Reconocimiento Visual con Deep Learning: Detección de Objetos para el Reconocimiento de Montos

Canales G. Paula A.[1] y Neira R. Alvaro R.[2]

Departamento de Ciencias de la Computación
Universidad de Chile

[1]paulacanales@ug.uchile.cl
[2]alvaroneirareyes@gmail.com

***Abstract*— Este documento tiene como objetivo entender de forma práctica el funcionamiento de detectores de objetos basados en modelos convolucionales. Se entrenó un modelo YOLOv5 sobre un dataset con imágenes de montos manuscritos para detectar cada dígito y su orden correspondiente. Se obtienen resultados tanto en validación como en test alcanzando un accuracy de 85.96%, lo que se considera un buen resultado dada la complejidad del problema.**

I. Introducción

El siguiente documento describe la tarea realizada para el curso Reconocimiento Visual con Deep Learning, el cual corresponde al diplomado de Inteligencia Artificial 2021.

El objetivo es entender, en forma práctica, el funcionamiento de detectores de objetos en imágenes, basados en modelos convolucionales, a través de una aplicación para el reconocimiento de montos manuscritos.

Se decidió utilizar el modelo de detección de objetos YOLO v5 para predecir el monto que aparece en una imagen de entrada, obtenida desde el dataset *orand-car-with-bbs*. Por último, se discutirán los resultados tanto en el conjunto de validación como en el conjunto de test.

Para el desarrollo de este trabajo, se utilizó un *notebook* en Google Colab [4] en el cual está disponible el código fuente y la visualización de resultados.

II. Desarrollo

La detección de objetos consiste en localizar objetos de interés sobre una imagen. La localización consiste generalmente en predecir un rectángulo (a.k.a. bounding box) definido por un punto de referencia (x,y) y el ancho y alto del rectángulo. Así, no debería ser difícil darnos cuenta que el reconocimiento de montos puede reducirse a un problema de detección de objetos, en donde los objetos de interés son los dígitos.

A. Elección de Modelo de Detección

Para resolver este problema se disponía de los siguientes modelos:

- RetinaNet.
- TridentNet.
- FCOS.
- YOLO versión 2 o superior.

Se eligió YOLO (versión 5) por sobre las demás opciones por las siguientes razones:

- Velocidad de ejecución: Desde su introducción [1], las mediciones publicadas indican que su velocidad de detección es superior al de otros algoritmos similares en el año en que se probó. O por lo menos, otorga un buen *trade-off* entre precisión y eficiencia. Las tablas I y II indican estas comparaciones desde la fuente original. Estas corresponden a la versión del modelo 1 y 2 respectivamente.
- Debido a esta mayor eficiencia, nos fue posible probar más combinaciones de hiperparámetros y así poder optimizar aún más.
- Se eligió la versión 5 [3] porque fue probada con éxito en clases.

TABLA I
Comparación FPS de primera versión de YOLO [1]

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

TABLA II
Comparación FPS de YOLO9000 [2]

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

B. Preparación de datos

La implementación del modelo YOLO v5 requiere como primer paso, la conversión de las coordenadas de las

anotaciones desde el formato *[class_idx, xmin, ymin, width, height]* que viene con el dataset de origen, a un formato *[class_idx, x_center, y_center, width, height]*, donde las coordenadas están normalizadas por el tamaño de la imagen.

Para esta conversión utilizamos el código empleado en clases [5] y el resultado es como el ejemplo detallado a continuación.

- Notación antigua

```
4: 83, 8, 43, 53

6: 143, 11, 39, 50

0: 204, 11, 43, 43

6: 264, 15, 43, 43

6: 329, 11, 36, 51
```

- Notación nueva

```
4 0.2301762114537445 0.4928571428571429
  0.0947136563876652 0.7571428571428571

6 0.3579295154185022 0.5142857142857142
  0.08590308370044053 0.7142857142857143

0 0.4966960352422907 0.4642857142857143
  0.0947136563876652 0.6142857142857143

6 0.6288546255506607 0.5214285714285715
  0.0947136563876652 0.6142857142857143

6 0.76431718061674 0.5214285714285715
  0.07929515418502203 0.7285714285714285
```

Finalmente, se realizó la división del dataset de origen en un conjunto de entrenamiento y uno de validación, con proporción 90% - 10% respectivamente. Por lo tanto, del total de 6506 imágenes, 5855 se utilizarán para el entrenamiento y 651 para la validación.

C. Entrenamiento

En esta sección se siguen utilizando los pasos de [5]: se obtiene la implementación de YOLO v5 desde GitHub [3] y se crea un nuevo archivo de configuración *digits.yaml* con algunas rutas necesarias y la descripción de las 10 clases (1 clase por dígito):

```
# digits.yaml
train: /content/data/orand-car-with-bbs/training/train.txt
val: /content/data/orand-car-with-bbs/training/test.txt
# number of classes
nc: 10
# class names
names: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

El entrenamiento propiamente tal requiere solo 1 línea *Colab* y requiere los siguientes parámetros. Se utilizaron los mismos de [5].

- *--weights*: pesos pre-entrenados iniciales. Se elige yolov5s, el cual es casi el más rápido y simple dentro de los que se ofrecen en el repositorio.
- *--img*: Tamaño de la imagen. Se utilizaron 640 píxeles, porque es lo que requiere el argumento mencionado anteriormente.
- *--batch*: Tamaño del *batch*. Se utilizó un batch de 16.
- *--epochs*: Número de épocas. Se utilizaron 10 inicialmente.
- *--data*: Archivo de configuración recién creado.

```
!python train.py --weights yolov5s.pt --img 640
--batch 16 --epochs 10 --data digits.yaml
```

D. Testing

La detección sobre el conjunto de test se realizó ejecutando el archivo detect.py del repositorio de YOLO v5 utilizando los siguientes parámetros:

- *--source:* Ruta de la carpeta contenedora de imágenes a detectar.
- *--weights:* Ruta de la carpeta contenedora de los pesos obtenidos en el entrenamiento.
- *--conf-thres (confidence threshold)*: Nivel de confianza que se requiere para las detecciones.
- *--iou-thres (NMS IoU threshold)*: Intersección sobre la unión, filtro de detecciones para un mismo objeto.
- *--save-txt:* Guarda resultados en un archivo de texto.

```
!python detect.py --source
/content/data/orand-car-with-bbs/test/images --weights
runs/train/exp/weights/best.pt --conf-thres 0.5
--iou-thres 0.1 --save-txt
```

El nivel de confianza utilizado fue de 0.5 y NMS de 0.1. Para llegar a estos valores se realizaron iteraciones probando diferentes valores y estos fueron los que entregan mejores resultados.

La detección genera una carpeta *detect*, la cual contiene las imágenes del conjunto de test etiquetadas con las predicciones y además, archivos de texto asociados a cada imágen con la información de las predicciones.

La información de las predicciones se muestran en el siguiente formato:

<clase> <x>, <y>, <w>, <h> donde:

- **clase**: es la clase del dígito predecida con valores que van desde 0 hasta 9.
- **x**: coordenada x de la esquina superior izquierda del bounding box.
- **y**: coordenada y de la esquina superior izquierda del bounding box.
- **w**: ancho del respectivo bounding box.
- **h**: alto respectivo bounding box.

E. Accuracy conjunto de test

Se implementó una clase *Metrics*, en la cual existen dos funciones, una que realiza el cálculo de accuracy sobre el conjunto de test y otra que muestra los resultados negativos para facilitar el análisis de por qué el modelo falló en esos casos.

Para el cálculo de *accuracy*, se divide la cantidad de casos positivos en el total de casos. La contabilización de casos positivos se obtuvo comparando el monto detectado con el monto real, si éstos coinciden entonces significa que es un caso positivo.

El monto real se obtuvo desde el archivo list.txt disponible en el dataset inicial. Para simplificar su uso en el código, se realizó una copia de dicho archivo al repositorio de YOLO v5 en la carpeta runs. Luego, se utilizó un *dataframe* para poder identificar de mejor manera los datos del documento de texto.

El monto detectado se obtuvo de los archivos de texto generados en la detección, se revisó cada archivo de texto como dataframe, de tal manera que las clases se pudiesen ordenar respecto a la coordenada x para luego concatenar cada clase y formar el monto correspondiente.

La iteración se realizó por cada archivo de detección donde se comparan los montos y se almacenan los casos positivos y los negativos. Una vez finalizada la iteración se calcula el accuracy.

Para mostrar los resultados negativos, se recorre cada caso negativo y se genera una figura que muestra el manuscrito del monto correspondiente y como título se detalla el monto real y el monto detectado.

III. Resultados Experimentales y Discusión

A continuación se detallan los resultados obtenidos en el conjunto de validación entregados por YOLO v5, los resultados obtenidos en el conjunto de test calculados por nosotros y por último la discusión de lo anteriormente expuesto.

A. Resultados conjunto de validación

Se obtuvo un 97.168% de *precision* y 97.937% de *recall*, el comportamiento durante las 10 épocas se observa en Fig 1.

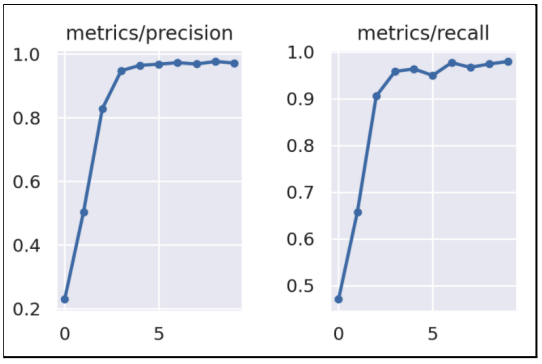

Fig. 1 Gráfico de métricas *precision* y *recall* durante 10 épocas del entrenamiento

Se observa que en términos de *precision*, con 5 épocas se alcanza un valor óptimo similar a las 10 épocas. Con respecto al *recall*, se considera que el valor alcanzado a las 10 épocas es mejor que si se hubieran utilizado 5 épocas. Por lo tanto, se definió que 10 épocas sería lo óptimo para la realización del entrenamiento.

Como sabemos, las métricas de *precision* y *recall* están relacionadas de manera directa y podemos visualizar su comportamiento en una curva, visualizada en la Fig.2.

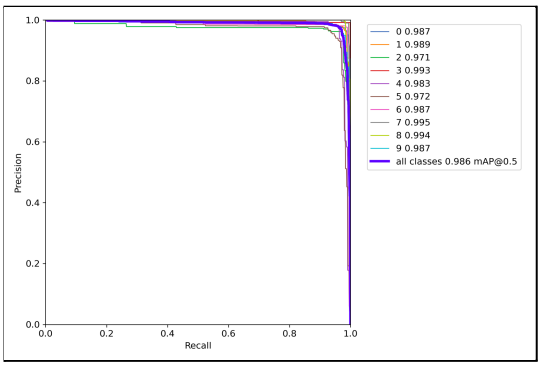

Fig. 2 Gráfico curva PR

Se observa que el rendimiento del modelo es bastante bueno ya que la curva se acerca notoriamente a la esquina superior derecha.

En la Fig. 3 se visualiza la matriz de confusión obtenida, la cual nos permite observar de forma explícita el comportamiento entre las clases.

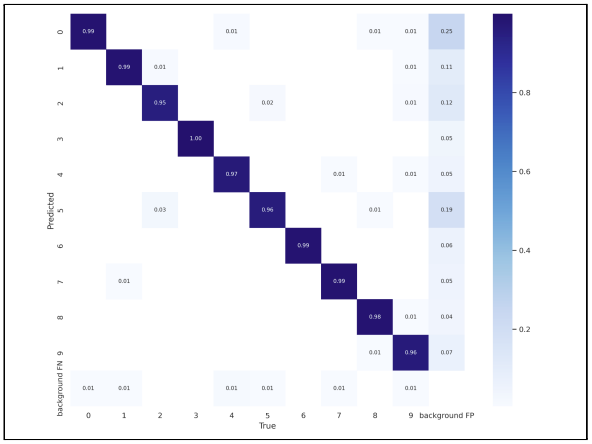

Fig. 3 Matriz de Confusión

En general se observa un buen comportamiento entre las clases, pero los dígitos que presentan más variabilidad en la clasificación serían el 2, 5 y 9. El dígito 2 suele confundirse con un 5 y 1, el dígito 5 con el 2 y el dígito 9 con el 0, 1, 2, 4 y 8.

El modelo de YOLO v5 genera las imágenes etiquetadas con cada predicción en el conjunto de validación, a continuación en la Fig. 4 se muestra un ejemplo de estas imágenes.

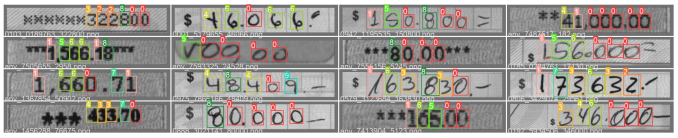

Fig. 4 Imágenes con etiquetas de montos detectados

Se observa que los montos predichos acertaron en su totalidad.

B. Resultados conjunto de test

De un total de 463 archivos, 398 fueron casos positivos, por tanto se obtuvo un **85.96%** de *accuracy*.

Se identificó que los 65 casos negativos fueron mal detectados por diversas razones, las cuales se detallan a continuación con algunos ejemplos.

i. La mala calidad o poca visibilidad de imagen del monto manuscrito (Fig. 5)

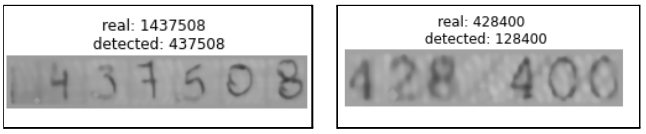

Fig. 5 Montos mal detectados por poca visibilidad del monto real.

ii. La presencia de símbolos previo o posterior al monto manuscrito. (Fig. 6)

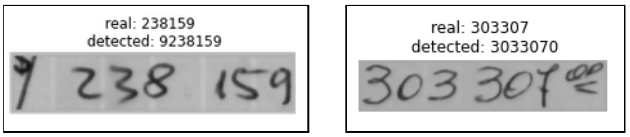

Fig. 6 Montos mal detectados por símbolos previo/posterior al monto.

iii. La forma de los números puede variar según la tipografía. (Fig. 7)

Fig. 7 Montos mal detectados por identificar el dígito 6 como 8 debido a la tipografía.

iv. Se encontraron dos casos en el que el monto real no corresponde a la imagen, fueron mal etiquetados. (Fig. 8)

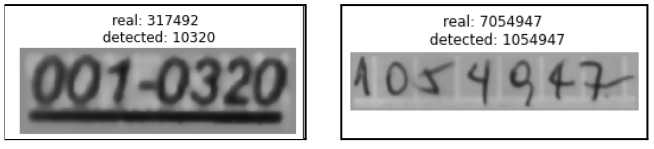

Fig. 8 Montos mal detectados por etiquetado incorrecto.

v. El modelo falla cuando el monto presenta gran cantidad de ceros correlativos, la detección omite uno o varios. (Fig. 9)

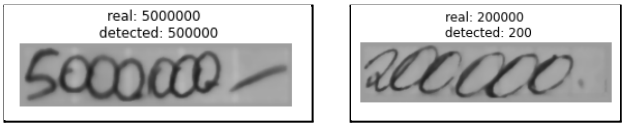

Fig. 9 Montos mal detectados por omisión de ceros.

vi. Hay nueve casos donde la detección entrega dos clases para un mismo dígito a pesar de haber utilizado un *iou-thres* en 0.1. Por ejemplo en la Fig. 10 podemos observar que el monto detectado 9133**25** y el 1229**25**1 fue erróneo ya que ese **25** destacado corresponde a dos clases, que puede ser un 2 o un 5. En todos los casos ocurre entre los dígitos 2 y 5.

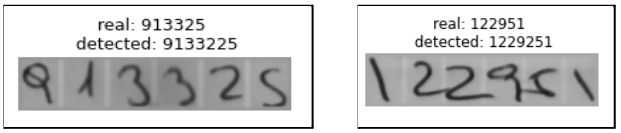

Fig. 10 Montos mal detectados por intersección de clases.

La visualización de todos los casos negativos y el detalle de las coordenadas para los casos de falla por intersección se encuentran disponibles en el notebook de Google Colab [4].

IV. Conclusiones

Se concluye que el modelo entrenado con YOLO v5 alcanzó un nivel bastante bueno, ya que, considerando la complejidad del problema de reconocimiento de dígitos y en el orden correcto, se alcanzó un 85.96% de accuracy sobre el conjunto de test.

Las métricas obtenidas en el entrenamiento, tanto *precision* y *recall* nos demuestran que la utilización de 10 épocas fue suficiente para alcanzar los resultados señalados en el documento.

Las clases con mayor error en la detección fueron los dígitos 2 y 5. Esto se puede observar tanto en la matriz de confusión como en los casos negativos detectados en el conjunto de test. El modelo se confunde entre estos dos dígitos y ésto es más frecuente que con los demás. Podría deberse a la forma en que las personas suelen escribirlos en manuscrito.

La calidad de las imágenes a detectar y su etiquetado correspondiente juegan un rol importante a la hora de obtener buenos resultados. En la ciencia de datos esto suele ser un problema recurrente y es esencial tenerlo en cuenta a la hora de entrenar cualquier modelo. En este caso en particular identificamos que parte de los errores se relaciona con lo descrito anteriormente.

Se recomienda utilizar otros dataset de dígitos manuscritos para comprobar la efectividad del modelo en mayor escala.

Finalmente, como trabajo futuro se propone utilizar modelos pre-entrenados de mayor complejidad y verificar una mejora en la *accuracy* obtenida. También es de nuestro interés la comparación con otras redes convolucionales utilizando el mismo set de datos. Para verificar que YOLO v5 mantiene el

menor tiempo de ejecución y también comparar la precisión de las detecciones entre todos los tipos de redes en cuestión.

Referencias

[1] Redmon, Joseph, Santosh Divvala, Ross Girshick, y Ali Farhadi. 2015. "You only look once: Unified, real-time object detection". En Proceedings of the IEEE conference on computer vision and pattern recognition, 779–88. https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html.

[2] Redmon, Joseph, y Ali Farhadi. 2016. "YOLO9000: Better, Faster, Stronger". arXiv [cs.CV]. arXiv. http://arxiv.org/abs/1612.08242.

[3] Jocher, Glenn. 2020. "YOLOv5 GitHub". GitHub. 29 de mayo de 2020. https://github.com/ultralytics/yolov5.

[4] Canales, Neira. 2021. "Tarea 2: Object detection YOLOv5". https://colab.research.google.com/drive/1cCXyeA3EmyOGdn-agLAkrTP-7a2mg7d0?usp=sharing.

[5] Loyola. "Clase 6: detección de objetos". https://colab.research.google.com/drive/1tKjg4N_jXidYcDdf8ee2--g_hgQlD2yf.