Red neuronal convolucional para la detección de tomates cultivados en invernadero

Paulina Arregocés Guerra, *paulina.arregocesg@udea.edu.co*
Juan Pablo Velásquez Cardona, *juan.velasquez25@udea.edu.co*

Fundamentos de Deep Learning
Universidad de Antioquia
2024

1. Introducción.

En este informe se presenta un ejercicio académico de la aplicación de Redes Neuronales Convoluciones (CNN, de sus siglas en ingles). Particularmente, este trabajo se enfoca en la detección de objetos para el caso de uso de la detección de tomates en un cultivo de tomates bajo invernadero. El desarrollo de este trabajo beneficia la implementación del proyecto de regalías "Plataforma de Agricultura Inteligente", el cual busca mejorar el rendimiento de cultivos de tomate en invernadero a través de modelos de inteligencia artificial.

2. Estructura de los archivos.

Los archivos del repositorio de GitHub están organizados de la siguiente manera:

```
Poyecto_FD
├── datos_tomate.zip
│   ├── images
│   ├── test_img
│   ├── test_xml
│   ├── explo_img
│   └── explo_xml
├── YoloWeight
│   └── yolov4.weights
├── yolo-v4-tf.keras
├── 00 - selección de datos.ipynb
├── 01 - exploración de datos.ipynb
├── 02 - preprocesado.ipynb
├── 03 - procesamiento.ipynb
├── Clases.txt
├── Annotations.txt
└── Annotations_test.txt
ENTREGA1
INFORME_PROYECTO
```

El archivo de `datos_tomate.zip` contiene cinco carpetas con imágenes y etiquetas de fotos de cultivo de aguacate bajo invernadero. La carpeta llamada `images` contiene 8 imágenes de formato `.jpg` de un dataset. Adicionalmente, las carpetas `explo_img` y `explo_xml` `test_img`, `test_xml`, corresponden a datos implementados para entrenamiento validación y pruebas. La carpeta `YoloWeight` contiene los pesos pre-entrenados de la red YoloV4 disponibles en el repositorio de GitHub official del proyecto darknet de *AlexeyAB* (`https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights`). La carpeta `yolo-v4-tf.keras.` contiene la implementación de YoloV4 por *taipingeric* (`https://github.com/taipingeric/yolo-v4-tf.keras.git`), con la siguiente estructura de carpetas:

```
yolo-v4-tf.keras
├── notebook
├── logs
├── img
├── eval
├── class_names
├── asset
├── .git
└── _pycache_
```

Adicionalmente, la carpeta `yolo-v4-tf.keras.` contiene 7 archivos de formato `.py` que son necesarios para los diferentes procesos de procesamiento de las imágenes, algunos de ellos modificados para nuestro proposito.

En la carpeta principal, además, se encuentran 4 notebooks de Google Colaboratory de formato `.ipynb`, los cuales contienen:

00. seleccion de datos: Este script no es necesario ejecutarlo. Sin embargo, se deja a disposición del usuario para seleccionar de manera aleatoria imágenes y archivos .xml. Es posible configurar la cantidad de datos dependiendo de la memoria y de capacidad de cómputo del usuario.

01. exploración de datos: En este script se exploran algunos de los archivos del conjunto de imágenes con sus respectivas etiquetas. Para lograrlo se toma la carpeta con las imágenes y sus correspondientes etiquetas PASCAL VOC. Las etiquetas de formato .xml se transforman en un archivo de formato `CSV`. Este archivo es utilizado para la correcta ubicación de las cajas de detección de los tomates. Además, se exploran algunos modelos de clasificación como InceptionV3 y ResNet para verificar su desempeño con este conjunto de datos tomando algunos recortes de la imagen.

02. preprocesado: En este script se transforman las etiquetas del archivo `.xml` a etiquetas de formato YOLOv4 en un archivo de formato `.txt`. Adicionalmente se verfica la correcta transformación de las etiquetas de manera gráfica.

03. procesamiento: En este script se procesan las imágenes. Se descargan los pesos y el modelo YOLOv4 de keras. Se dividen las imágenes en entrenamiento, validación y prueba.

Se entrena el modelo según las indicaciones de del modelo YOLOv4 y finalmente se evalúan algunas métricas de desempeño del entrenamiento y para el conjunto de pruebas

En la carpeta principal se encuentran también los archivos Clases.txt, Annotations.txt y Annotations_test.txt. El archivo Clases.txt contiene el nombre de las clases, en este caso solo es una clase "tomato". Adicionalmente, Annotations.txt y Annotations_test.txt son archivos limpios que se completaran con las etiquetas de los datos para entrenamiento, validación y prueba.

3. Descripción de la solución.

En esta sección se describe el pre-procesamiento de los datos, necesario para el entrenamiento en la etapa siguiente. Se presenta a YOLOv4 como modelo empleado y la arquitectura de la solución. Por último se definen los hiperparámetros para el entrenamiento.

3.1. Pre-procesamiento.

El pre-procesamiento requiere las carpetas donde se guardan las etiquetas PASCAL VOC (.xml), un archivo con las clases del dataset seleccionado llamado "Clases.txt", que tiene un único registro con la palabra "tomato" y adicionalmente se requiere un archivo llamado "Annotations.txt"que debe estar en blanco, para cargar allí cada uno de los registros correspondientes a cada imagen. Todos estos archivos deben estar en la misma carpeta como se ve en el siguiente ejemplo.

- XML_PATH=/mydrive/Proyecto_FD/explo_xml
- CLASSES_PATH=/mydrive/Proyecto_FD/Clases.txt
- TXT_PATH=/mydrive/Proyecto_FD/Annotations.txt

La transformación de los archivos .xml da como resultado un único archivo con registros .txt en formato YOLO. Cada registro tiene la siguiente estructura:

nombre_imagen x1,y1,x2,y2,id_clase

Donde "x1, y1, x2, y2" corresponden a las coordenadas de la *bounding box*, seguidas por la clase a la que pertenece. Adicionalmente, en el archivo models.py se encuentra un preprocesamiento que consiste en redimensionar la imagen a 416x416 con sus 3 canales y normalizar la imagen dividiendo por 255 cada pixel de la imagen.

3.2. Arquitectura

Se utilizo una red YoloV4 adaptada con un dataset de tomates para el detector. Un detector de objetos está conformado por varios módulos divididos en dos etapas como se puede ver en la imagen 1. YoloV4 usa para estos módulos CSPDarknet53 como *Backbone*, PANet

como *Neck* y YOLOv3 como *Head*, comprendiendo este último los módulos de *Dense Prediction* y *Sparse Prediction* [1]. La configuración asociada a la arquitectura de este modelo puede observarse en el archivo custom_layers.py del directorio yolo-v4-tf.keras. o en el siguiente enlace. (`https://github.com/taipingeric/yolo-v4-tf.keras/blob/master/custom_layers.py`)

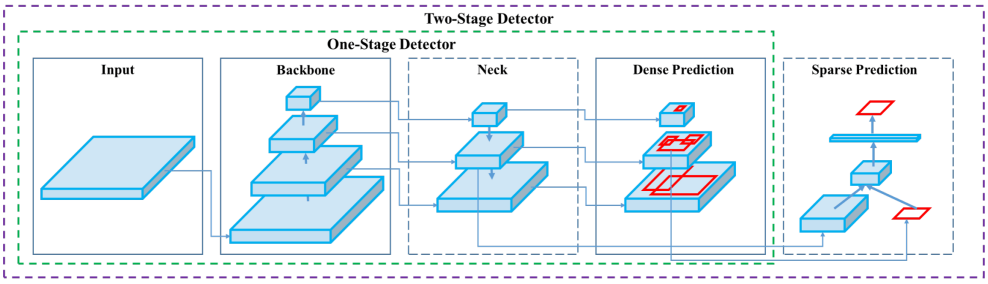

Figura 1: Arquitectura de YoloV4. [1]

3.3. Hiperparámetros

El método utiliza 18 anchors diferentes para detectar los objetos y 3 strides.

anchors: [12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401]
strides: [8, 16, 32],

Adicionalmente, para en entrenamiento se configuran los siguientes hiperparámetros en el archivo config.py y por fuera se seleccionan el número de épocas.

Training
iou_loss_thresh: 0.5, batch_size: 8, num_gpu: 1,

Inference
max_boxes: 100, iou_threshold: 0.413, score_threshold: 0.3,

- Número de imágenes a entrenar: 300.
- Número de épocas: 100.

4. Implementación

Se utilizó la red pre-enetrenada con los pesos de YoloV4 y se entrenó nuevamente con el dataset de imágenes de tomates en invernadero. El entorno de ejecución fue Google Colab PRO con las características listadas a continuación.

- RAM del sistema: 100 GB.
- RAM de la GPU: 15.0 GB.

4.1. Datos

El dataset utilizado para la solución es un recorte de 300 imágenes de la publicación de Magalhaes *et al.* [2]. Esto debido a que el dataset original contiene 25758 imágenes con sus respectivas anotaciones con un peso en disco bastante alto (1.0 GB). Las imágenes para entrenamiento y validación se escogieron de forma aleatoria entre las disponibles en el dataset original que incluyen imágenes con aumento de datos, por lo tanto, algunas están rotadas o con ruido añadido. Para tener disponibilidad de los archivos es necesario clonar el repositorio completo de GitHub, este contiene tanto el dataset utilizado como los programas a reproducir para obtener las soluciones.

4.2. Iteraciones

Para cada época se realizan 32 iteraciones.

4.3. Enlace de youtube

En el video presentado del siguiente **enlace** se explica en máximo 10 minutos los siguientes puntos:

- Descripción y muestra de los datos
- Descripción y muestra del código
- Ejecución paso a paso de los notebooks
- Resultados de la solución.

5. Resultados

5.1. Inception3 y ResNet

Tomando como referencia la exploración de datos, se puede observar que el conjunto de etiquetas con el cual fueron entrenados los métodos Inception3 y ResNet, son limitadas y no es posible clasificar los tomates presentes en las imágenes. Los objetos, aunque se encuentren detectados correctamente, son reconocidos en su mayoria como otro tipo de objeto diferente a tomate.

5.2. YOLOv4

Empleando el modelo de YoloV4 con sus respectivos pesos preentrenados, y sus clases predeterminadas, el resultado de detección y clasificación fue bastante similar al comportamiento de los modelos Inception3 y ResNet, en el cual el modelo detecta los tomates como manzanas “apple”. El resultado se puede ver en la imagen 2.

Figura 2: Resultado de la detección con yolov4 con pesos pre-establecidos de la misma red.

5.2.1. Entrenamiento

Teniendo en cuenta que el modelo no reconoce los tomates, es necesario re-entrenalo con las imágenes y anotaciones propias del dataset. Con los hiperparámetros establecidos y configurados se pasa a hacer el entrenamiento. Las métricas de entrenamiento y validación se observa en la figura 4

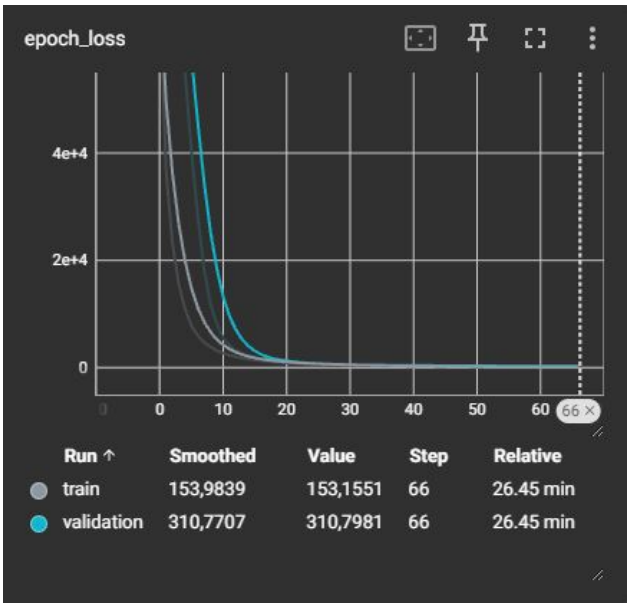

Figura 3: Métricas de entrenamiento

Una vez reentrenado el modelo, se prueba nuevamente la predicción para verificar su correcto funcionamiento como se ve en la figura 4.

Figura 4: Resultado de la detección con yolov4 con modelo re-entrenado.

Con el modelo entrenado es posible detectar tomates que provienen del dataset original con un buen grado de confianza, como se puede ver en la figura 4, sin embargo, al utilizar una imagen propia de plantas de tomates en invernadero, el modelo aún no generaliza adecuadamente como se ve en la figura 5. Trabajos futuros deberían centrarse en alimentar el dataset de entrenamiento con imágenes propias, que permita así calibrar el modelo hacia la detección de tomates en otros ambientes.

Figura 5: Resultado de la detección con yolov4 con modelo re-entrenado para imagen del proyecto de regalías *Plataforma de Agricultura Inteligente (PAI)*.

5.2.2. Prueba

Con los datos de prueba se realizó la evaluación del modelo a partir de diferentes métricas de desempeño. Obteniendo los siguientes resultados.

Para un conjunto de 19 imágenes, se tienen desde el groundtruth 58 estiquetas. De las cuales el modelo detecto 56. 20 de ellas falsos positivos y 36 de ellas las detecto correctamente. Esto genera una precisión del 36 % y un recall del 36 %. La gráfica presicion-recall permite observar un Average Precision (AP) del 57.85 %, la cual corresponde con el mAP debido a que solo se está detectando una única clase. Estos resultados se ven graficamente en las figuras 6, 7 y 8.

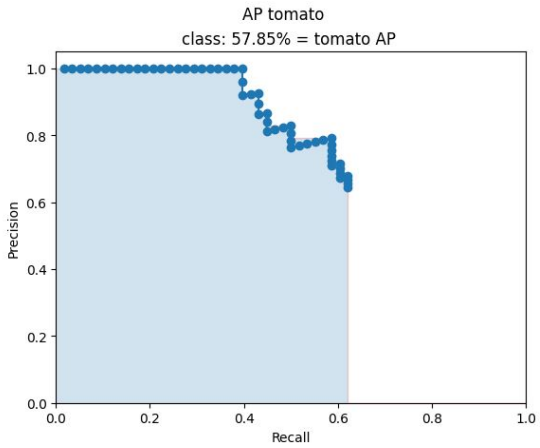

Figura 6: AP del modelo para datos de prueba.

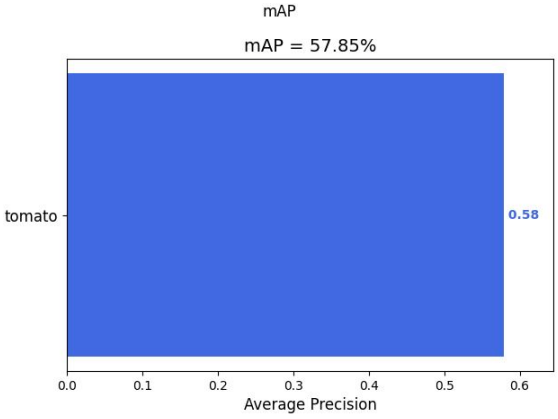

Figura 7: mAP del modelo para datos de prueba.

Estos resultados pueden mejorarse incluyendo mayor cantidad de imágenes en entrenamiento, teniendo en cuenta la capacidad de memoria y procesamiento de cada usuario.

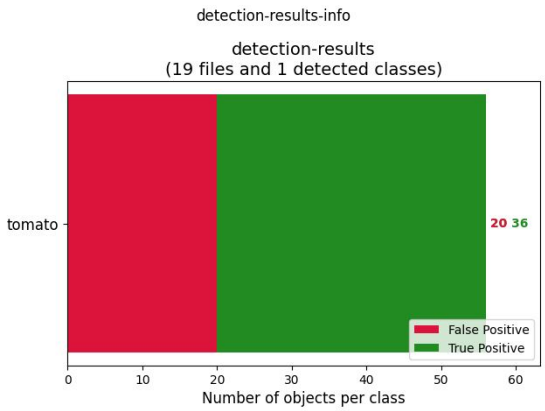

Figura 8: Resultado de la detección del modelo.

Referencias

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[2] S. A. Magalhães, L. Castro, G. Moreira, F. N. Dos Santos, M. Cunha, J. Dias, and A. P. Moreira, "Evaluating the single-shot multibox detector and yolo deep learning models for the detection of tomatoes in a greenhouse," *Sensors*, vol. 21, no. 10, 2021.