Proyecto N°2 Redes Neuronales Artificiales

Carlos Doffiny S-V, Universidad Autónoma de Occidente
carlos.doffiny@uao.edu.co
Caracas - Venezuela

***Resumen* – En el presente documento se llevará a cabo la explicación de cada uno de los procedimientos y pasos seguidos durante el estudio, aplicación, entrenamiento y validación de una red neuronal capaz de detectar objetos de cinco (5) categorías pertenecientes a un data set de autoría propia, y aplicando una arquitectura de red Fast R-CNN. Durante el desarrollo de lo anteriormente diseñado, se partió de las bases dictadas en las categorías, en apoyo con diferentes herramientas como Google Colab, Roboflow, LabelImg y librerías de redes como Tenserflow y Detectron2.**

Índices – Capas, Data set, Detección de objetos, Detectrron2 Entrenamiento, Fast R-CNN, Imagen, Pesos sinápticos, Red Convolucional, Roboflow, Neurona

I. INTRODUCCIÓN

Las redes neuronales artificiales son muy abstractas en su concepción, pero muy útiles cuando las mismas son diseñadas, entrenadas y validadas correctamente. Sus aplicaciones son infinitas y sus variantes lo son aún más, por lo cual llegar a entenderlas y recrearlas no es sencillo, y requiere de mucho estudio, conocimiento, entendimiento del problema y práctica del mismo.

En el caso de la presente investigación, es necesario aplicar todos los conocimientos adquiridos con la final de diseñar, entrenar y validar una red neuronal que permita detectar objetos a través de una arquitectura de tipo Faster R-CNN, que dado un data set de elaboración propia que consta de 150 imágenes de carros, aviones, gatos, pingüinos y tanque, sea lo más efectiva y certera posible detectando estas categorías y/o objetos en las imágenes que se le pasen a esta red.

Para lograr el objetivo es necesario primero obtener la data del dataset, preprocesarla para unificar criterios de tamaño, y aplicarle data augmentation con el fin de evitar el sobre entrenamiento, al ser un data set pequeño, luego esta data se debe dividir en data de entrenamiento, validación y prueba para luego utilizar arquitecturas como la Faster R-CNN en su versión 101, y que por ende al ejecutarse ya cuenta con los pesos sinápticos entrenados para la detección de objetos, obteniendo así luego del entrenamiento, una predicción en cada imagen sobre la detección o no de cualquiera de los cinco objetos que contiene el data set creado, indicando así el nombre del objeto encontrado, y encerrando a este en una caja.

Sin más nada que agregar, están cordialmente invitados a comprender paso a paso los procedimientos realizados durante la construcción de la red neuronal.

II. MARCO TEÓRICO

Capas: Conjunto de neuronas que comparten una misma función de activación, que reciben las mismas entradas, y que se dirigen hacia la misma salida.

Data set: Recopilación en masa de datos que pueden estar clasificados o no.

Detección de objetos: Es una tecnología que permite identificar y nombrar (y por ende detectar) cualquier tipo de objeto con el cual una red convolucional haya sido entrenada para detectar, arrojando el nombre del mismo y su ubicación en la imagen o vídeo utilizado.

Detectron2: Es un framework basado e implementado en Pytorch, y que está repleto de herramientas muy útiles que permiten construir, entrenar y validar modelos de redes neuronales enfocadas en la detección de objetos.

Entrenamiento: Proceso a través del cual una red, a partir de una serie de datos de entrada, es entrenada para generar salidas específicas. Puede ser supervisado o no supervisado, dependiendo de durante el entrenamiento es conocido o no el valor esperado de la salida.

Faster R-CNN: Es una arquitectura basada en una red convolucional basada en regiones, por lo cual en este caso la imagen se divide en muchas regiones simétricas que luego se van analizando una a una con la red convolucional con el fin de detectar posibles objetos en cada una de estas mini regiones. Lo que diferencia a este tipo de arquitectura a las demás arquitecturas R-CNN, es que estas últimas son más lentas ya que primero extraen todas las regiones y luego las van pasando por las capas convolucionales de la red, mientras que la versión Faster R-CNN (como su nombre lo indica) es mucho más rápida al pasar directamente la imagen a la capa convolucional, siendo esta capa la que genera las regiones de interés. Sin embargo, no es la arquitectura más rápida de

Imagen: Consiste en una representación gráfica y visual de un objeto o de un conjunto de ellos, el cual está construido por una matriz de pixeles con valores que van de 0 a 255 en cada posición. Pueden ser de tres canales (que tienen colores RGB) o de un solo canal (en blanco y negro).

Red Convolucional: A diferencia de las redes estudiadas anteriormente en las que solo procesaban información y adaptaban pesos, este tipo de redes está conformadas por múltiples capas de filtros convolucionales de una o más dimensiones, con las cuales van extrayendo información y diferentes características de la data ingresada, lo que la hace perfecta para el procesamiento de imágenes y vídeos.

Este trabajo fue apoyado por los docentes Jesús Lopez y Andrés Escobar, docentes de la asignatura de Redes Neuronales Artificiales y Deep Learning en la Universidad Autónoma de Occidente, Al igual que por los diferentes frameworks y librerías utilizadas como Roboflow, Detectron2 y Tensorflow.

Roboflow: Es un framework construido para la elaboración, manejo y preprocesamiento de data sets, así como también cuenta con diferentes modelos y arquitecturas de redes neuronales para la detección y segmentación de objetos.
Neurona: Unidad estructural de una red, encargada de procesar la data recibida a partir de su peso sináptico, peso sináptico y bias.

III. DESCRIPCIÓN DEL PROBLEMA A SOLUCIONAR

El problema planteado corresponde que a partir de un data set de autoría propia [1] conformado por imágenes tipo jpg pertenecientes a cinco (5) categorías (carro, avión, gato, pingüino y tanque), en la cual cada categoría posee unas treinta (30) imágenes, se cree de una red neuronal que realice la detección de dichos objetos indicados en cada categoría, en cualquier imagen que se le ingrese a la red. Pero, además, se deberá usar cualquier arquitectura de red convolucional basada en regiones (R-CNN), con la finalidad de automatizar, facilitar y agilizar el proceso de detección de imágenes, utilizando y aplicando arquitecturas que ya están previamente entrenadas, por lo cual poseen sus pesos sinápticos ajustados a las necesidades que se tienen, así como para los objetivos planteados en el presente proyecto.

Cabe destacar que para la creación del data set, las categorías o clases fueron escogidas al azar, pero la selección de cada una de las imágenes si fue pensada en obtener la mayor diversidad posible de perspectivas del objeto en sí, con la finalidad de que la red aprendiera las características únicas que identifican a cada objeto, y no colores, posiciones o fondos que a pesar que forman parte del objeto y su entorno, no son características intrínsecas y/o vitales del mismo.

IV. PLANTEAMIENTO DE LA SOLUCIÓN

Para encontrar la solución al problema descrito anteriormente se plantea lo siguiente:

A. Etiquetado del Data Set

Una vez encontradas y guardadas todas las imágenes seleccionadas entre las cinco (5) categorías que conforman el data set, es necesario etiquetar los objetos que se encuentran en cada una de ellas. Y para ello, se puede utilizar la herramienta de Label Img [2] o también puede hacerse en el mismo framework de Roboflow [3].

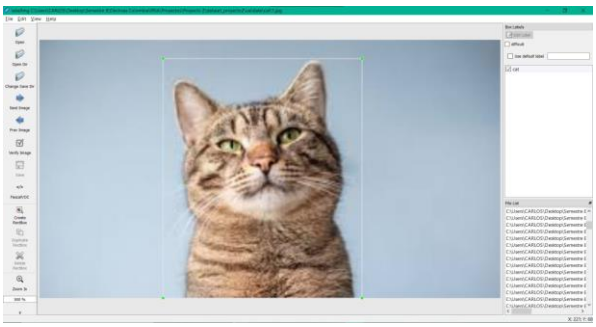

Fig. 1. Etiquetado con Label Img.

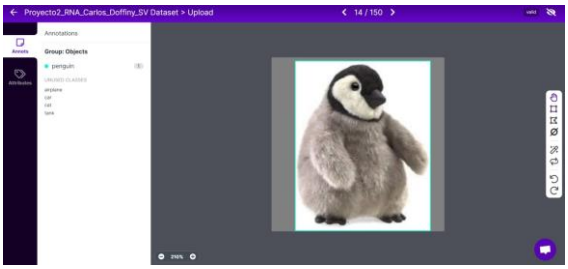

Fig. 2. Etiquetado con Roboflow.

B. División de la data

El data set original de ciento cincuenta (150) imágenes, se dividirá en los siguientes tres (3) subgrupos:

* Datos de entrenamiento (70% - 105 imágenes)
* Datos de validación (20% - 30 imágenes)
* Datos de prueba (10% - 15 imágenes)

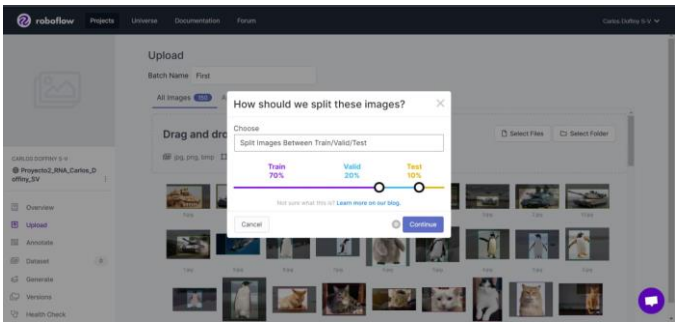

Fig. 3. División de las imágenes del data set.

C. Preprocesamiento del data set y aplicación de data augmentation

Antes de utilizar el data set, es necesario aplicar un preprocesamiento en donde cada una de las imágenes sufrirá un resize de 416x416 pixeles, ya que para el procesamiento de estas, lo más útil es que la imagen sea una matriz cuadrada que es más sencillo de filtrar y extraer características que en una imagen con el ancho y el alto de diferentes dimensiones.

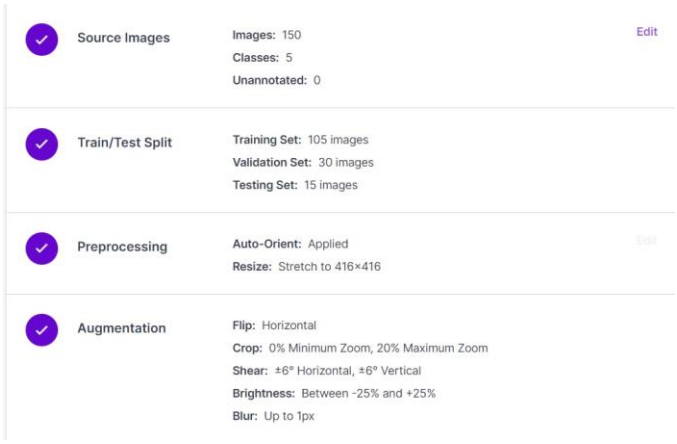

Fig. 4. Preprocesamiento y data augmentation del data set.

Así mismo, el data set original es realmente pequeño, comparado con la cantidad de imágenes que suelen tener los

data sets más grandes, que poseen cientos y hasta miles de imágenes. Por ello, al ser pequeño, la red puede sufrir sobre entrenamiento, y para evitar esto se aplica una técnica conocida como data augmentation, en donde a partir de las imágenes de entrenamiento, se crearán otras que tengan diferentes perspectivas a las originales, como se indica en la Fig 4, en donde se observa que por cada una de las 105 imágenes de entrenamiento se crearán dos nuevas imágenes que pueden estar invertidas horizontalmente, con un zoom del 20%, un brillo de +-25%, una rotación de +-6 grados y un difuminado de 1px. Estas 210 imágenes nuevas ayudarán a que el data set sea mucho más completo al tener un mayor espectro de pruebas que además serán con diferentes perspectivas, lo que promoverá a la red a enfocarse en aprender las características intrínsecas y naturales de cada objeto, y no colores, posiciones, figuras o fondos.

La data augmentation aplicada únicamente a las imágenes de entrenamiento, dejaría al data set con un total de 360 imágenes distribuidas de la siguiente forma:

* Datos de entrenamiento (88% - 315 imágenes)
* Datos de validación (8% - 30 imágenes)
* Datos de prueba (4% - 15 imágenes)

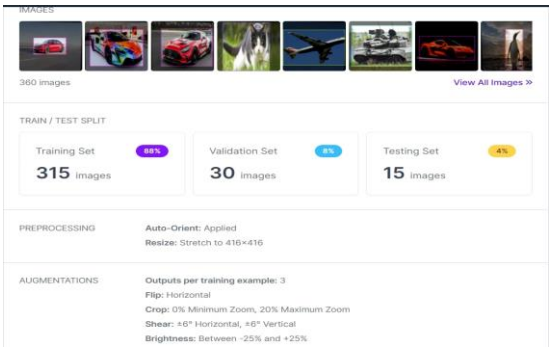

Fig. 5. Distribución final del data set.

D. Conectar una GPU

Las redes convolucionales son redes mucho más avanzadas que las trabajadas anteriormente en el curso, y que a su vez requieren de una fuerza de computo mayor, ya que al contar con más capas, más funciones de activación, más neuronas y demás elementos de una red, además de cientos de features maps, se requieren realizar un enorme número de operaciones que no serían posibles sin la ayuda de un GPU, por lo cual sin este, simplemente no sería posible el entrenamiento y validación de la red entrenada en el presente proyecto.

En esta ocasión, al contar con el plan gratuito de Colab, solo se cuenta con un máximo de 8 horas seguidas de uso de la GPU cada 24 horas, siendo esta última una Tesla K80.

Fig. 6. Verificando el uso de una GPU Tesla K80.

E. Imports

Fig. 7. Descargando las dependencias necesarias para Detectron2.

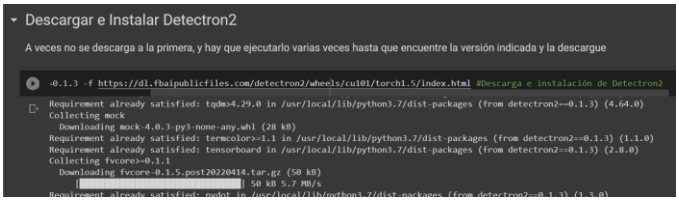

Fig. 8. Descargando Detectron2.

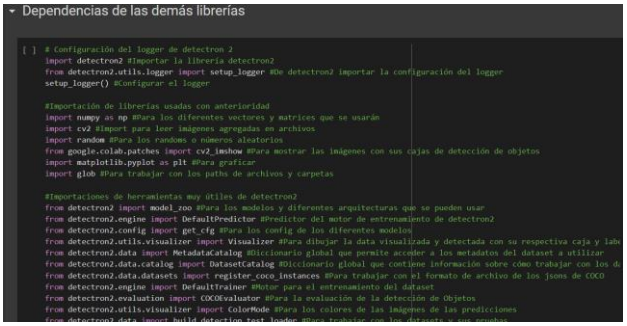

Fig. 9. Descargando las dependencias y librerías restantes.

Ahora llegó la hora de descargar todas las dependencias, librerías y herramientas necesarias para la red que se desea entrenar. Como se indicó anteriormente, la misma usará el framework de Detectron2 [4], el cual contiene un gran número de herramientas muy útiles para el entrenamiento, validación y prueba de la red de detección de objetos. Esto se puede observar en la “Fig 7”, en donde es necesario descargar e instalar primero dependencias como Pytorch y Pyyaml antes de descargar e instalar Detectron2 en la “Fig 8”.

Por último, en la “Fig 9” se descargan el resto de librerías y dependencias necesarias para la red neuronal.

F. Importar y descargar el data set

Una vez descargadas e instaladas todas las librerías y dependencias necesarias, es hora de exportar el data set creado y trabajado en Roboflow, el cual nos dará una línea de código que servirá para importar y descargar el data set en el Colab, cuyas carpetas e imágenes se alojarán en los archivos de la ventana abierta actual de Colab.

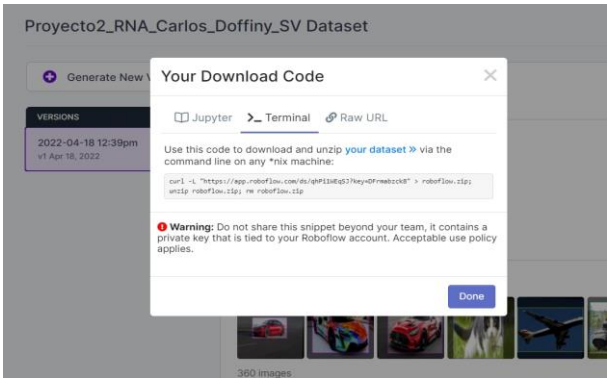

Fig 10. Exportar el data set desde Roboflow.

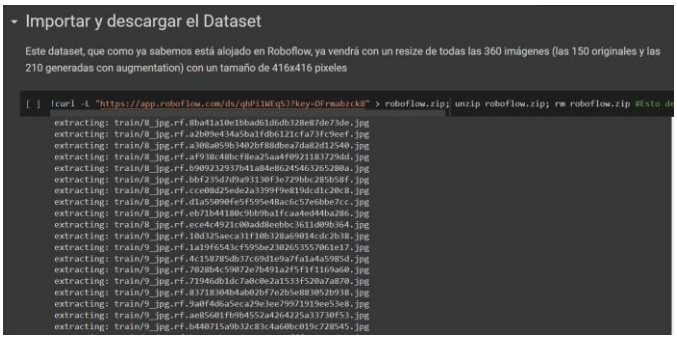

Fig 11. Importar y descargar el data set.

Así mismo, es necesario que una vez se descargue todo el data set, registrar cada grupo de datos (entrenamiento, validación y prueba) en instancias COCO en detectron2. Cabe destacar que COCO es uno de los data sets de imágenes más grandes usados en la detección y segmentación de objetos, y con él es que trabaja detectron2, por lo cual las imágenes que estamos usando, deben registrarse ahí también.

Fig 12. Identificación y registro de los diferentes grupos de datos de imágenes.

G. Visualización de las imágenes de entrenamiento

Una vez descargado el data set, se quiere verificar si las imágenes de entrenamiento se visualizan directamente, por lo cual se buscan aleatoriamente 8 imágenes de estas junto a sus metadatos, para mostrarlas en el Colab, y así corroborar que

toda la descarga se hizo perfectamente y que hay imágenes para cada una de las clases o categorías indicadas en el data set, con lo cual se estaría ya listo para iniciar con la configuración del entrenamiento de dichas imágenes.

Fig. 13. Visualización de las imágenes de entrenamiento.

H. Entrenamiento de la red

Antes de iniciar con el entrenamiento de la red, es necesario configurar los directorios que se usarán en el mismo, así como importar el módulo de entrenamiento de COCO y Detectron2.

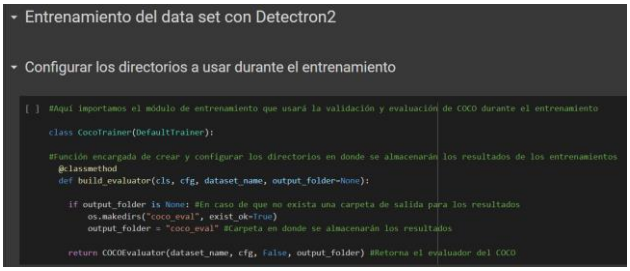

Fig. 14. Configurando los directorios a usar e importando el módulo de entrenamiento.

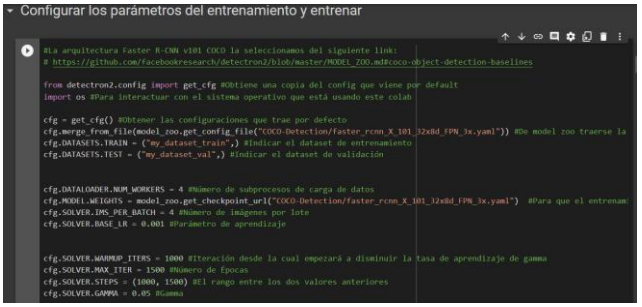

Fig. 15. Primera parte de la configuración de los parámetros de entrenamiento.

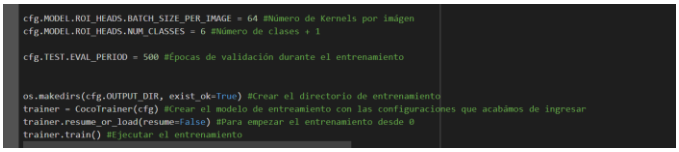

Fig. 16. Segunda parte de la configuración de los parámetros de entrenamiento.

A su vez, también es necesario configurar los parámetros del entrenamiento, los cuales se indican y modifican con archivo config que viene por defecto en el módulo de entrenamiento de Detectron2. En él se indica al iniciar, cuál será la arquitectura de la red, siendo esta una Faster R-CNN v101-32x8d COCO [5]. Esta arquitectura [6] está compuesta por una capa de una red convolucional que es la que genera los feature maps sobre los cuales se van a generar las regiones de interés con anchor boxes, y luego es que esas regiones propuestas se unen a los otros feature

maps, en el RoI Pooling, para pasar a una fracción de la red de capas Fully Connected, para finalizar con un box prediction y su % de clasificación del label.

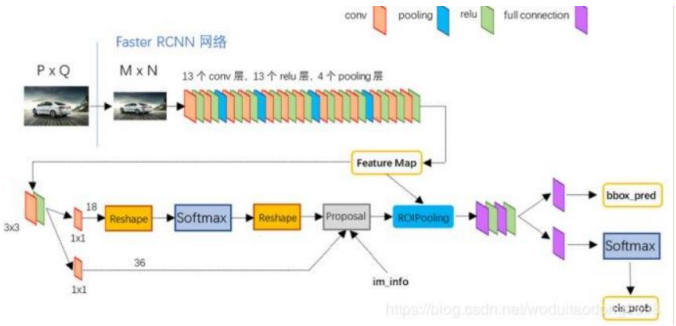

Fig. 17. Arquitectura Faster R-CNN.

Posteriormente de seleccionada la arquitectura de la red y los pesos que usará la misma (que son los que la arquitectura Faster R-CNN ya tiene entrenados y ajustados) se proceden a ingresar diferentes parámetros que se observan en las “Fig 15” y “Fig 16” como lo son:

* Indicar el data set de entrenamiento a usar (my_dataset_train).
* Indicar el data set de validación a usar (my_dataset_val).
* Número de imágenes por lote (4 imágenes por lote).
* Parámetro de aprendizaje base (0.001).
* Número de épocas del entrenamiento (1500 épocas).
* Número de regiones de interés por imagen (64 ROI).
* Número de clases (5 + 1 por default, 6 en total).
* Número de épocas de validación (500 épocas).

Posterior a la configuración de todos los parámetros, es que se ordena ejecutar el entrenamiento de la red, el cual imprime la arquitectura capa por capa de la red, el resultado del entrenamiento con un checkpoint cada 20 épocas, así como va mostrando el progreso de la red cada 500 épocas.

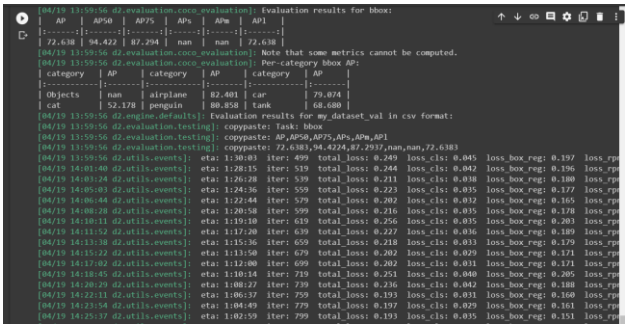

Fig. 18. Probando diferentes optimizadores.

I. Evaluación del modelo entrenado

Una vez finalizado el entrenamiento de la red según los parámetros ingresados en las “Fig 15” y “Fig 16”, se procede a evaluar la misma con las 15 imágenes de prueba que posee el data set. Pero para ello, primero se deben almacenar los nuevos pesos sinápticos obtenidos durante los ajustes de cada época de entrenamiento, así como indicar el umbral de rechazo, y configurar el modelo de validación de la data que ofrece detectron2.

Una vez finalizada la evaluación del modelo, esta nos indicará el porcentaje de éxito reconociendo cada una de las 5 categorías del dataset, que se encuentran dentro del grupo de datos de prueba.

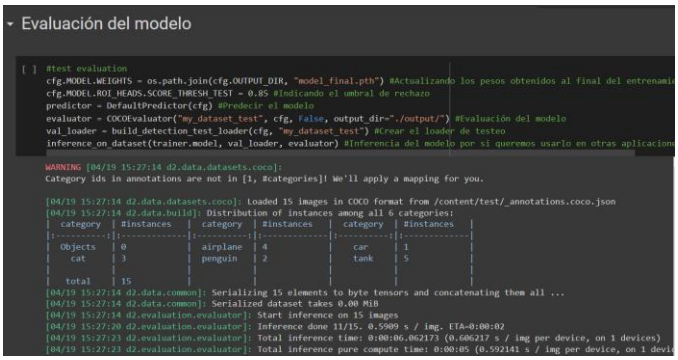

Fig. 19. Evaluación del modelo entrenado.

J. Predicción del modelo

En este punto, solo queda construir y ejecutar la predicción del modelo, que es muy similar al punto anterior, con la importante diferencia que en esta ocasión se usará un umbral menor, y se obtendrán en vez de los porcentajes de aciertos de cada clase, las imágenes con sus respectivos metadatos, por lo cual se podrá apreciar la detección de los objetos, con sus cajas, labels y porcentaje de similitud con la categoría indicada.

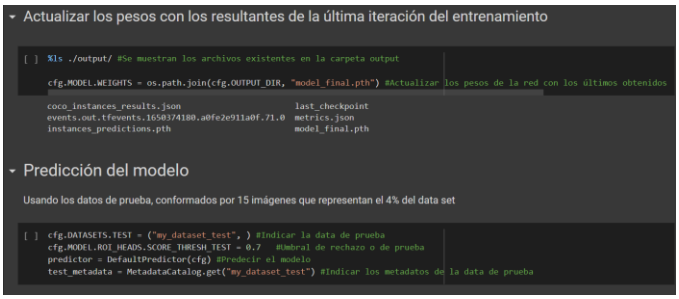

Fig. 20. Predicción del modelo con los datos de prueba.

Fig. 21. Predicción del modelo con las imágenes de prueba, mostrando los objetos detectados.

Luego de obtenidas las predicciones, como se puede apreciar en la “Fig 21”, se procede a observar el resultado en cada una de las 15 imágenes que contiene el data set para pruebas.

K. Probar la red con cualquier imagen

Ya para finalizar el planteamiento de la solución al problema, se desea probar la red con varias imágenes no pertenecientes a las originales del data set, con la finalidad de realizar diferentes pruebas que representen un reto para la red, y que de ser exitosas, demostraría la eficiencia del modelo entrenado.

Los ejemplos que se buscarán analizar son los siguientes:

* Combinación de objetos entrenados.
* Objeto diferente a los entrenados.
* Objeto igual a los entrenados.
* Múltiples objetos de los entrenados en la misma imagen.
* Imagen libre indicada por el profesor.

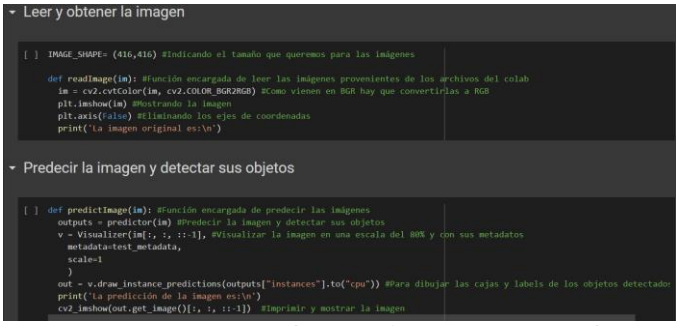

Fig. 22. Funciones que se usarán en los diferentes ejemplos de imágenes no pertenecientes al data set original, con las cuales se probará el modelo.

V. RESULTADOS

Luego de ejecutado al pie de la letra el planteamiento de la solución propuesto, se obtienen los siguientes resultados que ya se explicarán:

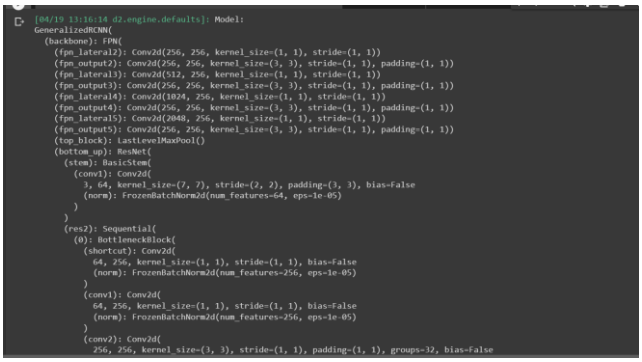

Fig. 23. Arquitectura del modelo resultante.

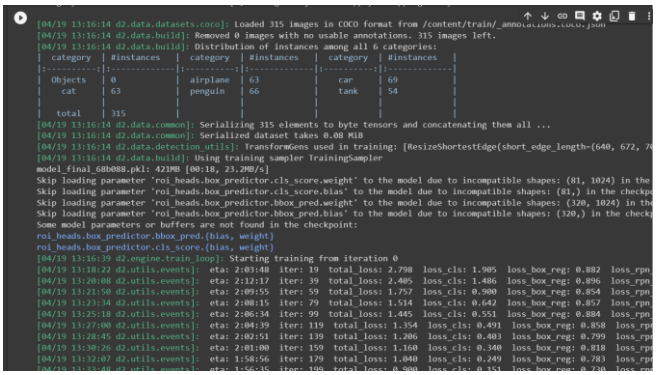

Fig. 24. Resumen de las imágenes por clase del data set de entrenamiento, y de los primeros checkpoints del entrenamiento.

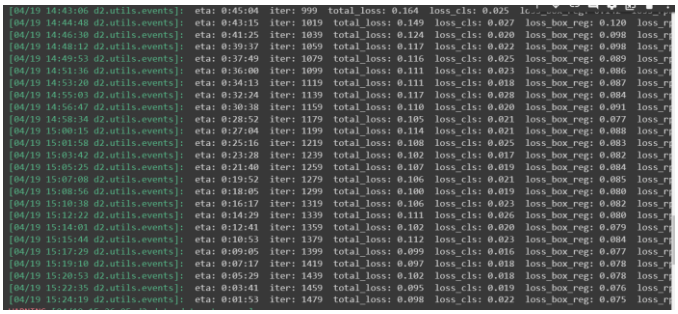

Fig. 25. Resultados de los últimos checkpoints del entrenamiento.

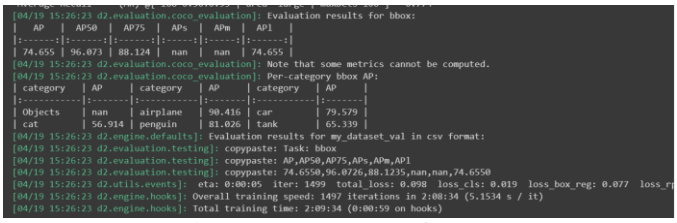

Fig. 26. Porcentaje final de acierto por categoría luego del entrenamiento.

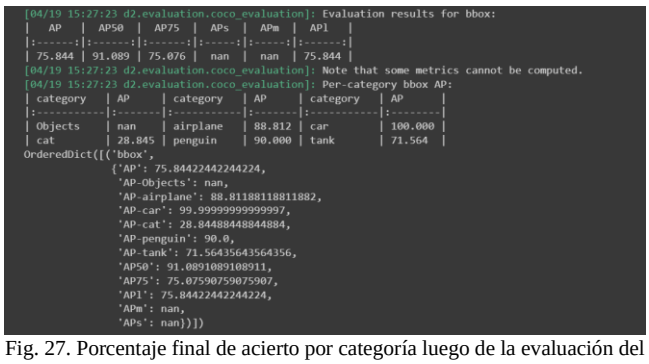

Fig. 27. Porcentaje final de acierto por categoría luego de la evaluación del modelo.

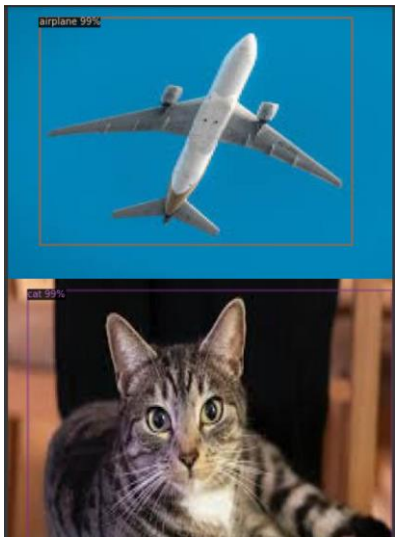

Fig. 28. Resultados de las imágenes pertenecientes al data set de prueba.

Fig. 29. Resultados de las imágenes no pertenecientes al data set original.

En la “Fig 23” se presenta la arquitectura por capas del modelo Faster R-CNN entrenado anteriormente y que se usó como arquitectura base para el modelo aplicado para solucionar el problema planteado. En él se indican las capas, tamaño de los kernels, strides, paddings, tamaño de las matrices de entrada y salida, entre otras cosas.

En la “Fig 24” se hace un resumen de la cantidad de imágenes que contiene cada categoría en el data set de entrenamiento, que consta de 315 imágenes. A su vez, luego de esto se inicia una impresión de los resultados que se obtiene en cada checkpoint, el cual ocurre cada 20 épocas, en donde indica el tiempo restante, el número de iteraciones por la cual se va, y los distintos valores de pérdida que presenta la red en ese punto.

En la “Fig 25” se observan los resultados mostrados en cada uno de los últimos checkpoints, mientras que en la “Fig 26” se visualizan los resultados finales del entrenamiento, con un 94.42% de acierto para los aviones, 79.58% para los carros, 56.61% para los gatos, 81.02% para los pingüinos y 65.34% para los tanques. Adicionalmente, también indica la pérdida final del modelo, la cual es de 0.098.

Posteriormente, en la “Fig 27” se presentan los resultados finales del modelo luego de su evaluación con las 15 imágenes de prueba. Y ya para finalizar, se visualizan estas imágenes de pruebas con los objetos identificados en ellas, como se observa en la “Fig 28”, para luego probar con imágenes no pertenecientes originalmente al data set, para comprobar y verificar la efectividad de la red con diferentes propuestas de imágenes, así como se aprecia en la “Fig 29”.

VI. CONCLUSIÓN

Luego de llevar a cabo el presente trabajo, en el cual sin duda alguna se ha aprendido bastante sobre redes neuronales convolucionales de detección de objetos, podemos concluir que:

El primer paso, y que sirve de base fundamental para la construcción del modelo de redes neuronales, es construir un data set que contenga la mayor cantidad posible de imágenes sobre las diferentes categorías o clases que se quieran detectar, así como aplicar diferentes técnicas de preprocesamiento que ayudarán al entrenamiento y posterior efectividad de la red, como un resize a todas las imágenes, así como aplicar data augmentation a las que formen parte del data set de entrenamiento con el objetivo de evitar problemas como el sobre entrenamiento, y ayudar a la red a trabajar con los objetos en muchas perspectivas diferentes que le permitan identificar las características propias e innatas de cada uno de ellos, y no colores, formas o fondos.

Posteriormente, no hace falta reinventar la rueda para lograr resolver el problema planteado, las redes convolucionales son redes que consumen muchísimos recursos de computo, y que tardan horas o días en entrenarse, además que requieren de data sets enormes para empezar a ser efectivas, es por ello que es muy útil en estos casos utilizar arquitecturas que ya tienen modelos y pesos sinápticos entrenados y ajustados a las necesidades de una red de detección de objetos, y que por ende, al usarla de base para el modelo a implementar, serán de una enorme ayuda. En este caso, se usó una arquitectura Faster R-CNN que ha sido entrenada y potenciada por grandes empresas como Facebook, y que son sumamente usadas alrededor del mundo, y que se implementa en el framework de detectron2, que es el que se utilizó para esta ocasión.

Adicionalmente, luego de mucho estudio, y de dejar la red entrenando unas cuantas horas, esta pudo aprender muy bien el arte de detectar carros, aviones, tanques, gatos y pingüinos. De hecho, en la evaluación del modelo con las imágenes del data set de prueba, detectó 14 de los 15 objetos, ya que hubo una imagen de un avión que no detectó, pero que analizando el data set y por ende las perspectivas de entendimiento de la red, al esta imagen ser la única de un avión sin alas, entonces seguramente no lo habrá identificado por esa razón.

Sin embargo, si se analizan los resultados de la detección de objetos en imágenes diferentes a las usadas al data set, vemos que en los 4 ejemplos implementados, en todos y cada uno de ellos se obtiene un resultado exitoso, lo que está muy bueno y habla excelente de la red entrenada.

Por último, se puede concluir que diseñar y construir un modelo y arquitectura de redes neuronales convolucionales que detecten objetos no es nada sencillo. Requiere de mucha investigación de arquitecturas y modelos, de horas recolectando imágenes y etiquetándolas, de ensayos, de

pruebas y error, de ir cambiando y testeando diferentes parámetros hasta ir encontrando soluciones que arrojen errores cada vez más cercanos al cero, y, por ende, cada vez más certeros. Y hacer todo eso sin ayuda de las diferentes librerías y herramientas mencionadas durante el presente trabajo, es mucho más complicado aún.

VII. AGRADECIMIENTOS

Especiales agradecimientos a los profesores de la asignatura de Redes Neuronales Artificiales y Deep Learning en la Universidad Autónoma de Occidente, los profesores Jesús López y Andrés Escobar, por sus enseñanzas y material de ayuda de gran utilidad para el presente trabajo. También agradecimientos a Roboflow y a todo su staff por construir y enseñar su increíble framework, que me fue de muchísima utilidad durante el presente proyecto.

VIII. REFERENCIAS

Páginas web:

[1] Carlos Doffiny S-V. (2022). Proyecto2_RNA_Carlos_Doffiny_SV Dataset, de Roboflow. Sitio web: https://app.roboflow.com/carlos-doffiny-s-v/proyecto2_rna_carlos_doffiny_sv/1

[2] tzutalin. (2016). LabelImg, de Github. Sitio web: https://github.com/tzutalin/labelImg

[3] Roboflow. (2022). Roboflow, de Roboflow. Sitio web: https://roboflow.com/

[4] Detectron2. (2022). Detectron2 Documentation, de Detectron2. Sitio web: https://detectron2.readthedocs.io/en/latest/index.html

[5] Detectron2. (2022). Detectron2 Model Zoo and Baselines, de Github. Sitio web: https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md#coco-object-detection-baselines

[6] Programador Clic. (2022). Explicación detallada del algoritmo R-CNN más rápido, de Programador Clic. Sitio web: https://www.programmerclick.com/article/3370793669/

IX. BIOGRAFÍA

Carlos Doffiny S-V, nació en Caracas-Venezuela el 15 de agosto de 2001. Realizó sus estudios secundarios en el Instituto Escuela. En el 2018 inició sus estudios universitarios de Ingeniería Informática, en su ciudad natal, en la Universidad Católica Andrés Bello, carrera en la cual se encuentra actualmente cursando el 8vo semestre. Desde el 2021 desempeña el cargo de Mobile Developer en Global Consulting Factory, siendo partícipe de diferentes proyectos Fintech. Adicionalmente, se encuentra realizando un intercambio académico en la Universidad Autónoma de Occidente, en Cali, Colombia.
Áreas de interés: informática, seguridad, criptomonedas, desarrollo de software y redes neuronales. (carlos.doffiny@uao.edu.co)