DETECCIÓN DE BOTELLAS DE PLASTICO MEDIANTE REDES CONVOLUCIONALES

JORGE MARIO CASTRILLÓN OCAMPO
BETZY JULIANNY CÁRDENAS IBÁÑEZ
KARIN JULIETH AGUILAR IMITOLA

Proyecto Final
Fundamentos de Deep Learning

Docente
Raúl Ramos Pollan

2023-II

INTRODUCCIÓN

En el marco del cumplimiento de la resolución 1407 de 2018, la cual establece directrices para la responsabilidad extendida del productor en envases y empaques, resulta relevante la correcta clasificación y detección de envases plásticos en los centros de acopio de residuos. En este contexto, este proyecto busca entrenar y probar modelos de detección de objetos que puedan identificar botellas de plástico en escenarios del mundo real, para favorecer la trazabilidad y el control sobre la cantidad de envases que entran al sistema. Este informe aborda desde la obtención de los datos hasta la validación de desempeño del modelo utilizado.

1 OBTENCIÓN DE DATOS

El conjunto de datos utilizado para el diseño y entrenamiento del modelo consiste en un *dataset* obtenido en el repositorio de Kaggle, con las siguientes características:

- **Tipo de datos:** Imágenes anotadas.
- **Tamaño en disco:** 1.71 GB.
- **Clase única:** Botellas de plástico

Para acceder al *dataset*: https://www.kaggle.com/datasets/siddharthkumarsah/plastic-bottles-image-dataset.
De los archivos disponibles en el *dataset*, se usaron las imágenes y etiquetas contenidas en las carpetas *test* y *train*.

El proceso de depuración de esta información se encuentra en notebook 01_Depuración_de_imagenes

2 ESTRUCTURA DEL MODELO

El algoritmo You Only Look Once (YOLO) es un sistema de código abierto para la detección de objetos en imágenes y videos, desarrollado por Joseph Redmon en el 2016. Para la implementación del proyecto se hizo uso de la versión 3 de YOLO que se destaca por su capacidad para identificar objetos en imágenes y videos con rapidez. Además de su agilidad, sobresale en la detección de múltiples clases, manejando un amplio rango de categorías de objetos en una sola ejecución. Su enfoque innovador incluye la división de la imagen en celdas (grillas), permitiendo predicciones precisas en distintas partes de la imagen, y el uso de escalas múltiples para detectar objetos de diferentes tamaños de manera efectiva. YOLOv3 utiliza Darknet-53 como su sólida arquitectura base, una red neuronal convolucional profunda que aprende patrones complejos de los datos de entrada (Almong, 2020).

Dentro de las métricas de desempeño de YOLO para detección de imágenes se destaca el mAP (*Mean average precision*), la cual permite evaluar el rendimiento del modelo en un conjunto de

datos a través de diferentes umbrales de confianza. Para cada clase de objeto se calculan las precisiones y los recalls, y se obtiene el área bajo (El mAP se obtiene de la siguiente manera:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

Donde N es el número total de clases y AP_i es el área bajo la curva de precisión-recall para la clase i (Soto Serrano, s.f.).

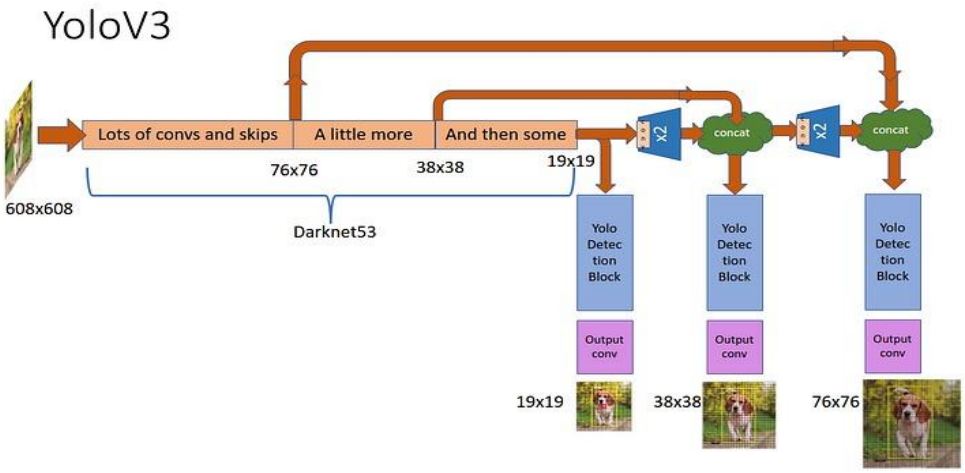


Figura 1. Arquitectura YOLOV3. Fuente: Almog (2020)

3 IMPLEMENTACIÓN DEL MODELO YOLOV3

Paso 1: Habilitar y probar la GPU: En el entorno de Colab se selecciona como tipo de entorno de ejecución T4 GPU debido para tener un entrenamiento más rápido.

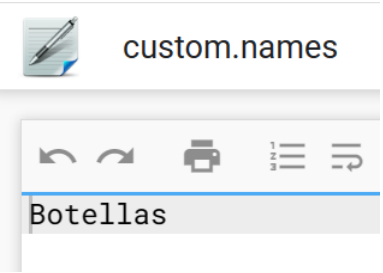
1. custom_data/images
2. custom_data/custom.names
3. custom_data/train.txt
4. custom_data/test.txt
5. custom_data/backup
6. custom_data/detector.data
7. custom_data/cfg

Paso 2: Montaje de la unidad para almacenar y cargar archivos: Se elige montar los archivos desde Google Drive para que siempre estén disponibles en la nube y cualquier integrante del grupo pueda acceder a ellos con facilidad, además permite que los resultados del entrenamiento se salven y queden disponibles mientras termina el entrenamiento y estén seguros al no estar en un entorno local. así se crea la carpeta *custom_data* y los demás archivos y carpeta quedan como en la imagen.

Es necesario vincular su propio Drive para utilizar las imágenes. Dado que se proponen dos estructuras de modelo, los datos usados para el notebook 02_Modelo_inicial se

encuentran disponibles en el siguiente enlace Carpeta Custom_data, mientras que para el notebook 03_Modelo_modificado los archivos se encuentran en la Carpeta Custom_data.

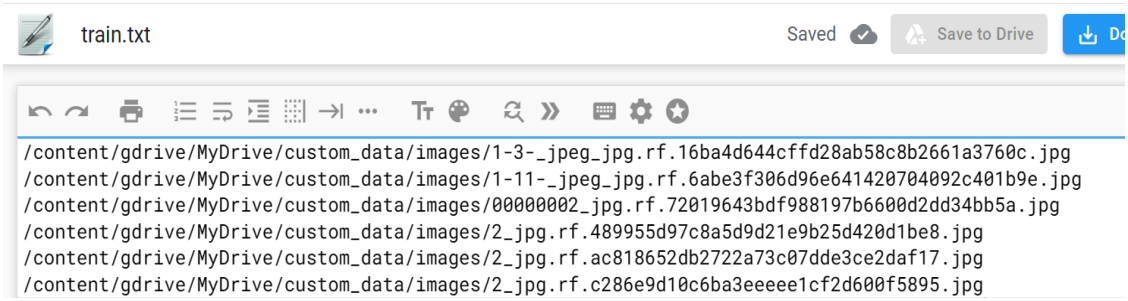
Paso 3: Creación del directorio de imágenes: Se descomprimen las imágenes y los archivos con los labels de entrenamiento y test en una carpeta única llamada *images* en Google Drive, desde donde estarán disponibles para el entrenamiento y la evaluación. Durante la creación de este directorio, se eliminó una de las imágenes correspondiente a la carpeta *train* y su respectiva etiqueta, dado que tenía mal el nombre. En total, se registraron 5648 imágenes para su uso en el entrenamiento y la evaluación.



Paso 4: Creación del archivo Custom.names:

En un archivo se customiza el nombre de las clases, en este caso es una única clase y son las botellas de plástico, este archivo queda ubicado en la carpeta *custom_data*.

Paso 5: Creación de los archivos Train y Test: Como se han unido todas las imágenes tanto las de entrenamiento como las de test en un solo directorio, como se explica en el paso 3, es necesario indicarle al modelo cuales serán de entrenamiento y test, ya que ambas comparten la misma dirección, por lo que se crean los archivos llamados *train.txt* y *test.txt* los cuales contienen los nombres de las imágenes que serán utilizadas para cada una de las tareas. En ambos casos el contenido del archivo .txt contiene la dirección y el nombre del archivo que será utilizado ya sea para entrenamiento o test.



Paso 6: Creación del directorio Backup: Dentro de la carpeta creada para almacenar los archivos requeridos para el entrenamiento también se crea otra carpeta llamada *backup* donde se irán guardando los pesos alcanzados en el entrenamiento, dado que es un proceso largo y puede ocurrir algún problema que ocasión que se detenga, es inteligente guardar el progreso hasta el momento. Allí se guardará de forma recurrente el archivo con los pesos actualizados, los pesos luego de 1000 baches, 2000 baches y los pesos finales.

Paso 7: Creación del fichero de datos YOLO: Por medio de este archivo se le indica al modelo cuales son los archivos de entrenamiento, test, nombres de las clases, número de las clases y donde guardar los pesos del entrenamiento.

```
classes=1
train=custom_data/train.txt
valid=custom_data/test.txt
names=custom_data/custom.names
backup=backup/
```

Paso 8: Clonación del directorio para usar Darknet: se clona el modelo de Yolo desde https://github.com/AlexeyAB/darknet, y se realizan los cambios necesarios para que el modelo se entrene con GPU.

Paso 9: Realización de cambios en el archivo de configuración de Yolo: Este archivo contiene las definiciones de la arquitectura del modelo, aquí se especifican las características de cada una de las capas. De igual forma se definen los cambios que se quieren realizar al modelo en caso de que se hagan, además se definen elementos adicionales como el número de baches, tamaño de las imágenes, *batch*, canales, saturación, exposición. En términos generales aquí es donde se customiza el modelo Yolo de acuerdo con las necesidades.

Paso 10: Descarga de los pesos pre-entrenados: Se descargan pesos pre-entrenados que van a ser utilizados en el modelo para tener mejores resultados en menor tiempo y que el entrenamiento no comience desde 0.

Paso 11: Entrenamiento del modelo: Se entrena el modelo por 2000 épocas y se guardan los nuevos pesos de forma recurrente por si ocurre algún tipo de falla en el entrenamiento no sea necesario comenzar desde los pesos pre entrenados mencionados en el paso 10.

Paso 12: Cálculo de la precisión media: Se carga la función que calcula del map con los últimos pesos alcanzados por el modelo y se evalúa

```
 CUDA-version: 11080 (12000), cuDNN: 8.9.6, CUDNN_HALF=1, GPU count: 1
 CUDNN_HALF=1
 OpenCV version: 4.5.4
 0 : compute_capability = 750, cudnn_half = 1, GPU: Tesla T4
net.optimized_memory = 0
mini_batch = 1, batch = 16, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  14 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  15 Shortcut Layer: 12,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  16 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
```

4 AJUSTE A LA ESTRUCTURA DEL MODELO

4.1 Estructura del modelo inicial

En la estructura del modelo inicial se hace uso de una arquitectura YOLOV3, la cual considera un extractor de características Darknet-53 que tiene 52 convoluciones, además contiene conexiones de salto como ResNet y 3 cabezales de predicción como FPN. En la prueba inicial realizada se considera para las imágenes una resolución de 416X416 con 3 canales. Los parámetros considerados en la configuración del modelo están establecidos en la tabla 1.

Tabla 1. Configuración de los parámetros de los modelos de Yolo

Parámetro	Configuración
Resolución	416 x 416
Batch	64
Max batches	2000
Steps (80%, 90%)	1600, 1800

4.2 Estructura del modelo modificado

En la estructura del modelo modificado se consideró la misma configuración para el modelo inicial presentado en la tabla 1, pero con un cambio a un solo canal. Esta modificación representa una alteración en la intensidad de los colores, lo que lleva a que las imágenes se presenten en una escala de grises. Este ajuste busca optimizar la eficiencia del modelo al trabajar con datos de tonalidades simplificadas. Adicionalmente, se consideró cambiar el tamaño de las cajas delimitadoras.

5 VALIDACIÓN DEL DESEMPEÑO PARA LAS ESTRUCTURAS DEL MODELO

5.1 Desempeño del modelo inicial

Tabla 2. Información del desempeño del modelo inicial.

Precisión promedio media (mAP)	Verdadero positivo (TP)	Falso positivo (FP)	Falso negativo (FN)
21.77%	358	290	947

Teniendo en cuenta la Tabla 2, se puede observar que la métrica mAP es del 21.77%, lo cual indica un valor pequeño para la precisión promedio media del modelo para la detección de objetos en las imágenes evaluadas, lo cual sugiere que el modelo tiene un margen de mejora en términos de precisión en la detección de objetos. Adicionalmente, se presentan 947 falsos negativos, un valor

que es alto con respecto a los otros, lo que indica que el modelo está perdiendo la detección de muchos objetos que realmente están presentes en las imágenes.

En conclusión, lo anterior sugiere que hay margen para mejorar la precisión en la detección de objetos. Es posible que ajustes en los hiperparámetros o en la arquitectura del modelo puedan mejorar estos resultados.

5.2 Desempeño del modelo modificado

Tabla 3. Información del desempeño del modelo modificado.

Precisión promedio media (mAP)	Verdadero positivo (TP)	Falso positivo (FP)	Falso negativo (FN)
9.73%	0	0	1305

De acuerdo con lo observado en la Tabla 3 se presenta un valor del mAP del 9.73%, lo que indica que el ajuste de las tonalidades y el tamaño de las cajas delimitadoras empeora la precisión promedio media del modelo para la detección de objetos. Además, hay 1305 casos en los que el modelo predijo incorrectamente la clase negativa, estos son casos en los que el modelo predijo que las imágenes no tenían botellas, pero la etiqueta era positiva, es decir que si había botellas.

6 VERIFICACIÓN EN TIEMPO REAL DE LA DETECCIÓN DE IMÁGENES

Para la detección de objetos en tiempo real se hizo uso de una biblioteca OpenCV (cv2). El código utilizado (https://pastebin.com/bb9zKc7j) permite que por medio de la cámara web se pueda capturar fotogramas continuos y procesa cada fotograma para identificar objetos en la escena. Dentro de la configuración, se hace uso de los pesos arrojados por el modelo YOLOV3 y se obtiene la visualización en el fotograma original.

Figura 2. Detección de botellas de plástico en tiempo real.

Referencias

Almog, U. (2021). YOLO V3 explained - towards data science. *Medium*. https://towardsdatascience.com/yolo-v3-explained-ff5b850390f

Soto Serrano, A. (2017). *YOLO Object Detector for Onboard Driving Images*.