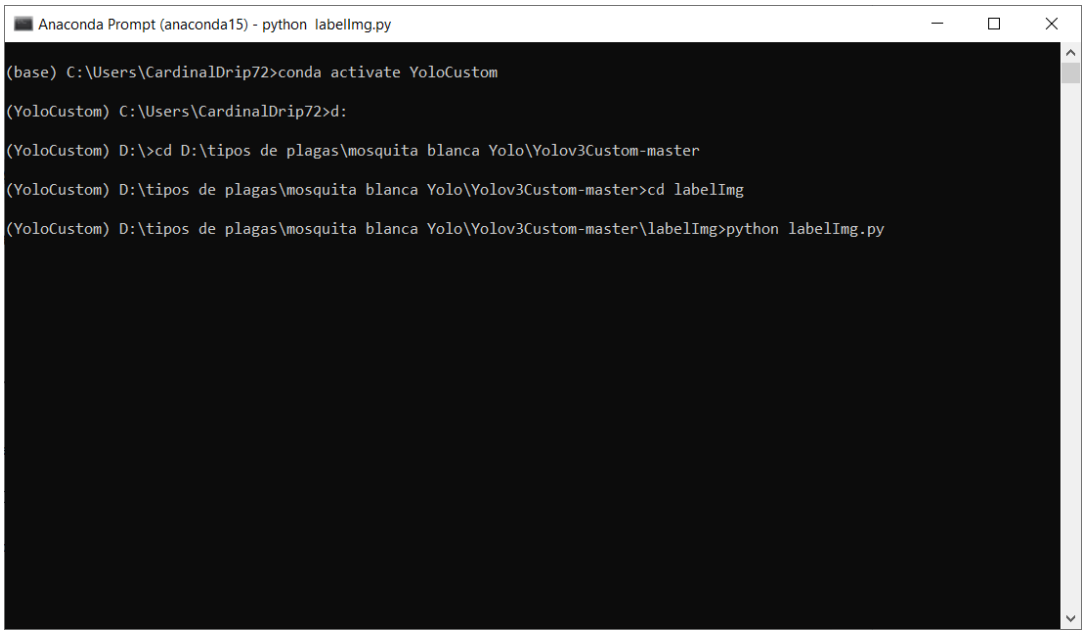

Figura 7.- Ejecución del programa de etiquetación Label Img.

Si se procedió correctamente nos abrirá otra ventana como la que se ve en la figura 8.

Figura 8.- Ventana del programa Label Img.

En la parte superior izquierda de la ventana de la figura 8, seleccionamos open dir y seleccionaremos la carpeta donde se encuentran nuestras imágenes recopiladas, mientras que en la parte de change sabe dir designaremos la carpeta donde se guardara los archivos que contienen las coordenadas del objeto etiquetado (archivos txt) y su clase.

Una vez abierto la carpeta con nuestras imágenes, necesitamos cambiar la opción de etiquetación a Yolo, esta opción se encuentra justo debajo del símbolo de guardar en la barra de herramientas de la parte izquierda, si no cambiamos esto la etiquetación no será compatible con el modelo Yolo y no se podrá hacer el entrenamiento.

Por último, para proceder a la etiquetación usamos le herramienta que dice Créate RectBox, esto nos permitirá crear un rectángulo en el objeto que pensemos identificar dentro de la imagen, también nos permitirá designar el nombre. O etiqueta, de la clase de este objeto, tal como se ve en la figura 9.

Figura *9.*- Etiquetación de las imágenes para el entrenamiento.

Recomiendo hacer la etiquetación en una sola pasada, ya que si abrimos y cerramos el programa hay una posibilidad de que al retomarlo desde algún punto en la etiquetación nos marque una clase diferente, por lo cual tendríamos dos tipos de clases de objetos para uno mismo, esto reflejará una inconsistencia en el entrenamiento posteriormente, procediendo a no reconocer todas las imágenes como el mismo objeto.

Posterior a la etiquetación, revisamos los archivos generados, copiamos los archivos de la ubicación del objeto y nos dirigimos a la carpeta de data/custom/labels donde pegaremos estas etiquetas, hacemos lo mismo con las imágenes solo que la carpeta será data/custom/images, por ultimo copiamos el archivo que contiene el nombre de nuestra clase y lo guardamos en la carpeta data/custom cambiándole la extensión a names., ver figura 10

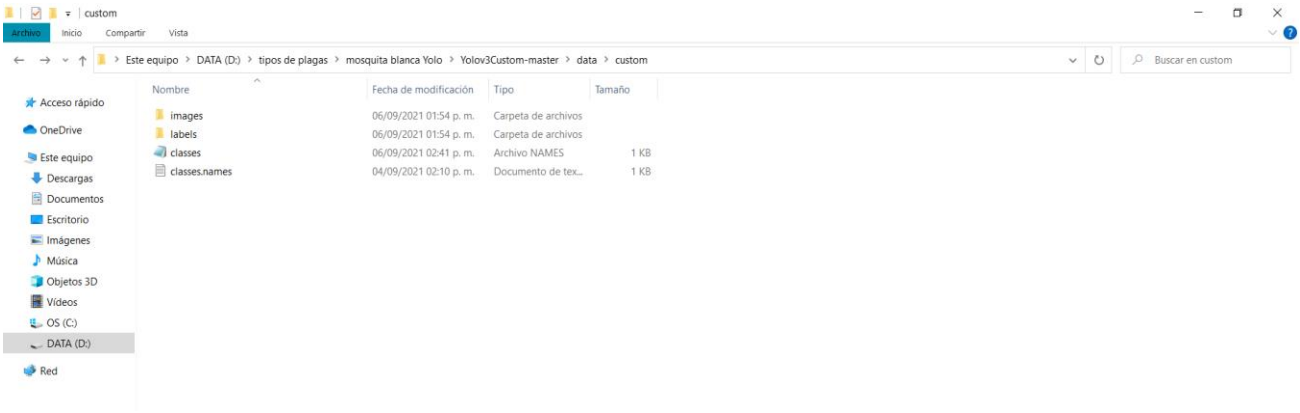

Figura 10.- Ubicación de los archivos que generamos.

Nota: Estas carpetas tendrán que ser previamente vaciadas ya que contienen los archivos del data set de cartas de poker, si dejamos tales archivos, estos se revolverán y puede ocasionar problemas en el entrenamiento.

Por último, tenemos que generar o designar que archivos serán usados para entrenar el modelo y que archivos serán usados como validación del entrenamiento.

Para lo anterior regresamos a nuestra consola de comandos y ejecutamos el siguiente comando:

jupyter notebook

Esto nos abrirá otra pestaña donde podremos ejecutar un código de Python, figura 11.

Seleccionamos el archivo de nombre Yolov3Files.ipyb

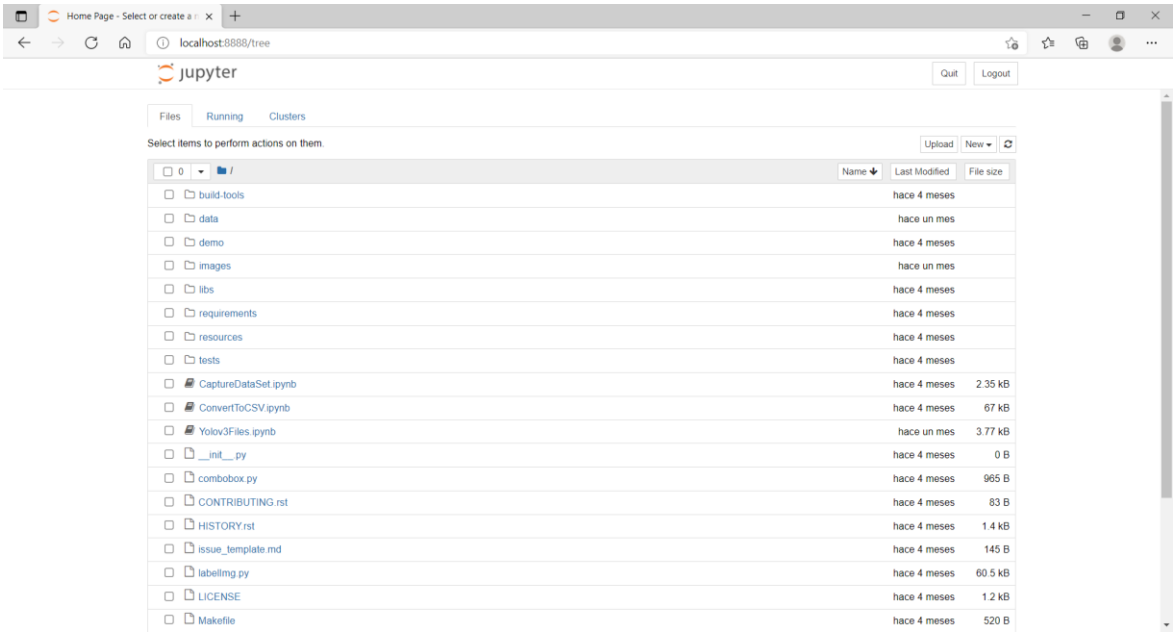

Figura 11.- Nueva pestaña para ejecutar un archivo de Python.

Lo único que debemos cambiar se encuentra en la celda 2 y es el directorio donde se encuentran guardadas nuestras imágenes del data set (usar la ubicación de la carpeta de data/custom/images).

Básicamente lo que realizaremos con la ejecución de este programa es seleccionar una cierta cantidad de las imágenes para entrenar y otra para evaluar en la ejecución de cada época del entrenamiento, figura 12.

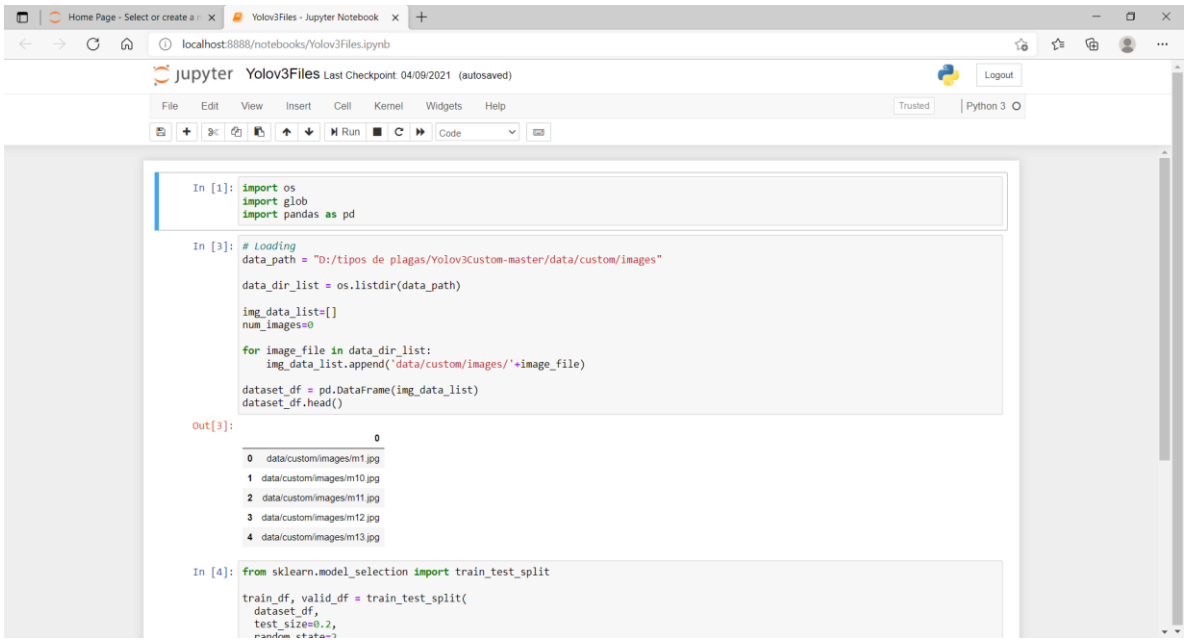

Figura 12.- Programa para seleccionar que imágenes son de entrenamiento y cuales son de verificación.

Hay que recordar que se generan dos archivos con el nombre de train y valid, los cuales se encontraran en la carpeta de labelImg, procedemos a copiarlos y pegarlos en la carpeta de data/custom.

Paso 4 Entrenamiento del modelo

Para realizar el entrenamiento procederemos a usar un entorno en línea proporcionado por *Google, Google colab*, para acceder es necesario contar con una cuenta en dicha plataforma. Este entorno nos permite ahorrar memoria y podemos usar una GPU si es que no contamos con una en nuestro equipo.

Primero iniciamos un nuevo documento y lo configuramos para ejecutarse con una GPU, figura 13.

También se puede entrenar con el entorno estándar, sin embargo, esto conlleva más tiempo en el entrenamiento.

Nota: Prevenir que solo podremos entrenar con la GPU cada cierto tiempo por lo tanto **aprovechar cada vez que nos permita usarla**.

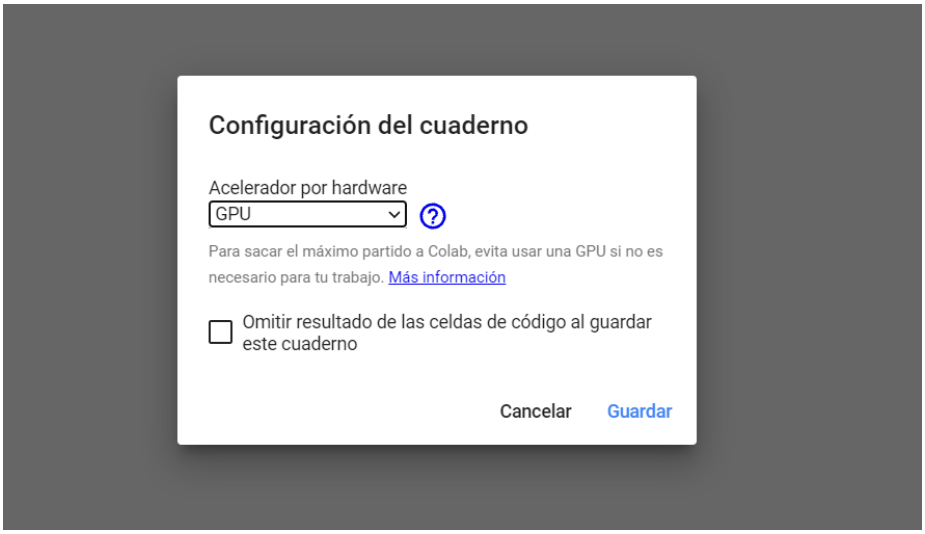

Figura 13.- Configuración del entorno de ejecución en *Google colab*.

Primero instalamos las mismas paqueterías para garantizar la compatibilidad con las de nuestro entorno de anaconda, figura 14.

```
!pip install torch==1.1 torchvision==0.3
```

```
!pip install opencv-python numpy matplotlib tensorboard terminaltables pillow tqdm
```

Posteriormente tenemos que instalar también la carpeta base, mediante el siguiente condigo:

```
!git clone https://github.com/DavidReveloLuna/Yolov3Custom.git
```

Y acedemos a ella con el siguiente comando:

```
cd Yolov3Custom
```

Posterior a esto (Figura 15), introducimos el siguiente comando que nos descargara la red pre entrenada en la carpeta de *weights*:

```
import urllib.request
urllib.request.urlretrieve('https://pjreddie.com/media/files/darknet53.conv.74','/content/Yolov3Custom/weights/darknet53.conv.74')
```

Para la siguiente parte podemos hacerla de dos maneras uno es subir los archivos directamente en las carpetas designadas como las que armamos en nuestro equipo o proceder y enlazar nuestra cuenta de *Google drive* y extraerlos de ahí, se tomó la decisión de seguir el segundo método debido a que es más rápido que le primero.

Entonces subimos nuestros archivos de la carpeta *custom* a una carpeta en drive como se ve en la figura 16.

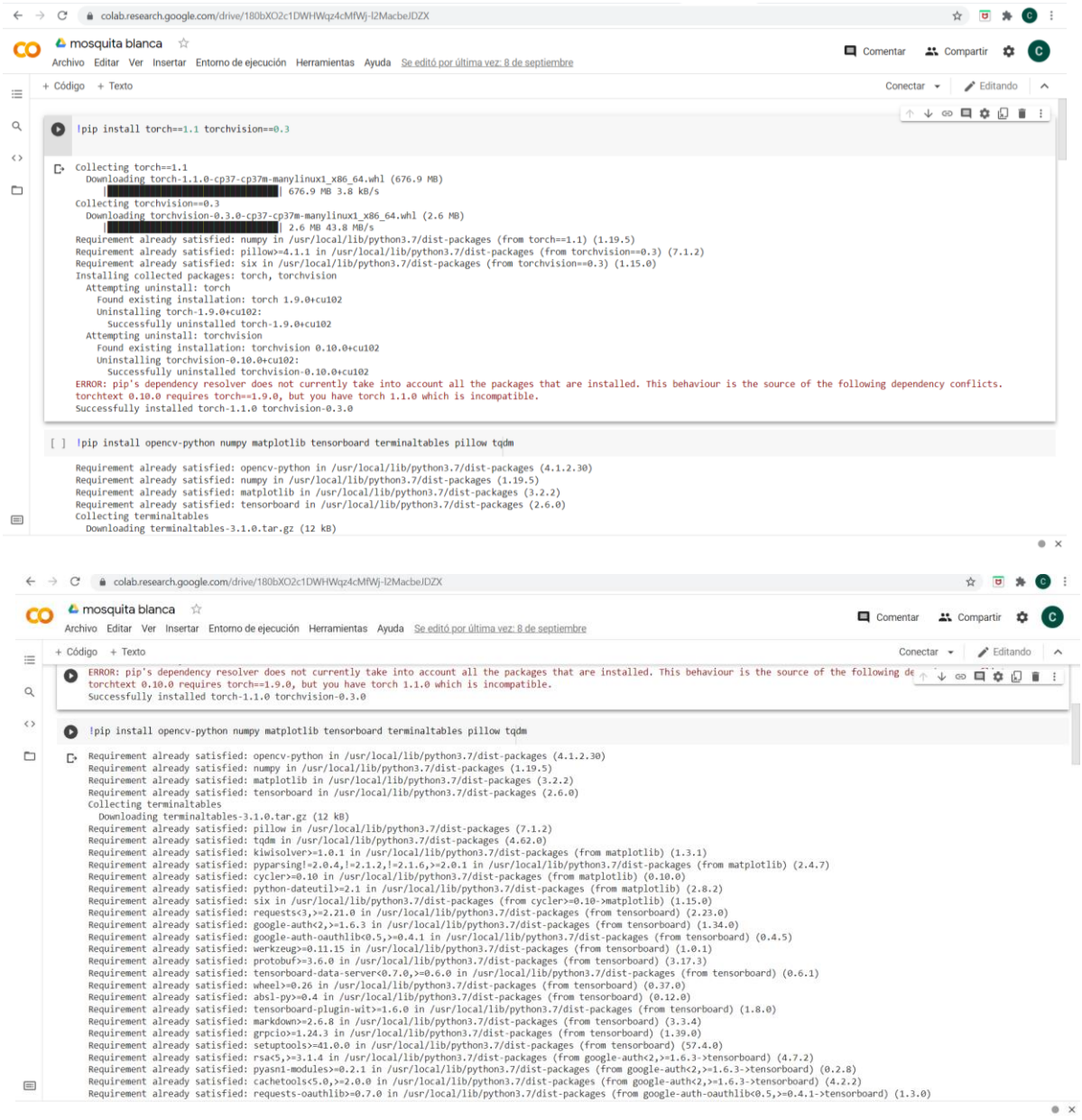

Figura 14.- Muestra de la instalación de las paqueterías.

Regresando a nuestro entorno en goolge colab, introducimos los siguientes códigos para enlazar la cuenta de drive:

```
from google.colab import drive

drive.mount('/content/drive')
```

Nos pedirá un código de confirmación, este lo obtenemos siguiendo el enlace que nos da la celda en ejecución.

Una vez enlazada la cuenta introducimos el siguiente comando:

!cp -r "/content/drive/MyDrive/plaga mosquita blanca /custom"
"/content/Yolov3Custom/data"

```
[ ] !git clone https://github.com/DavidReveloLuna/Yolov3Custom.git

    Cloning into 'Yolov3Custom'...
    remote: Enumerating objects: 4000, done.
    remote: Counting objects: 100% (248/248), done.
    remote: Compressing objects: 100% (228/228), done.
    remote: Total 4000 (delta 27), reused 212 (delta 8), pack-reused 3752
    Receiving objects: 100% (4000/4000), 256.93 MiB | 28.65 MiB/s, done.
    Resolving deltas: 100% (52/52), done.

[ ] cd Yolov3Custom

    /content/Yolov3Custom

[ ] import urllib.request
    urllib.request.urlretrieve('https://pjreddie.com/media/files/darknet53.conv.74','/content/Yolov3Custom/weights/darknet53.conv.74')

    ('/content/Yolov3Custom/weights/darknet53.conv.74',
     <http.client.HTTPMessage at 0x7fbb5670bf50>)
```

Figura 15.- Comandos introducidos.

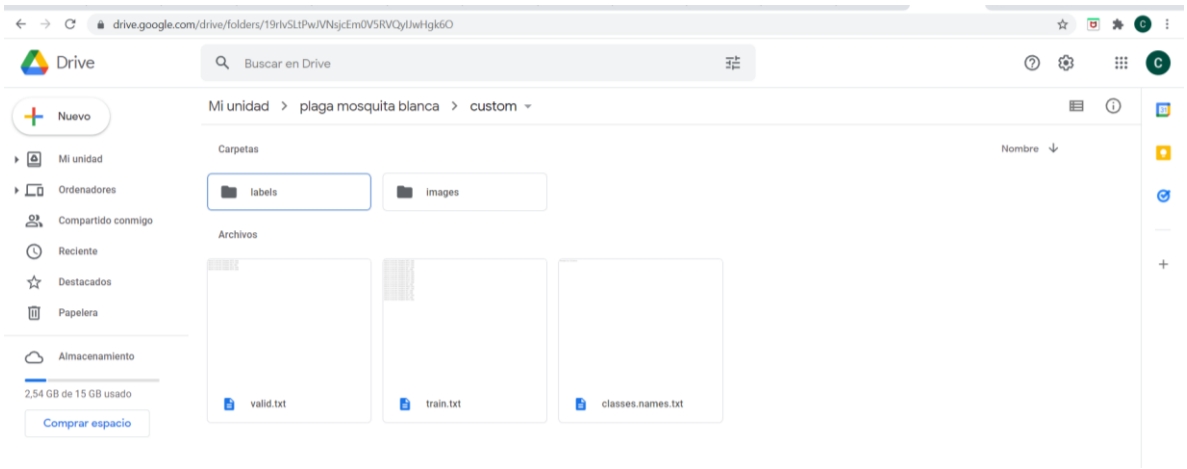

Figura 16.- Archivos subidos a drive.

La primera dirección es el de la carpeta de los archivos en drive y la segunda es la carpeta de destino donde se descargarán los archivos, figura 17.

Recomiendo borrar primero los archivos que vamos a sustituir que se encuentran por defecto en la carpeta que usamos de base, esto porque si se combinan los dos, traerá problemas al momento de hacer el entrenamiento debido a que son dos data set diferentes.

Debemos revisar si los archivos se pasaron correctamente para evitar errores

Por ultimo debemos ingresar a la siguiente dirección para poder cambiar un comando de un programa de ejecución del entrenamiento.

/usr/local/lib/python3.7/dist-packages/torchvision/transforms/functional.py

Debemos de cambiar la línea 5 de:

from PIL import Image, ImageOps, ImageEnhance, PILLOW_VERSION

a

```
from PIL import Image, ImageOps, ImageEnhance, __version__ as
PILLOW_VERSION
```

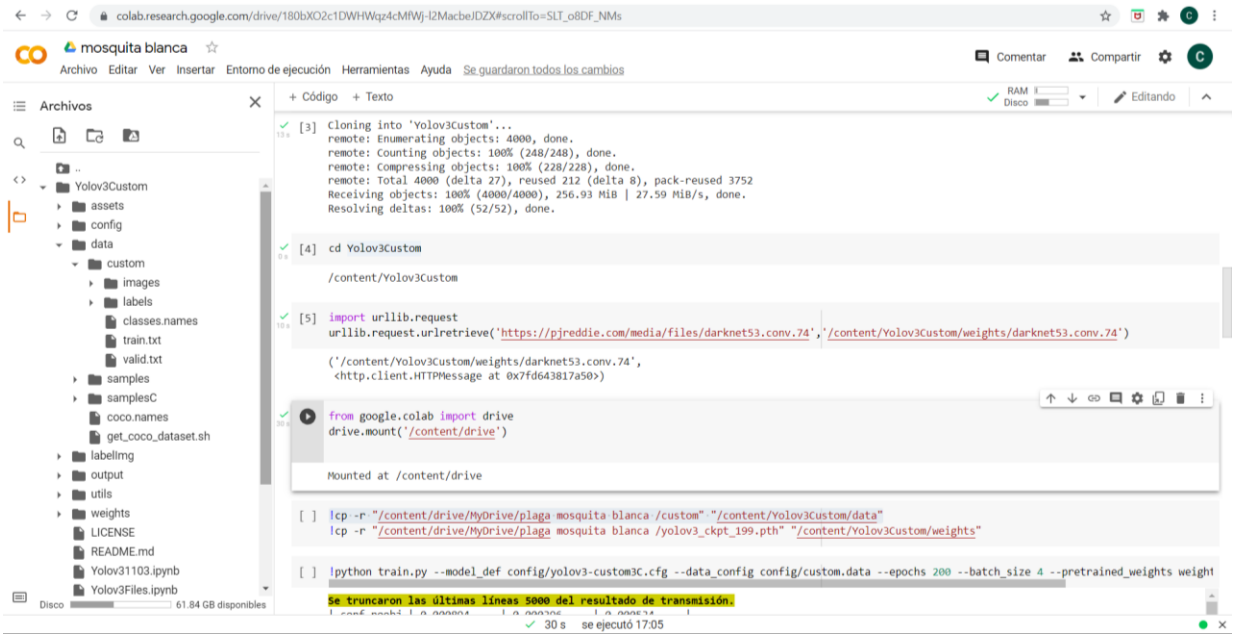

Figura 17.- Comandos y carpetas que se utilizan en el entorno.

Debemos guardar los cambios y regresamos a la carpeta donde están nuestro data set, este cambio es debido a que las nuevas versiones de Python pueden ocasionar problemas de compatibilidad, figura 18.

Por último, el comando para el entrenamiento será ale siguiente:

```
!python train.py --model_def config/yolov3-custom3C.cfg --data_config
config/custom.data --epochs 200 --batch_size 4 --pretrained_weights
weights/darknet53.conv.74
```

Si el entorno cuenta con una GPU, debemos modificar las épocas debido a que la memoria del entorno se llenara más rápido (poner 80 en lugar de 200) si el caso es contrario **podremos poner incluso hasta 250 épocas** de entrenamiento.

Si todo resulta bien el entrenamiento progresará hasta llegar a las épocas que deseamos, de tener algún error de compatibilidad en el data set nos arrojara errores después de tratar de entrenar la primera época, o después de 5 por lo regular, esto se debe a que hay un error de compatibilidad del data set por lo tanto hay que estar haciendo pruebas sacando imágenes y volviendo a generar los respectivos archivos de train y valid.

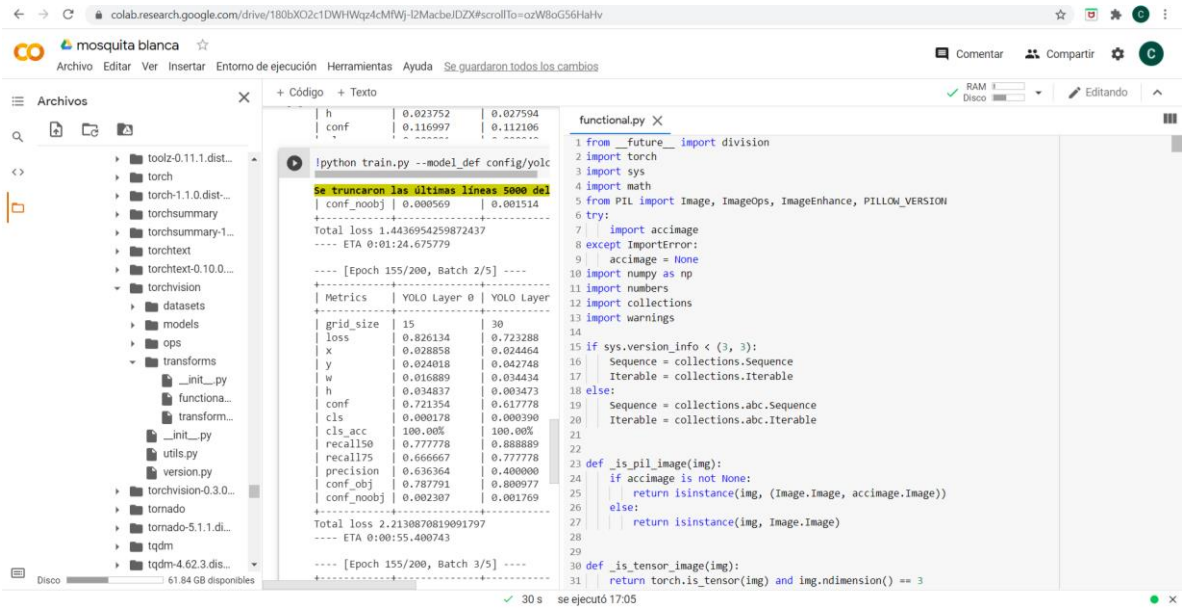

Figura 18.- Archivo a modificar.

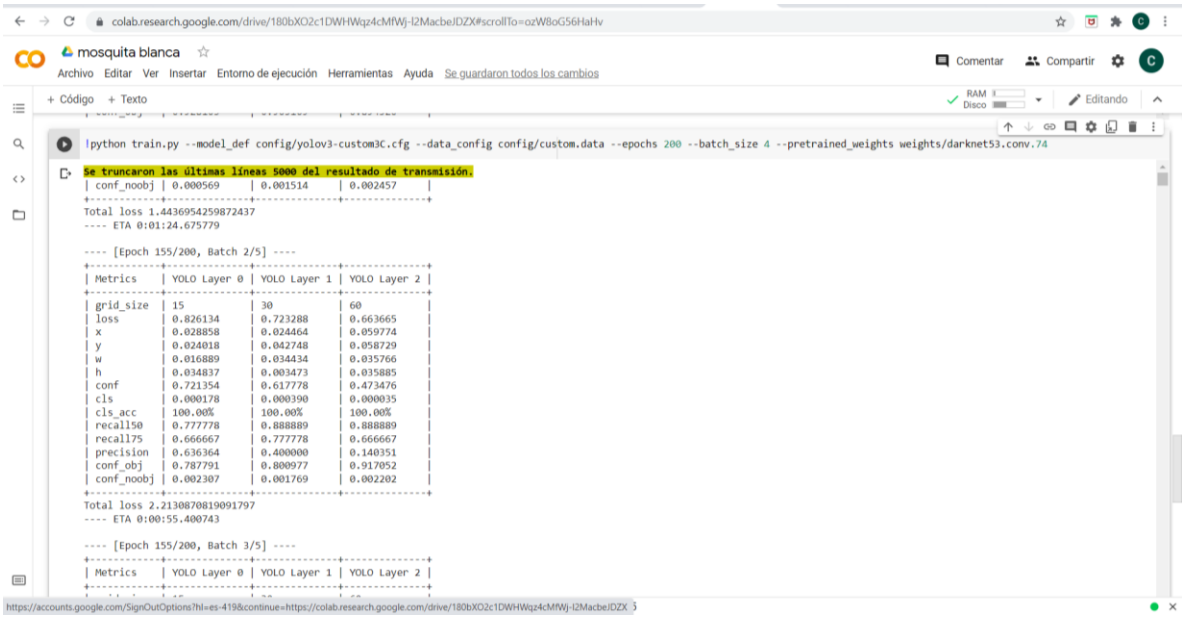

Figura 19.- Demostración de los datos arrojados por el entrenamiento.

Las épocas entrenadas se deberán de estar guardando en la parte inferior dentro de la carpeta general, ya que necesitaremos volver a entrenar pero ahora desde una época en específico, debemos descargar esos archivos y subirlos al drive para posteriormente introducir el siguiente comando:

!cp -r "/content/drive/MyDrive/plaga mosquita blanca /yolov3_ckpt_199.pth" "/content/Yolov3Custom/weights"

Nota: Hay que hacer otro entorno con las mismas características debido a que el que tenemos no contará con la suficiente memoria para hacer el reentrenamiento.

El comando anterior nos permite ingresar los pesos previamente entrenados en el otro entorno, con el objetivo de seguir desde ese punto, figura 20.

Comando de reentrenamiento:

```
!python train.py --model_def config/yolov3-custom3C.cfg --data_config
config/custom.data --epochs 200 --batch_size 4 --pretrained_weights
weights/yolov3_ckpt_199.pth
```

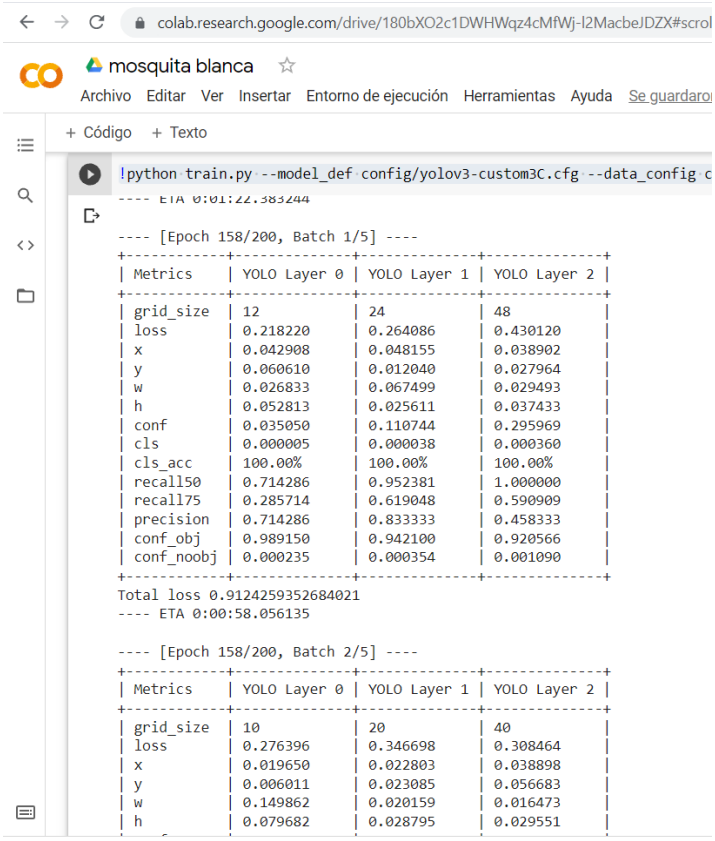

Figura 20.- proceso de reentrenamiento.

Para nuestro modelo de detección de plagas, procedimos a entrenar durante 400 a 600 épocas obteniendo un resultado aceptable en la ejecución.

Los archivos finales con las épocas que queremos probar los descargaremos y tendremos que crear una nueva carpeta en la carpeta de nuestro equipo la cual tendrá el nombre de *checkpoints*, en esta se guardaran los archivos de extensión pth, como se ve en la figura 21.

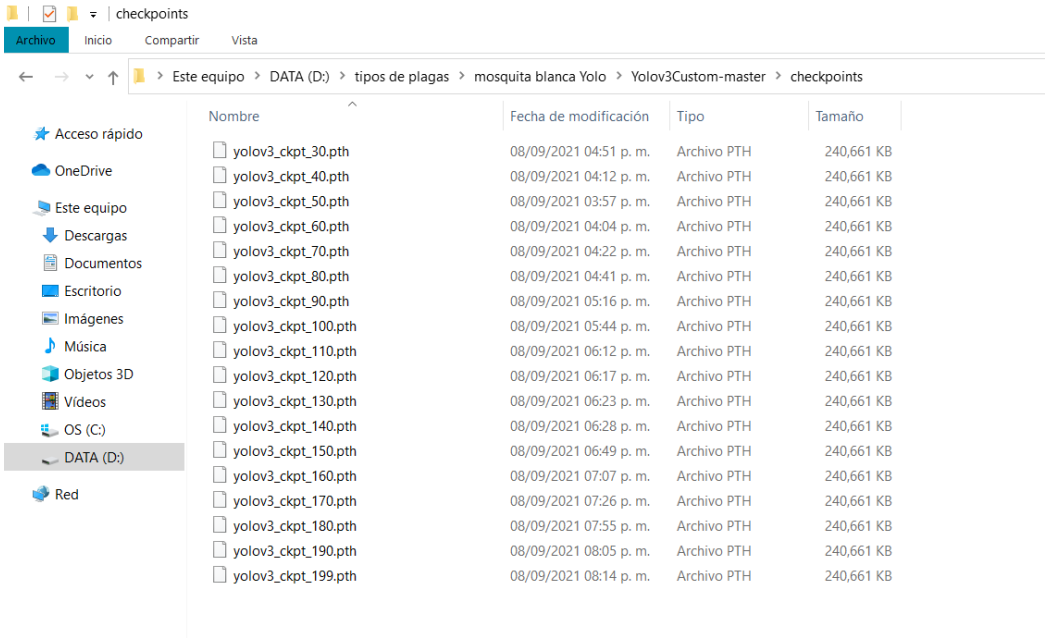

Figura 21.- Archivos del entrenamiento guardados en la carpeta checkpoints.

Paso 5 Ejecución del modelo

Para la ejecución debemos de reiniciar el entrono de anaconda, cerrando y volviendo a iniciar el ambiente creado, volvemos acceder a la carpeta.

Antes de pasar a la activación del modelo debemos ir a data/samplesC aquí es donde introduciremos las imágenes que va a evaluar, recomiendo introducir imágenes del entrenamiento para comprobar el desempeño.

Introducimos el siguiente comando en el entorno de anaconda:

```
python detectC.py --image_folder data/samplesC/ --model_def config/yolov3-custom3C.cfg --weights_path checkpoints/yolov3_ckpt_199.pth --class_path data/custom/classes.names
```

Hay que comprobar que el número de la época coincida con el archivo a evaluar en la carpeta de checkpoints.

Lo que nos dice este comando es que se tomaran las imágenes de la carpeta data/samplesC, usara el archivo de número de clases encontrado en config/yolov3-custom3C.cfg (este se ejecuta para una o tres clases diferentes), por último se usara el archivo con los pesos de entrenamiento de la carpeta checkpoints y el archivo con el nombre de la clase.

Una vez ejecutado procederá a entrar a la carpeta de simplesC y verificar si detecta los objetos o no, analizara cada imagen y dará información si es que detecto algo en la imagen así como el nivel en que está seguro que es el objeto coincide con el de la clase, figura 22.

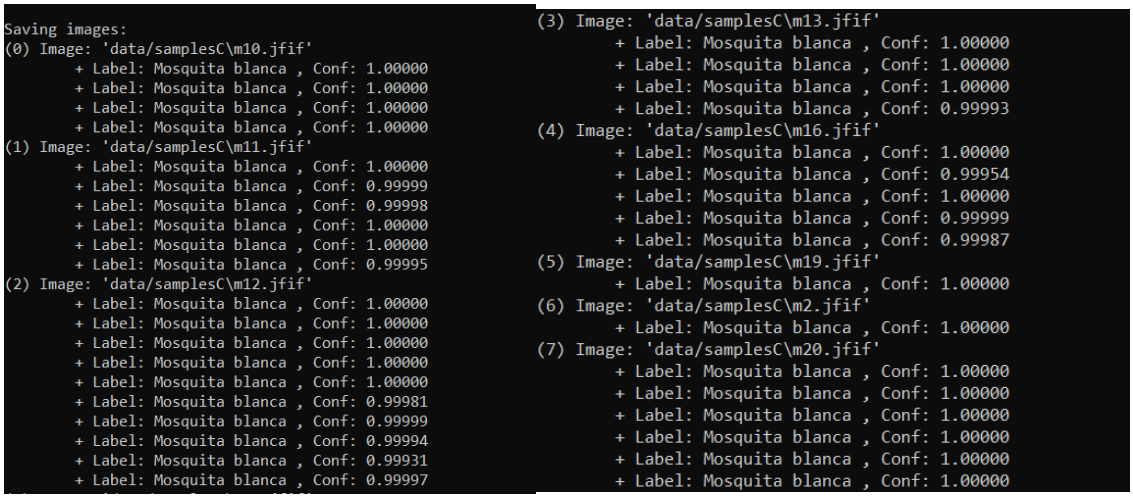

Figura 22.- Datos que arroja el modelo en ejecución.

Para checar las imágenes de salida debemos dirigirnos a la carpeta de output/ samplesC.

Podemos ver que es lo que detecto el modelo y si coincide con lo arrojado en la consola de comandos, en nuestro caso nos arroja las imágenes de la figura 23.

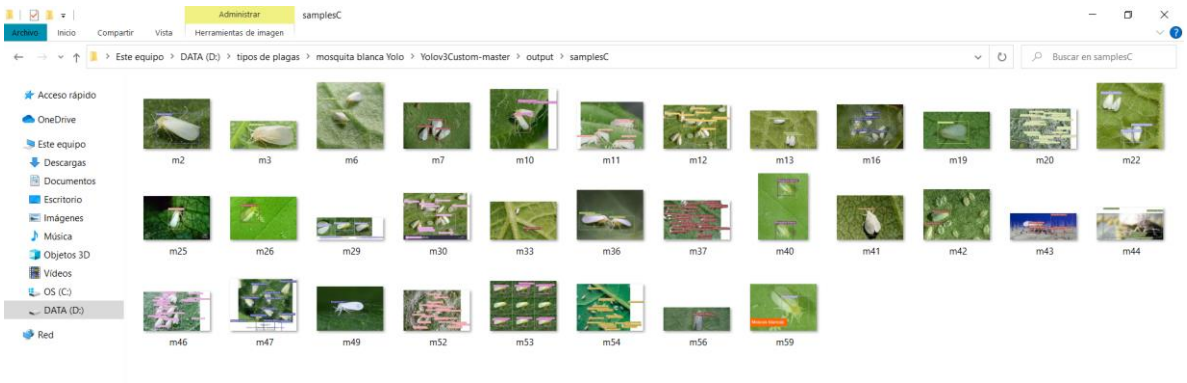

Figura 23.- Carpeta con las imágenes de salida.

En la figura 24, se muestra el etiquetado correcto de la plaga Mosquita blanca dentro las imágenes evaluadas, lo que nos indica que el reconocimiento de la plaga se está llevando de forma adecuada.

Figura 24.- Representación del antes y después de la ejecución del modelo usando Yolo.

Si queremos hacer otra prueba simplemente borramos las imágenes de la capeta de salida y volvemos a ejecutar el código, en este caso se realizará la prueba con una plaga distinta, Trips, tal como se puede ver en la figura 25.

Figura 25.- Ilustraciones del segundo modelo entrenado para la plaga trips.

Podemos observar que en el segundo modelo puede llegar a marcar doble vez el objetivo esto dependerá de la ejecución y épocas en que se entrenó.

Recomendaciones para evitar complicaciones

Hay algunas complicaciones al momento de hacer el entrenamiento las cuales son las siguientes:

1. Algunas imágenes son incompatibles o no las reconoce, se recomienda hacer un data set de 20 imágenes e irlos probando si se logra entrenar bien el modelo por lo menos 50 épocas, si procede a un error sería cuestión de descartar imágenes de preferencia las de menor calidad o con mucha interferencia con el objeto en cuestión.
2. El entrenamiento tarda mucho tiempo en realizar, 200 épocas en aproximadamente 8 a 10 horas, si la conexión a internet es estable, por lo cual se recomienda dejarlo y checar por ratos el avance, además de no hacer otra tarea en la computadora si no puede llegar a trabarse o cancelarse la ejecución.
3. Tratar de entrenar por lo menos 400 épocas del modelo ya que así se garantiza un mejor rendimiento, aunque es preferible probar cada 50 para **evitar sobre entrenamiento.**

Esos son algunos errores o cuestiones que se notaron en el desarrollo de este proyecto, puede haber más como por ejemplo no introducir un código adecuadamente, saltarse un paso etc., en esos casos recomiendo volver a empezar en otro entorno con más calma.

Beneficio del desarrollo de este modelo de detección

Este tipo de herramienta nos permite aplicarla en diferentes áreas de conocimiento, son esquemas cada vez más aceptados ya que se puede generar sistema de apoyo sin contar con una gran experiencia o mucho equipo. Además de que solo se necesita obtener la imagen a analizar y un equipo de cómputo.

Para fines de este proyecto, la identificación de plagas resulta beneficioso ya que puede aplicarse a cualquier tipo de cultivo afectado por diferentes plagas y en consecuencia se puede tomar la decisión más acertada de cómo combatirla.

Saber que plaga es y actuar en consecuencia, implica la posibilidad de si podemos salvar la planta o la perdemos por completo, así como si la propagación será rápida o lenta en caso de ser más de una planta, en el caso de las dos plagas analizadas estas suelen extenderse rápidamente y a veces es difícil lidiar con ellas debido a esta característica.

Conclusiones

Podemos determinar que los modelos de detección de objetos tienen un alto potencial hoy en día no solo en aplicaciones industriales, sino también en aplicaciones relacionadas con actividades cotidianas, o en este caso, en los cultivos, en los que podemos evitar problemas de producción agrícola identificando a tiempo las diferentes plagas que pueden afectar una planta.

El modelo YOLO que se utilizó representa una ventaja e innovación frente a otros modelos existentes ya que este prioriza el tiempo de ejecución además de que posibilita un mejor funcionamiento debido al uso de redes mejor adaptadas.

Personalmente, y como estudiante, se puede ver lo amigable que es este tipo de modelo en cuanto al equipo y conocimientos necesarios para poder ejecutarlo, uno como ingeniero centrado en las áreas de mecánica y eléctrica a veces no cuenta con conocimientos más allá de los básicos en programación, por ende **este tipo de sistemas puede llegar a ser aterrador al armarlos, pero con las ventajas que ofrece hoy en día la tecnología y que cada vez más se conoce el uso de la inteligencia artificial, uno puede adaptarse y proseguir sin tantas trabas y de manera sencilla en el camino para desarrollar proyectos de alto nivel**.

Por último, observamos que este modelo de detección de plagas tipo insecto elaborado tiene una confiabilidad aceptable y su entorno de ejecución no es tan demandante como algunos otros similares, pudiéndose ejecutar en equipos sencillos y portátiles.