



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA

# QR DETECTOR:

Fabio A. Gonzalez  
Introducción a los sistemas inteligentes  
2020-I

---

## Integrantes

---

1. Luis Alejandro Higuaran Serrano.
2. Elmar Santofimio Suarez.
3. Dave Sebastián Valencia Salazar.

## Entrega

---

### 1. TÍTULO:

Proyecto: Detección de objetos tipo código QR

### 2. PROBLEMÁTICA:

Llevar a cabo la construcción de un modelo basado en aprendizaje de máquina, el cual permita detectar objetos específicos (*códigos QR*) en una imagen de forma eficiente.

### 3. PLANEACIÓN DEL PROYECTO:

Al iniciar el proyecto, como equipo de trabajo se estuvo debatiendo sobre la forma de trabajo y al final se decidió formular los siguientes pasos para ejecutar una iteración rápida bajo los lineamientos presentados en el curso “Structuring Machine Learning Projects”.

Se empieza con el estudio del dataset publicado usando visualizaciones tales como diagramas de caja y diagramas de barra, esto con el fin de tener una mayor comprensión del material con el que se trabaja, después de eso se decanta por usar la herramienta LabelBox para realizar el proceso de anotación de las zonas de QR, y poder tenerlas en el conjunto de datos de entrenamiento después de separar el dataset en *train / dev sets*, después realizar una investigación sobre el estado del arte sobre la detección de objetos en imágenes y luego decidir por el más conveniente bajo criterios de tiempo y complejidad de implementación y documentación asociada; finalmente después de eso, ajustar el dataset en base a las necesidades de la arquitectura escogida y finalmente realizar las tareas de implementación, entrenamiento, predicción y métricas para su respectiva evaluación.

De forma sencilla se lista de la siguiente manera:

- I. Análisis visual del dataset
- II. Realización de anotaciones requeridas
- III. Separación del conjunto de datos en *train / dev sets*
- IV. Investigación sobre el estado del arte.
- V. Discusión y decisión sobre arquitectura base del modelo
- VI. Implementación modelo
- VII. Entrenamiento y predicción modelo
- VIII. Evaluación modelo

### 4. EXPLORACIÓN DE DATOS Y ANÁLISIS:

Para empezar, el dataset utilizado representa un conjunto de datos el cual posee una cantidad variable de objetos de código QR, de forma inmediata se utilizó la herramienta LabelBox en la cual se permite realizar etiquetas(anotaciones) de objetos en imágenes de forma manual y tener como resultado el dataset con dichas etiquetas, más adelante se exportó el conjunto de datos con sus respectivas anotaciones y dejarlas a la espera para la decisión a nivel arquitectónico.

### 5. DETALLES DEL MODELO:

Ahora, la decisión tomada a nivel arquitectónico fue utilizar la arquitectura YOLO, por su versatilidad a nivel de aplicación, facilidad de implementación y

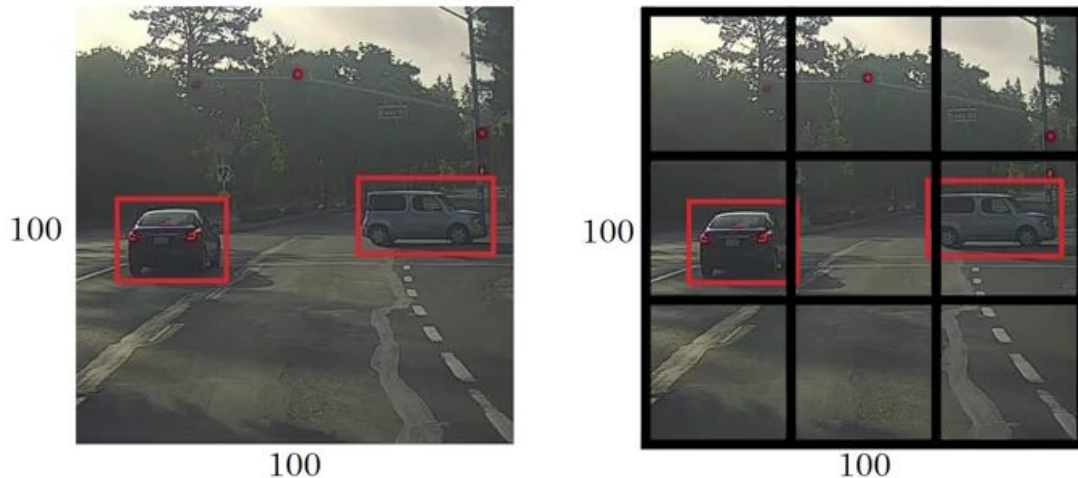
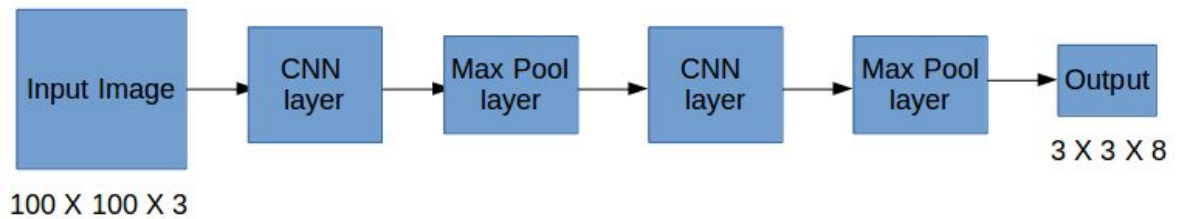
configuración; antes de explicar su fundamento, es imperante mencionar que para este punto, la planeación conlleva a una dificultad al tener que realizar un arreglo bastante arduo al dataset para que pueda ser utilizado por YOLO como entrada, retrasando bastante el progreso de la iteración y esto se debe a que la herramienta LabelImg ya realiza este proceso de forma automática.

Como se mencionó anteriormente, la implementación de los labels en YOLO era necesario realizar un paso adicional, ya que al utilizar LabelBox el formato de salida no era compatible con el requerido por el framework. Para esta tarea se desarrolló un script en python que tomaba los datos en .json generados por LabelBox, luego iterar sobre ellos hasta encontrar las medidas y los puntos en específico de las etiquetas(en este punto no normalizados), calculaba el centro y normalizaba las medidas, y ya como ultimo paso los asignaba a un archivo .txt por cada imagen de entrenamiento.

YOLO espera en sus sintaxis la siguiente secuencia: `<object-class>`  
`<x_center>` `<y_center>` `<width>` `<height>`

- Object-class: Clase del objeto
- x\_center: Coordenada normalizada x del centro de la caja
- y\_center: Coordenada normalizada y del centro de la caja
- width: Ancho normalizado de la caja
- height: Altura normalizada de la caja

Ahora, YOLO funciona de la siguiente manera, en vez de aplicar la estrategia de clasificadores para zonas con alta probabilidad de ser un objeto, este aplica una red neuronal a toda la imagen de entrada, dividiéndola en regiones y se toma la tarea de predecir las zonas de borde y asignando probabilidades (a modo de puntaje de confianza) a cada región, todo esto enmarcado en la extracción de características.



Por último dentro de la arquitectura de YOLO al momento de entrenar y predecir, las salidas de la capa de la red neuronal convolucional se dan de la siguiente manera

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

- pc: Define la probabilidad de que haya un objeto de cualquier clase en una cuadrícula
- bx,by,bh,bw: Especifica la caja si hay un objeto
- c1, c2, c3....cn: Representa las clases y se representa con 1 si la clase existe la imagen y 0 si no.

## 6. CONFIGURACIÓN EXPERIMENTAL:

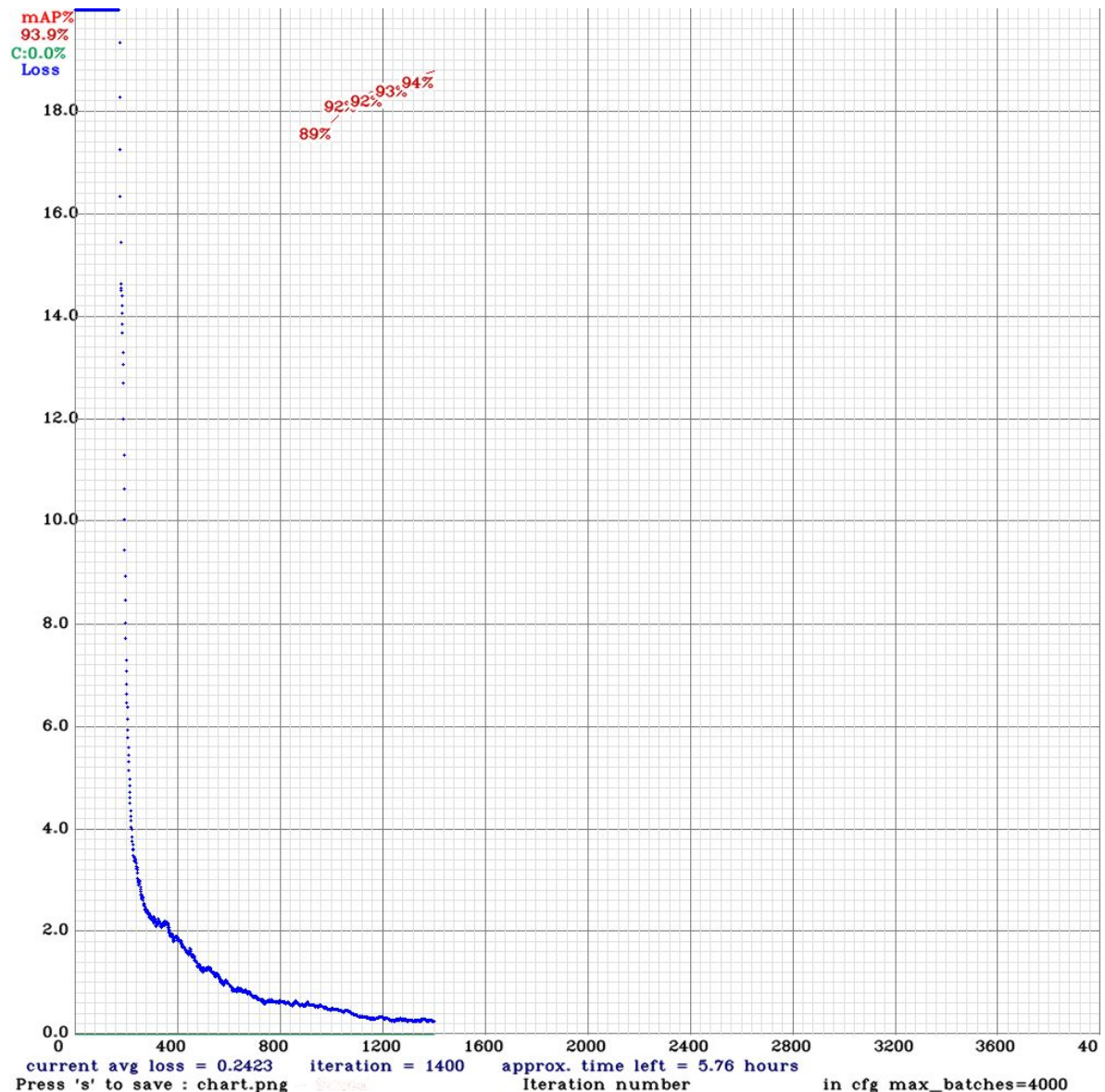
Primero cuadramos el entorno de desarrollo, la idea era poder acoplar el Colab que nos brinda Google a las necesidades del proyecto, además para el entrenamiento de esta. YOLO fue creado en Darknet, un framework de redes

neuronales de código abierto para entrenar detectores. El framework fue bajado y compilado para poder usar YOLO y entrenar sobre este, antes de entrenar se modificaron los archivos de configuración del framework y de YOLO a modo de hiperparametros para acoplar la arquitectura al proyecto y además obtener resultados veraces en poco tiempo, teniendo en cuenta que estas redes toman días de entrenamiento y muchas de estas usan más de 1 clase.

Los hiperparametros que tomamos fueron los siguientes:

- Imagenes validación: 15% \* total imágenes
- Uso de GPU: GPU = 1 (True)
- Uso de CUDNN (NVIDIA CUDA Deep Neural Network library): CUDNN = 1 (True)
- Uso de OPENCV: OPENCV = 1 (True)
- Ancho cuadrícula YOLO: width = 416
- Altura cuadrícula YOLO: height = 416
- Máximo número de Batches: max\_batches = 4000
- Pasos: steps = 5400
- Filtros: filters = 18 (filters = (classes + 5) \* 3)
- Clases: classes = 1
- Límite: threshold = 0.3
- Backup pesos: si es menor a 1000 cada 100 epochs, de lo contrario cada 1000

Para el proceso de entrenamiento usamos una red neuronal convolucional pre entrenada llamada darknet53.conv.74, que nos brindaba un modelo base para nuestro proyecto, así mismo generamos los archivos yolo.name y yolo.data que servian para indicarle a YOLO datos y clases iba a tener el modelo. YOLO a medida que entrenaba generaba los pesos de las iteraciones y así mismo un cuadro con los resultados y métricas del entrenamiento. Como podemos ver a medida que YOLO entrena el mAP va aumentando y así mismo el loss va disminuyendo.



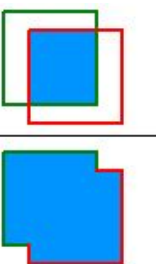
Resultados obtenidos de la fase de entrenamiento/validación de YOLO, mAP promedio = 93.9%, avg loss = 0.2423, mejor mAP = 94%, peor mAP = 89%

## 7. RESULTADOS:

Para la parte de predicción y evaluación del modelo usamos el modulo DNN o Deep Neural Network de OpenCV. Este nos entrega las clases y las probabilidades de las mismas. Usando un threshold, ignorábamos las probabilidades inferiores a 0.3, luego volvíamos a predecir para asegurar mejores resultados y nos aseguramos de al menos obtener un objeto en la imagen. Luego obtenemos los boxes, clases y confidences de cada uno de los objetos identificados.

Usamos 15 imágenes del dataset original para las métricas y explicación de resultados. Para ello usamos el proyecto o framework Object-detection-metrics que usa interpolación en todos los puntos o métricas para VOC PASCAL (Curva Precisión x Recuperación y Precisión promedio). El framework se base la intersección sobre la unión o (IOU), es una medida basada en el índice Jaccard que evalúa la superposición entre dos cuadros delimitadores

Aplicando IOU podemos saber si una detección es válida (Verdadero positivo) o no (Falso positivo), El IOU más alto viene dado por el área superpuesta entre el cuadro delimitador predicho y el cuadro delimitador de la verdad fundamental dividido por el área de unión entre ellos:

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}$$


- Verdadero Positivo (TP) : Una detección correcta. Detección con  $IOU \geq \text{umbral}$
- Falso positivo (FP) : una detección incorrecta. Detección con  $IOU < \text{umbral}$
- Falso negativo (FN) : una verdad fundamental no detectada
- Verdadero negativo (TN) : no se aplica. Representaría una detección errónea corregida. En la tarea de detección de objetos, hay muchos cuadros delimitadores posibles que no deberían detectarse dentro de una imagen. Por lo tanto, TN sería todos los cuadros delimitadores posibles que no se detectaron correctamente (tantos cuadros posibles dentro de una imagen). Es por eso que no es utilizado por las métricas.

Para nuestro caso el umbral fue de 0.3\*

Al final obtuvimos los siguiente resultados:

AP: 89.02% (qrcode) precisión media

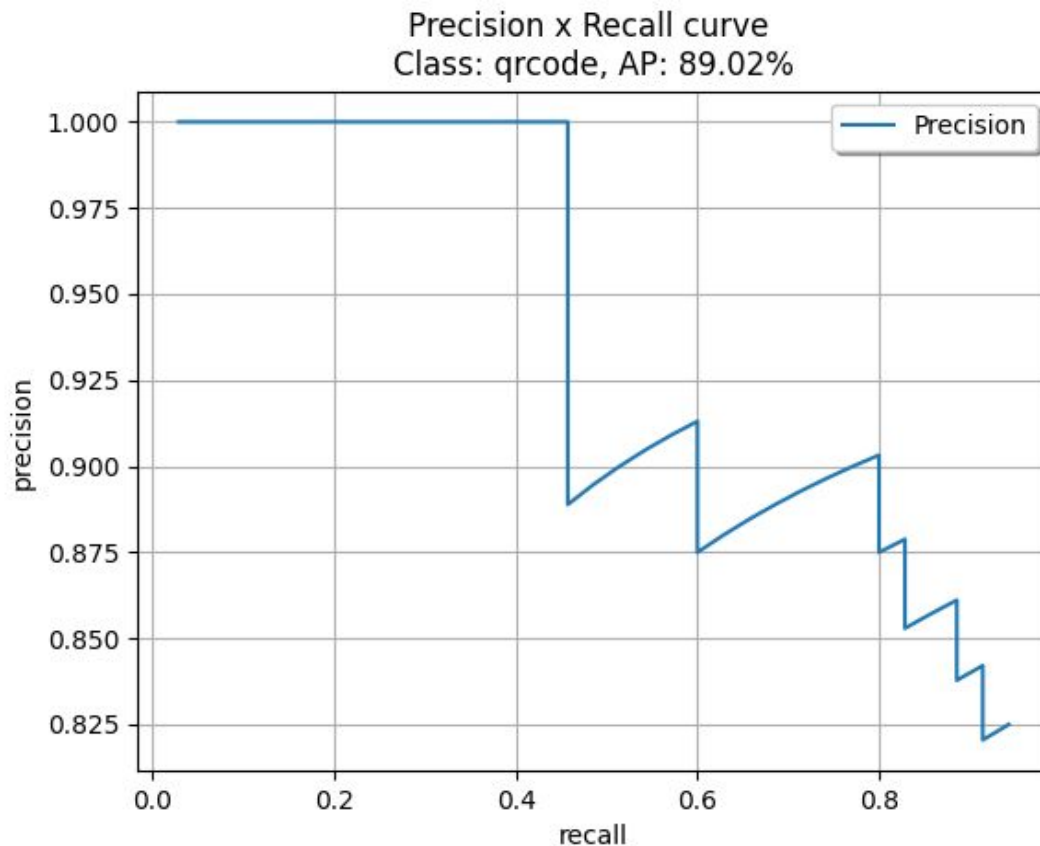
mAP: 89.02% precisión media ponderada

Class: qrcode

AP: 89.02%

Precision: ['1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '1.00', '0.94', '0.89', '0.89', '0.90', '0.90', '0.91', '0.91', '0.88', '0.88', '0.88', '0.89', '0.89', '0.90', '0.90', '0.90', '0.88', '0.88', '0.85', '0.86', '0.86', '0.84', '0.84', '0.82', '0.82']

Recall: ['0.03', '0.06', '0.09', '0.11', '0.14', '0.17', '0.20', '0.23', '0.26', '0.29', '0.31', '0.34', '0.37', '0.40', '0.43', '0.46', '0.46', '0.46', '0.49', '0.51', '0.54', '0.57', '0.60', '0.60', '0.63', '0.66', '0.69', '0.71', '0.74', '0.77', '0.80', '0.80', '0.83', '0.83', '0.86', '0.89', '0.89', '0.91', '0.91', '0.94']



Como podemos observar el modelo es bueno y con resultados satisfactorios, nos da a entender que el modelo mantiene en un buen tramo una alta precisión mientras aumenta la recuperación, dando a entender que si aumentamos la confianza al momento de predecir obtendremos resultados iguales o similares al anterior. Así mismo la precisión ponderada en todos los valores de recuperación entre 0 y 1 nos da 89.02 , un poco inferior al 93.02 de la fase de entrenamiento pero aun así, un buen resultado para ser la primera versión del proyecto.

## 8. CONCLUSIONES:

De acuerdo a las decisiones tomadas y los resultados vistos se llega a las siguientes conclusiones:

1. Una de las desventajas de la arquitectura YOLO, es su alta sensibilidad a la relación entre el tamaño del objeto a detectar y el tamaño total de la imagen, si esta relación es suficientemente grande es descartada para su uso.



2. Para proyecto futuros de IA que empiecen de cero, se recomienda trabajar las iteraciones bajo los siguientes lineamientos:

- I. Análisis visual del dataset.
- II. Investigación sobre el estado del arte.
- III. Discusión y decisión sobre arquitectura base del modelo.
- IV. Realización de anotaciones requeridas.
- V. Separación del conjunto de datos en *train / dev sets*
- VI. Implementación modelo.
- VII. Entrenamiento y predicción modelo.
- VIII. Evaluación modelo.

Cabe aclarar que para la segunda iteración en adelante, los pasos II. y III. se deben reemplazar por un análisis de error, y actuar de acuerdo a los resultados del análisis.

## 9. REFERENCIAS, GUÍAS Y LIBRERÍAS USADAS:

- <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
- <https://github.com/AlexeyAB/darknet>
- <https://github.com/rafaelpadilla/Object-Detection-Metrics>
- <https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-on-yolo-framework-python/#:~:text=In%20YOLO%2C%20the%20coordinates%20assigned,with%20respect%20to%20this%20grid.>
- <https://medium.com/@quangnhatnguyenle/how-to-train-yolov3-on-google-colab-to-detect-custom-objects-e-g-gun-detection-d3a1ee43eda1>
- [https://www.youtube.com/watch?v=\\_FNfRtXEbr4](https://www.youtube.com/watch?v=_FNfRtXEbr4)
- <https://labelbox.com/>
- [https://es.wikipedia.org/wiki/Redes\\_neuronales\\_convolucionales](https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales)

## 10. ANEXOS:

- Vídeo explicación poster: <https://youtu.be/14k9OBg8fSA>
-