Entrenamiento de red neuronal YOLO Darknet

Trabajo práctico correspondiente a la cátedra: Visión por Computadora. Ingeniería electrónica. Año 2020

Giletta Julian, Legajo: 69555
Universidad Tecnológica Nacional, FRC
Córdoba, Argentina
juligiletta97@gmail.com

***Abstract*—Este paper presenta el desarrollo del práctico 13 de la materia Visión por Computadora, en el cual, se debe entrenar el modelo YOLO darknet.**

***Abstract*—This paper presents the development of practical 13 of the Computer Vision subject, in which the YOLO darknet model must be trained.**

***Index Terms*—YOLO, darknet, neuronal network, Computer Vision, UTN**

I. Introducción

El problema a resolver con la red neuronal planteada, es la detección de cascos de seguridad o cabezas sin ellos con el objetivo de monitorear a empleados de una obra y poder alertar a la persona a cargo de que se incumplen las normas de higiene y seguridad. El modelo usado es Tiny-YOLO v4. Esta es una versión más pequeña que sus hermanos mayores, también significa que es menos preciso.
La mayoría de los modelos modernos y precisos requieren muchas GPU para entrenar con un gran tamaño de minibatch, y hacer esto con una GPU hace que el entrenamiento sea realmente lento y poco práctico. YOLO v4 aborda este problema haciendo un detector de objetos que puede entrenarse en una sola GPU con un tamaño de minibatch más pequeño. Esto hace posible entrenar un detector de objetos súper rápido y preciso con una sola GPU.

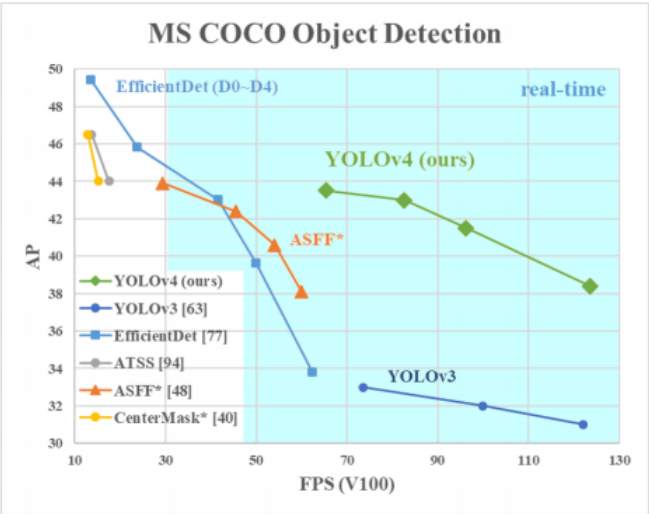

Figure 1. Comparación de YOLOv4 y otros detectores de objetos de última generación. Mejora AP y FPS de YOLOv3 en un 10% y 12%, respectivamente.

II. Dataset

El dataset consiste en 7035 imágenes etiquetadas de personas que trabajan en obras de construcción en todo el mundo, habiendo entre ellas, personas con cascos de protección de diversos colores y otro tanto, de personas sin ellos.

	Entrenamiento	Testeo
Imágenes	5266	1766

Tabla 1: División de imágenes.

Para el etiquetado se utilizó labelImg, la cual es una herramienta libre de etiquetado gráfico. Está escrita en python y usa Qt para la interfaz gráfica. Las etiquetas pueden ser guardadas como archivos XML para el formato PASCAL VOC, el cual es utilizado por otras redes neuronales y también soporta el formato YOLO darknet (.txt).

III. Preparación

Antes de entrenar, se deben preparar los archivos para luego ser subidos a la nube y correr el mismo en Google Colab. Para ello, se descarga el repositorio de darknet de AlexeyAB y el modelo preentrenado que se desee. En este caso, el utilizado es yolov4-tiny.conv.29. Este ultimo se coloca en la misma carpeta de darknet. Luego, en el directorio /darknet/cfg/ encontramos los archivos de configuracion, aqui buscamos yolov4-tiny-custom, le cambiamos el nombre a yolo-obj y por ultimo lo editamos con las siguientes especificaciones:

- Cambiar la linea batch por batch=64
- Cambiar la linea subdivisions por subdivisions=16
- Cambiar la linea max_batches por (clases*2000, siempre y cuando no sea menos del numero de imagenes de entrenamiento).
- Cambiar la linea steps por 80% y 90% de max_batches respectivamente.
- Ingresar el tamaño de entrada de la red width=416 height=416 o cualquier valor multiplo de 32.
- Cambiar la linea classes de las capas [yolo] (3 veces)
- Cambiar la linea filters de la capa anterior a [yolo] (3 veces) por $filters = (clases + 5)x3$

Luego se debe crear un archivo obj.name en el directorio /data/ con las clases en el orden que se asignaron al etiquetar, una por linea.

Además, crear el archivo obj.data en el mismo directorio anterior con el siguiente contenido:

```
classes= 2
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = backup/
```

Seguido a esto, colocar las imágenes de entrenamiento y testeo, cada una en sus carpetas y con sus etiquetas dentro de las mismas, en el directorio /data/obj/. Y por ultimo, modificar el Makefile, activando el uso de GPU, CUDNN y OPENCV.

IV. Entrenamiento

Para realizar el entrenamiento se modifico la plantilla hecha para correr en Google Colab brindada por la cátedra.

Los pasos son:

1) Comprimir los archivos antes preparados y subirlos al Google Colab (nos debemos asegurar que la maquina virtual proporcionada, tenga habilitado el uso de GPU). Lo que se puede realizar es subirlos a Drive y luego vincular Google Colab con Drive.
2) Si optamos por el ultimo paso, vinculamos drive con Google Colab.
3) Descomprimimos el archivo darknet.zip.
4) Compilamos y hacemos ejecutable darknet.

```
!cd /content/darknet
!make clean
!make
!chmod +x ./darknet
```

5) Generamos una carpeta backup en drive y creamos un enlace simbolico entre colab y drive para guardar los pesos que se van generando.

```
!rm /content/darknet/backup -r
!ln -s /content/drive/'My Drive'/colab/backup
/content/darknet
```

6) Si usamos windows, convertimos los archivos train.txt, test.txt, obj.data, obj.names y yolo-obj.cfg con la herramienta dos2unix (previamente debe ser instalada).
7) Por ultimo, corremos el entrenamiento con el siguiente comando:

```
cd /content/darknet
./darknet detector train data/obj.data cfg/yolo-obj.cfg
yolov4-tiny.conv.29 -map -dont_show
```

En el caso que la sesión de colab termine, porque se apago la computadora o finalizo el tiempo, podemos correr el entrenamiento con los últimos pesos generados con el siguiente comando:

```
!./darknet detector train data/obj.data cfg/yolo-obj.cfg
"/content/drive/My Drive/colab/backup/yolo-obj_last.
weights" -map -dont_show
```

A. Resultados

El entrenamiento duro aproximadamente 4 horas y se tuvieron los siguientes resultados:

- Precisión: 60
- Avg loss: 0,7416.

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 27
 Detection layer: 37 - type = 27
1768
 detections_count = 37799, unique_truth_count = 8915
class_id = 0, name = head, ap = 87.71%          (TP = 1894, FP = 367)
class_id = 1, name = helmet, ap = 91.25%        (TP = 5780, FP = 973)
class_id = 2, name = person, ap = 1.07%         (TP = 2, FP = 34)

 for conf_thresh = 0.25, precision = 0.85, recall = 0.86, F1-score = 0.85
 for conf_thresh = 0.25, TP = 7676, FP = 1374, FN = 1239, average IoU = 67.63 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.600086, or 60.01 %
Total Detection Time: 73 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

 mean_average_precision (mAP@0.5) = 0.600086
Saving weights to backup//yolo-obj_6000.weights
Saving weights to backup//yolo-obj_last.weights
Saving weights to backup//yolo-obj_final.weights
```

Figure 2. Detalle del final de entrenamiento.

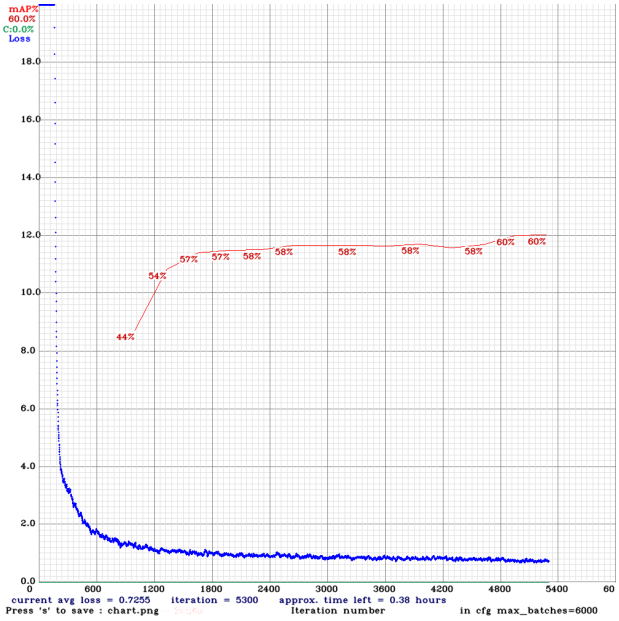

Figure 3. Salida de red.

V. Prueba del modelo

Para ver los resultados obtenidos se utilizaron las librerías de Yolo. Existen 2 APIs, la que utilizamos fue la programada en C e incluida a python con el ejemplo provisto en el Github de darknet de AlexeyAB.

Usando el codigo en python provisto en el repositorio con el nombre de "darknet.py", se pueden pasar imágenes a la red con la función **performDetect**(), la cual tiene los siguientes parámetros:

- imagePath: ruta de la imagen a evaluar.

- configPath: ruta del archivo de configuración.
- weightPath: ruta del archivo de los pesos.
- metaPath: ruta del archivo .data.
- showImage: booleano para activar/desactivar la muestra de bounding boxes.
- makeImageOnly: booleano que en True guarda la imagen pero no la muestra en el momento de ejecución.
- initOnly: booleano que en True solo inicializa globales. No ejecuta una predicción.

Ejemplo:

```
performDetect("/content/darknet/data/obra.jpg", 0.25,
"/content/darknet/cfg/yolo-obj.cfg",
"/content/darknet/backup/yolo-obj_final.weights",
"/content/darknet/data/obj.data", True, False, False)
```

Figure 4. Predicciones.

VI. Conclusión

Concluimos el planteamiento de este practico catalogándolo como exitoso según las pruebas realizadas. Es importante la incorporación de estos temas en materias electivas, si bien, se vio de forma básica, las redes neuronales están en auge nuevamente y tienen infinidades de usos.

En cuanto a conclusiones mas técnicas podemos argumentar que la utilización TinyYolo compromete la precisión de la red pero para los fines de este practico se adecua muy bien y con un coste de procesamiento mucho menor. Esto ultimo no debe ser menospreciado ya que se ahorra mas de la mitad de tiempo de entrenamiento con el mismo hardware.

Si bien el sistema actual puede ser puesto en marcha sin problemas, se recomienda reentrenar con el modelo Yolov4 y comprobar como aumenta la precisión.

References

[1] Redolfi Javier A. And Araguas Roberto Gaston, *Filminas de la Catedra "Vision por Computadora"*.

[2] Bochkovskiy Alexey, Chien-Yao Wang And Hong-Yuan Mark Liao, *YOLOv4: Optimal Speed and Accuracy of Object Detection*.

[3] Bochkovskiy Alexey, *https://github.com/AlexeyAB/darknet*.