INFORME_PROYECTO

Juan Sebastian Fajardo Zarta
Andres Felipe Parra Barragan

El objetivo de este proyecto es desarrollar un modelo de detección de tráfico desde la descarga y preparación de los datos, hasta la visualización y análisis de los resultados. Se han implementado dos notebooks principales que abarcan lo mencionado anteriormente, desde la exploración de datos hasta la creación y evaluación de modelos basados en arquitectura YOLO.

- Descripción de la estructura de los notebooks entregados
    - Notebook 1: Exploración de Datos
      En este notebook se busca preparar y analizar los datos de imágenes y etiquetas, el dataset en general. El conjunto de datos "*traffic-detection-projectse*" de *yusufberksardoan*, se obtuvo desde Kaggle [1], para ello se utilizó *!pip install kagglehub* y paquetes esenciales como *kagglehub*, así mismo se importaron librerías necesarias para el procesamiento de imágenes, el manejo y la visualización de datos.

      Una vez el dataset obtenido, se organizaron los archivos en carpetas específicas tales como "`train`", "`test`" y "`valid`"; luego se identificaron las imágenes y etiquetas para cada subconjunto, emparejando según la correspondencia entre imágenes y archivos de etiquetas. Para los formatos de anotación, se convierte el formato YOLO de las etiquetas a Pascal VOC para facilitar el análisis y la integración con herramientas estándar. Los datos se procesan y se cargan en conjuntos de datos TensorFlow (`train_dataset`, `val_dataset`, `test_dataset`), para visualizar los datos anotados se realiza el emparejamiento de imágenes y etiquetas con sus respectivos bounding boxes (cajas delimitadoras). Y al final se realiza un análisis estadístico de la distribución de datos en los conjuntos de entrenamiento, validación y prueba, mediante un gráfico de torta.

    - Notebook 2: Modelo Yolo
      En este notebook se implementó un modelo de detección de objetos basado en YOLO (You Only Look Once) [2], implementado en TensorFlow, con el propósito de detectar diferentes tipos de vehículos y objetos en un conjunto de datos de tráfico. El modelo toma imágenes, las preprocesa, las pasa por una red neuronal convolucional y devuelve las predicciones en forma de cajas delimitadoras con sus respectivas probabilidades. Así como en el notebook anterior, la primera parte está dedicada a la carga de datos desde los archivos locales y se realiza una revisión básica de la estructura y se hace una limpieza y transformación de datos, normalizando las variables asegurando que los datos sean adecuados

para la siguiente fase del modelado donde se visualizan las imágenes con cajas de delimitación correspondiente a cada objeto.

Luego se define un modelo básico de YOLO utilizando Keras, que contiene varias capas convolucionales que extraen características de la imagen, seguidas de capas densas que realizan la predicción de las cajas delimitadoras para cada celda de la cuadrícula, prediciendo la clase, el centro y el tamaño de cada objeto, de ahí se entrena con los datos de *trein* y *valid.* Una vez entrenado, el modelo realiza predicciones para obtener las coordenadas de las cajas delimitadoras y se aplica un umbral de confianza para filtrar las predicciones de baja probabilidad; así mismo, se aplica el algoritmo de Supresión de No Máximos (NMS) para eliminar las cajas que se solapan demasiado y finalmente se implementa una función para mostrar las predicciones en una imagen, con las cajas delimitadoras sobrepuestas.

- Descripción de la solución.
  La solución se centra en la preparación de un conjunto de datos para entrenamiento en detección de tráfico, se muestra el flujo de trabajo a continuación.

  - Exploración de datos del proyecto:
    El flujo del proyecto inicia en una exploración de datos, donde se analiza preliminarmente las imágenes y etiquetas, aplicando técnicas de normalización y conversión para garantizar que las imágenes y etiquetas sean compatibles con modelos de detección basados en TensorFlow. El notebook 1 procesa las etiquetas originalmente en formato YOLO, donde las coordenadas están normalizadas y las convierte al formato Pascal VOC (Visual Object Classes) que es útil para conjuntos de datos que trabajan en tareas de detección, segmentación y clasificación [3]; en este caso para análisis y visualización. Se utilizan objetos `tf.data.Dataset` para manejar de forma eficiente grandes volúmenes de datos durante el entrenamiento. Para el entrenamiento de modelos se desarrolló un modelo YOLO para detección. De ahí se esperaba evaluar el modelo en términos de precisión, velocidad y robustez en el notebook 2; sin embargo no fue posible realizar predicciones precisas. Por eso se proponen trabajos futuros para dar solución al modelo de modo que se logre realizar las predicciones correctas y optimizadas.

  - Procesamiento:
    Para el preprocesamiento se hizo un escalado y normalización de las imágenes, estas se decodifican y convierten a valores de punto flotante entre 0 y 1, se emparejan las imágenes con etiquetas para garantizar que todos los datos estén completos. Las imágenes se cargaron y redimensionaron a (640 x 640) píxeles. Las etiquetas son preprocesadas para que se ajusten al formato de la cuadrícula de YOLO y se tomó la información de clase, el centro de la caja, el ancho y la

altura, para organizarlo en una cuadrícula de 20x20. Esto se almacena en una matriz `y_true`, que es la salida esperada para la red.

Con la función `create_dataset` se genera un conjunto de datos de entrenamiento a partir de las imágenes y etiquetas procesadas con un tamaño de batch. Se realizaron pruebas con varios valores para el *batch size* con el objetivo de reducir el consumo de memoria de la GPU, ya que la capacidad de la GPU proporcionada por Colab era limitada. Este ajuste permitió identificar un tamaño de *batch* óptimo que balancea el uso eficiente de la memoria y el rendimiento del modelo. La elección final que fue 4, redujo significativamente la ejecución sin comprometer la estabilidad del entrenamiento. Se exploraron también otras configuraciones en la capa de salida (*output layer*) del modelo para garantizar una correspondencia adecuada con las dimensiones y etiquetas de las cajas delimitadoras (*bounding boxes*). Se utiliza `tf.data` para optimizar el rendimiento.

De igual manera se implementó el enfoque de precisión mixta (mixed precision) para ahorrar memoria en la representación de números de punto flotante. Esto se logró configurando la política global del entorno TensorFlow mediante:

```
#Con esta precisión mixta se optimiza el uso de la GPU, mejorando el rendimiento y reduciendo el uso de memoria
from tensorflow.keras.mixed_precision import set_global_policy
set_global_policy('mixed_float16')
```

Figura 1. Optimización de la GPU.

Este método permite que el modelo maneje datos de 16 bits en lugar de los 32 bits estándar, lo que permite una reducción en el uso de memoria.

Con estas medidas en conjunto se esperaba optimizar el proceso de entrenamiento, haciendo posible trabajar con configuraciones más complejas y alcanzar un mejor ajuste del modelo dentro de las limitaciones de hardware disponibles.

- Modelo Yolo:
  Se define un modelo básico de YOLO utilizando Keras. Este modelo tiene varias capas convolucionales que extraen características de la imagen, seguidas de capas densas que realizan la predicción de las cajas delimitadoras para cada celda de la cuadrícula, prediciendo la clase, el centro y el tamaño de cada objeto. Para entrenar al modelo, se compila con el optimizador Adam y la función de pérdida MSE (Error Cuadrático Medio). Y se entrena con los datos de entrenamiento y validación.

  Una vez entrenado, el modelo realiza predicciones que se decodifican para obtener las coordenadas de las cajas delimitadoras. Se aplicó un umbral de

confianza para filtrar las predicciones de baja probabilidad y se aplicó el algoritmo de Supresión de No Máximos (NMS) para eliminar las cajas que se solapan demasiado, lo cual mejoró la precisión de las predicciones finales. Así mismo se graficaron los bounding boxes correspondientes a cada objeto en la imagen, con su respectiva clase para verificar la calidad y precisión de las anotaciones en la visualización.

- Descripción de las iteraciones que hiciste
  En la única iteración implementada en el notebook 2, se desarrolló un método para visualizar las predicciones del modelo YOLO sobre un conjunto de datos de entrenamiento y validación. El objetivo principal de esta iteración era evaluar de manera preliminar el desempeño del modelo en términos de detección de objetos, mostrando visualmente las predicciones realizadas sobre imágenes del dataset.

  El código procesaba un lote de imágenes de los datasets (entrenamiento y validación) y utilizaba las técnicas que se han venido mencionando en el transcurso de este informe, que mencionamos nuevamente, a groso modo:

  - ➢ Predicción con el modelo YOLO: Se generaron predicciones utilizando el modelo entrenado. Las salidas del modelo representan los bounding boxes junto con las clases y probabilidades asociadas.

  - ➢ Decodificación de predicciones: Las predicciones crudas del modelo fueron transformadas en cajas delimitadoras (bounding boxes) utilizando una función de decodificación personalizada, que toma en cuenta el tamaño de la rejilla (grid size), el número de clases, y las dimensiones de las imágenes.

  - ➢ Supresión no máxima (Non-Max Suppression): Para refinar las predicciones, se eliminó el solapamiento entre cajas y se seleccionaron únicamente las de mayor confianza. Esto ayuda a reducir el ruido en las predicciones finales.

  - ➢ Visualización de las predicciones: Se graficaron las imágenes originales junto con las cajas predichas y las etiquetas correspondientes utilizando una función de ploteo (`plot_predictions`), con el fin de evaluar visualmente la calidad de las predicciones; sin embargo, las visualizaciones generadas no mostraron predicciones precisas

- Descripción de los resultados.
  Del notebook 1, se identificaron imágenes correctamente anotadas mediante las visualizaciones como se observa en la figura 1.a, y fueron exitosas las conversiones de coordenadas de YOLO a Pascal VOC; así mismo el gráfico de torta mostró la representación del conjunto de datos, como se muestra a continuación en la figura 1.b.

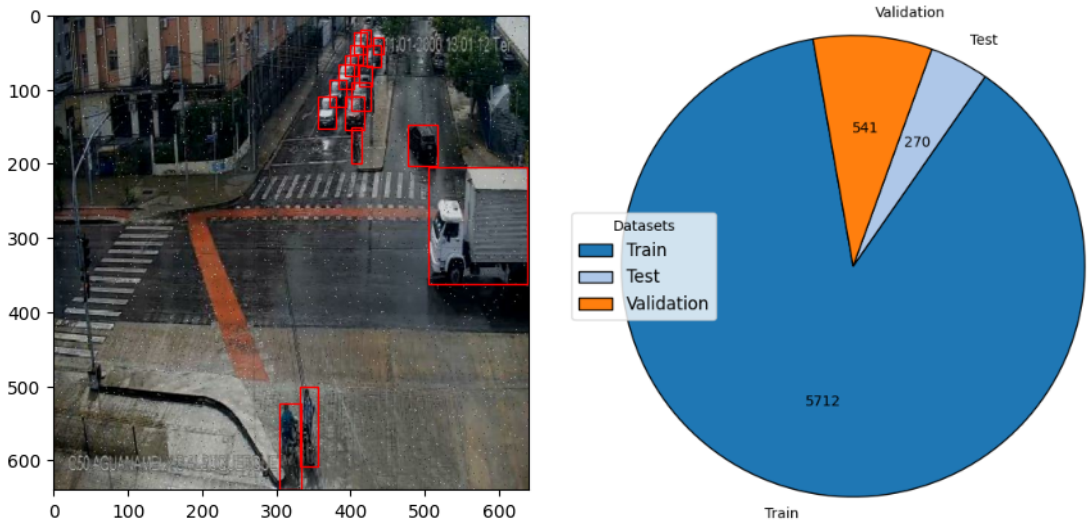

Figura 2. a. Visualización de imágenes anotadas. b. Gráfico de torta del conjunto de datos.

Inicialmente se puede evidenciar la correcta delimitación de los objetos en las imágenes anotadas, de acuerdo a su tamaño y forma, mostrando correcta identificación de objetos. Respecto al gráfico de torta, se puede evidenciar que los datos del train representan el 87%, los de validación el 8% y de test el 5%. Aunque los tamaños de los conjuntos son diferentes, la mayor parte de los datos se destina correctamente al entrenamiento (lo cual es ideal porque aquí el modelo aprende); mientras que los conjuntos de validación y prueba tienen un número menor de muestras, pero suficiente para ajustar y evaluar el modelo.

El conjunto de validación es lo suficientemente grande como para ajustar hiper parámetros y verificar que el modelo no esté sobre ajustado; así mismo, el conjunto de prueba tiene un tamaño adecuado para evaluar la capacidad del modelo de generalizar.

Para el caso del notebook 2, aunque el modelo YOLO generó métricas consistentes durante los *epochs*, como la disminución de la función de pérdida (*loss*) y valores razonables en los indicadores de desempeño, las predicciones visuales no lograron identificar objetos de manera precisa. Esto puede deberse a diversas razones inherentes al proceso de entrenamiento y evaluación, como un preprocesamiento insuficiente, hiper parámetros desajustados, o una configuración inadecuada del modelo.

La imagen que se muestra en la figura 3 con las cajas de delimitación correspondientes a cada objeto en la imagen, con su respectiva clase, muestra que las etiquetas correspondientes desde el dataset trae inconsistencias que influyen en las dificultades para hacer que el modelo prediga correctamente, como se observa en la clase person en el camión.

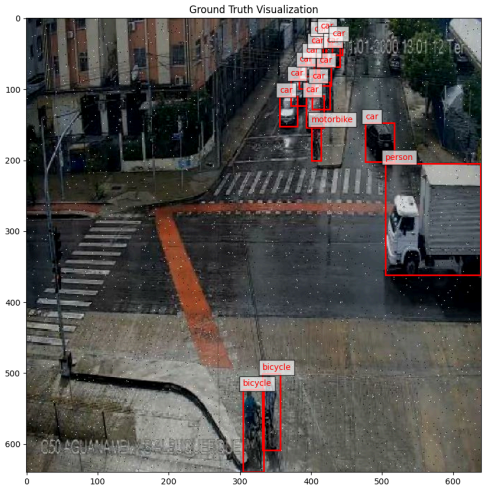

Figura 3. Visualización de las imágenes con las cajas delimitadoras correspondientes a cada objeto, con su respectiva clase.

Por otra parte puede deberse a un entrenamiento inicial insuficiente, que aunque el entrenamiento mostró una disminución de la pérdida, es posible que el modelo no haya alcanzado un punto de convergencia suficiente. Esto puede ocurrir si el número de epochs fue bajo o si el tamaño del lote (batch size) fue demasiado pequeño para capturar patrones generalizables. Tuvimos en cuenta que el tamaño del batch afecta directamente el rendimiento del modelo y la estabilidad del entrenamiento, por lo que usamos un batch size pequeño (entre 1 y 16), por las limitaciones de memoria GPU, que incluyó más ruido en las actualizaciones e hizo que las métricas oscilaran más, dificultando la convergencia, y haciendo un entrenamiento más lento.

Por otro lado, la métrica que usamos, como el MSE (Mean Squared Error), no considera el solapamiento de las cajas predichas, ni la clasificación correcta de las etiquetas. Aunque el MSE puede mejorar durante el entrenamiento, no garantiza que las predicciones sean precisas, que pudo ser una de las razones por las que no dio la predicción de nuestro modelo como se observa en la figura 4 donde ni siquiera aparecen las bounding boxes.

Figura 4. Visualización de la predicción del modelo desarrollado.

En resumen, a pesar de contar con métricas que evolucionan adecuadamente durante las épocas de entrenamiento, las predicciones visualizadas aún no reflejan un modelo bien ajustado. Este desajuste puede ser resultado de un preprocesamiento insuficiente de los labels y los anchors, así como de limitaciones en la selección de hiper parámetros.

- Conclusiones
  El notebook 1 proporciona una base sólida para proyectos de detección de tráfico, permitiendo explorar y preparar datos de manera eficiente. Las herramientas de análisis y visualización integradas garantizan la calidad de las anotaciones, y los resultados muestran que los datos están preparados para el entrenamiento de un modelo de detección de tráfico.

  En el notebook 2, se buscó entrenar un modelo de detección de objetos basado en YOLO para reconocer y localizar diferentes clases de objetos, utilizando un conjunto de datos con las clases bicycle, bus, car, motorbike y person. Aunque el entrenamiento generó métricas como la pérdida del modelo, que parecen tener sentido a lo largo de las épocas, las predicciones finales aún no son precisas ni satisfactorias. Este desempeño limitado puede atribuirse a varios factores identificados durante el análisis del proceso.

- Futuras implementaciones
  Aunque el modelo desarrollado muestra un rendimiento prometedor, existen áreas clave donde se podría mejorar su efectividad. A continuación, detallamos algunas consideraciones para futuras mejoras del modelo:

  En futuras iteraciones, se recomienda utilizar anchors específicos para diferentes clases de objetos, lo cual puede mejorar la precisión del modelo al adaptarse mejor a la variabilidad de tamaños. En el caso del uso de una métrica estándar como el MSE (Error Cuadrático Medio) podría no ser el enfoque más adecuado, especialmente cuando se trata de tareas de clasificación que implican la predicción de cajas delimitadoras o objetos con distintas escalas. Una mejora significativa sería la creación de una métrica personalizada que tenga en cuenta las características específicas del problema. El mAP (Mean Average Precision) puede mejorar la métrica, ya que considera no sólo la precisión de las predicciones, sino también la calidad del bounding box (incluyendo el overlapping entre cajas y el tamaño de las mismas), lo que lo hace más robusto para problemas de detección de objetos.

  Si bien este tipo de métricas proporcionan información más precisa, es fundamental balancear la precisión con la eficiencia, sobre todo en sistemas que requieren respuestas rápidas. En este sentido, se podrían explorar técnicas de optimización para reducir la carga computacional sin sacrificar el rendimiento, que fue lo que se hizo en el modelo actual, se sacrificó el rendimiento por reducir la carga computacional.

A pesar de que el modelo actual cumple con los objetivos establecidos, se propone investigar la implementación de arquitecturas más avanzadas, como redes neuronales convolucionales (CNN) más profundas, modelos de atención o incluso transfer learning, los cuales podrían mejorar aún más el desempeño en tareas complejas de clasificación y detección de objetos.

En resumen, implementar estos cambios permitirá abordar mejor los desafíos de clasificación de objetos con diferentes dimensiones y optimizar el tiempo de ejecución, lo que beneficiaría su aplicación en el mundo real.

- Referencias

[1] https://www.kaggle.com/datasets/yusufberksardoan/traffic-detection-project?resource=download
[2] https://www.datacamp.com/es/blog/yolo-object-detection-explained
[3] https://docs.ultralytics.com/es/datasets/detect/voc/