Object Detection using YOLO

YOLO Algorithm

YOLO or You Only Look Once, is a popular real-time object detection algorithm. YOLO combines what was once a multi-step process, using a single neural network to perform both classification and prediction of bounding boxes for detected objects. As such, it is heavily optimized for detection performance and can run much faster than running two separate neural networks to detect and classify objects separately. It does this by repurposing traditional image classifiers to be used for the regression task of identifying bounding boxes for objects.

YOLO is based on the idea of segmenting an image into smaller images. The image is split into a square grid of dimensions S×S. The cell in which the center of an object, for instance, the center of the dog, resides, is the cell responsible for detecting that object. Each cell will predict B bounding boxes and a confidence score for each box. The default for this architecture is for the model to predict two bounding boxes. The classification score will be from `0.0` to `1.0`, with`0.0` being the lowest confidence level and `1.0` being the highest; if no object exists in that cell, the confidence scores should be `0.0`, and if the model is completely certain of its prediction, the score should be `1.0`. These confidence levels capture the model’s certainty that there exists an object in that cell and that the bounding box is accurate. Each of these bounding boxes is made up of 5 numbers: the x position, the y position, the width, the height, and the confidence. The coordinates ‘(x, y)’ represent the location of the center of the predicted bounding box, and the width and height are fractions relative to the entire image size. The confidence represents the IOU between the predicted bounding box and the actual bounding box, referred to as the ground truth box. The IOU stands for Intersection Over Union and is the area of the intersection of the predicted and ground truth boxes divided by the area of the union of the same predicted and ground truth boxes.

In addition to outputting bounding boxes and confidence scores, each cell predicts the class of the object. This class prediction is represented by a one-hot vector length C, the number of classes in the dataset. However, it is important to note that while each cell may predict any number of bounding boxes and confidence scores for those boxes, it only predicts one class. This is a limitation of the YOLO algorithm itself, and if there are multiple objects of different classes in one grid cell, the algorithm will fail to classify both correctly. Thus, each prediction from a grid cell will be of shape $\boldsymbol{C + B * 5}$, where $\boldsymbol{C}$ is the number of classes and $\boldsymbol{B}$ is the number of predicted bounding boxes. $\boldsymbol{B}$ is multiplied by 5 here because it includes ***(x, y, w, h, confidence)*** for each box. Because there are $\boldsymbol{S \times S}$ grid cells in each image, the overall prediction of the model is a tensor of shape $\boldsymbol{S \times S \times (C + B * 5)}$.

The YOLO model is made up of three key components: the head, neck, and backbone. The backbone is the part of the network made up of convolutional layers to detect key features of an image and process them. The backbone is first trained on a classification dataset, such as ImageNet, and typically trained at a lower resolution than the final detection model, as detection requires finer details than classification. The neck uses the features from the convolution layers in the backbone with fully connected layers to make predictions on probabilities and bounding box coordinates. The head is the final output layer of the network which can be interchanged with other layers with the same input shape for transfer learning. As discussed earlier, the head is an $\boldsymbol{S \times S \times (C + B * 5)}$ tensor and is $\boldsymbol{7 \times 7 \times 30}$ in the original YOLO research paper with a split size $\boldsymbol{S}$ of 7, 20 classes $\boldsymbol{C}$, and 2 predicted bounding boxes $\boldsymbol{B}$. These three portions of the model work together to first extract key visual features from the image then classify and bound them.

As discussed previously, the backbone of the model is pre-trained on an image classification dataset. The final layer, which predicts both class probabilities and bounding box coordinates, uses a linear activation function while the other layers use a leaky ReLU function.

Each grid cell predicts multiple bounding boxes, but only one bounding box is responsible for detecting the object. The responsible bounding box is determined by choosing the predicted bounding box with the highest IOU. The effect of this is that certain bounding boxes will improve at predicting certain shapes and sizes of bounding boxes while others will specialize in other shapes. This occurs because of the following: if there is a large object when the multiple bounding boxes predict bounds, the best one is rewarded and continues to improve predicting large boxes. When a small object comes up, the previous predictor fails at predicting a good fit as its bounding box is too large. However, another predictor has a better prediction, and it is rewarded for bounding the small object well. As training goes on, the predictions of various bounding boxes diverge to specialize in the tasks they were good at early in training time.

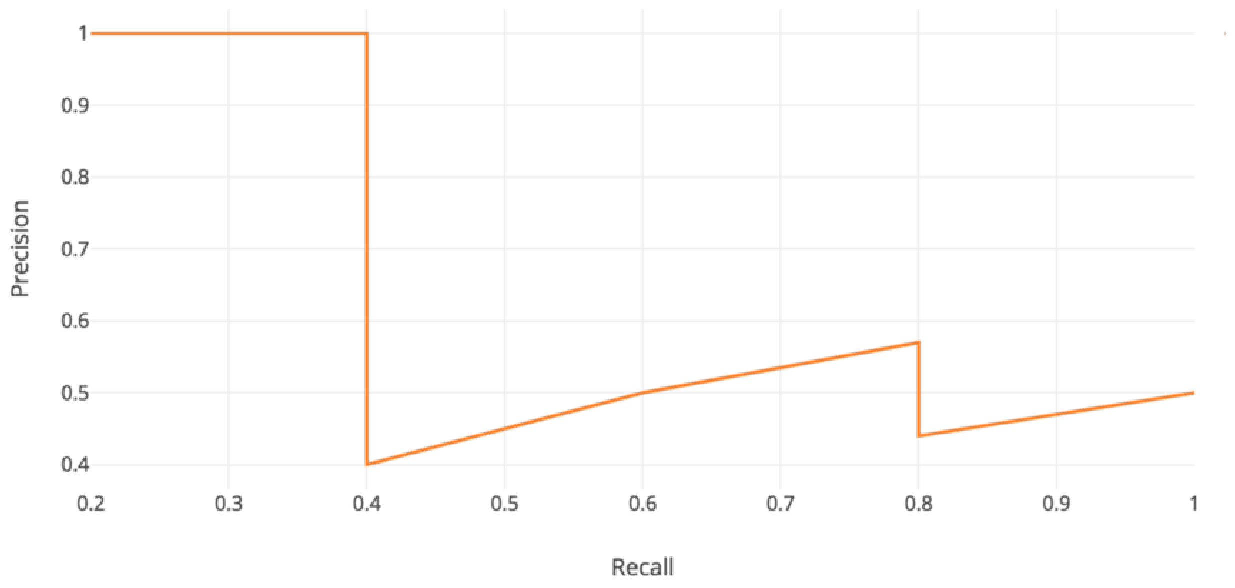
mAP Score

AP (Average precision) is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1.

The general definition for the Average Precision (AP) is finding the area under the precision-recall curve.

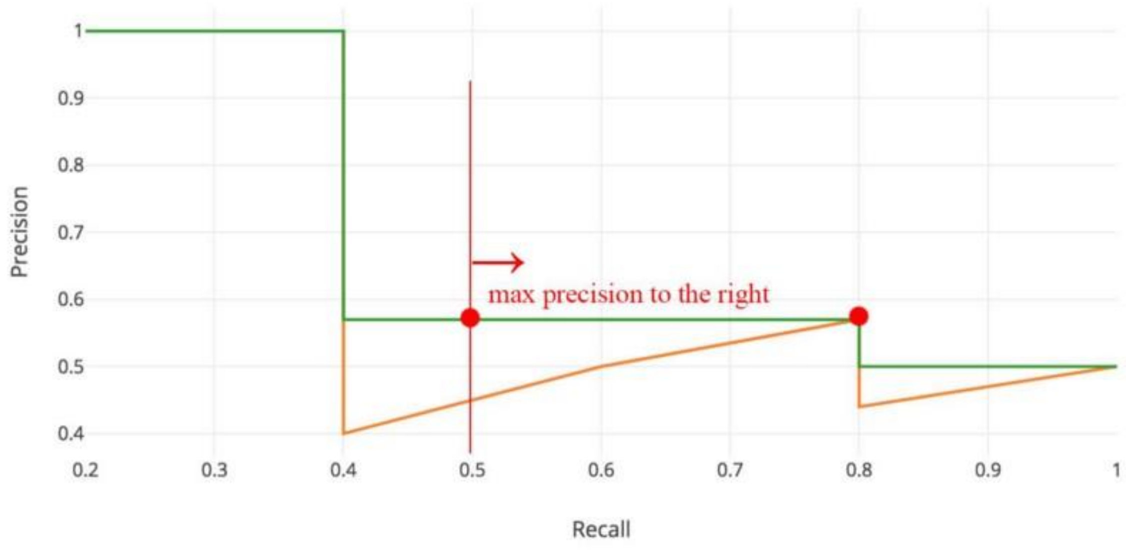
$$AP = \int_0^1 p(r)dr$$

A plot of precision against the recall value gives a zig-zag pattern like this



Precision and recall are always between 0 and 1. Therefore, AP falls within 0 and 1 also. Before calculating AP for the object detection, we often smooth out the zigzag pattern of the precision-recall curve first.

Graphically, at each recall level, we replace each precision value with the maximum precision value to the right of that recall level.



So, the orange line is transformed into the green lines and the curve will decrease monotonically instead of the zigzag pattern. The calculated AP value will be less suspectable to small variations in the ranking. Mathematically, we replace the precision value for recall $\hat{r}$ with the maximum precision for any recall $\geq \hat{r}$.

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$$

mAP (mean average precision) is the average of AP. In some context, we compute the AP for each class and average them. But in some context, they mean the same thing.

Detection Results Using Pre-Trained Model

For this part, I have used a pre-trained YOLOv5 model for object detection on a dataset containing images captured by a camera mounted on the hood of a car.

YOLOv5 is a family of object detection architectures and models pretrained on the COCO dataset, and represents Ultralytics open-source research into future vision AI methods, incorporating lessons learned and best practices evolved over thousands of hours of research and development.

The classes that the model can detect are:

‘person’, ‘bicycle’, ‘car’, ‘motorcycle’, ‘airplane’, ‘bus’, ‘train’, ‘truck’, ‘boat’, ‘traffic light’, ‘fire hydrant’, ‘stop sign’, ‘parking meter’, ‘bench’, ‘bird’, ‘cat’, ‘dog’, ‘horse’, ‘sheep’, ‘cow’, ‘elephant’, ‘bear’, ‘zebra’, ‘giraffe’, ‘backpack’, ‘umbrella’, ‘handbag’, ‘tie’, ‘suitcase’, ‘frisbee’, ‘skis’, ‘snowboard’, ‘sports ball’, ‘kite’, ‘baseball bat’, ‘baseball glove’, ‘skateboard’, ‘surfboard’, ‘tennis racket’, ‘bottle’, ‘wine glass’, ‘cup’, ‘fork’, ‘knife’, ‘spoon’, ‘bowl’, ‘banana’, ‘apple’, ‘sandwich’, ‘orange’, ‘broccoli’, ‘carrot’, ‘hot dog’, ‘pizza’, ‘donut’, ‘cake’, ‘chair’, ‘couch’, ‘potted plant’, ‘bed’, ‘dining table’, ‘toilet’, ‘tv’, ‘laptop’, ‘mouse’, ‘remote’, ‘keyboard’, ‘cell phone’, ‘microwave’, ‘oven’, ‘toaster’, ‘sink’, ‘refrigerator’, ‘book’, ‘clock’, ‘vase’, ‘scissors’, ‘teddy bear’, ‘hair drier’, ‘toothbrush’

References:

GitHub Repository: https://github.com/ultralytics/yolov5

Dataset: https://www.kaggle.com/sshikamaru/car-object-detection

Detection Results:

The model works successfully on the dataset and identifies objects in all the 418 images of the dataset.

To check the identification of the model, I have taken 10 images from the dataset and compared the objects identified by the model with the objects that are actually present in the image.

Image 1:

The objects detected are: 1 person, 1 car, 4 traffic lights

Actually, there are 1 car, 5 traffic lights and no person

A road sign is wrongly identified as a person and one traffic lights is not detected.

Image 2:

The objects detected are: 2 cars, 4 traffic lights

Actually, there are 2 cars and 4 traffic lights

So, the model correctly detects the objects in this image

Image 3:

The objects detected are: 1 car

Actually, there is 1 car

So, the model correctly detects the objects in this image

Image 4:

The objects detected are: 1 car, 5 traffic lights

Actually, there are 1 car, 5 traffic lights

So, the model correctly detects the objects in this image

Image 5:

The objects detected are: 6 cars

Actually, there are 6 cars and 2 traffic lights

The model could detect the 6 cars but could not detect the 2 traffic lights

Image 6:

The objects detected are: 4 cars, 2 traffic lights

Actually, there are 5 cars, 2 traffic lights

The model failed to detect one car

Image 7:

The objects detected are: 4 cars, 3 traffic lights

Actually, there are 5 cars, 4 traffic lights

The model failed to detect one car and one traffic lights

Image 8:

The objects detected are: 1 car

Actually, there is one car

So, the model correctly detects the objects in this image

Image 9:

The objects detected are: 1 car

Actually, there is 1 car

So, the model correctly detects the objects in this image

Image 10:

The objects detected are: 3 cars

Actually, there are 3 cars

So, the model correctly detects the objects in this image

Detection Results Using Fine-Tuned Model

For this part, I implemented the tiny version of YOLOv4 for training on a custom dataset.

YOLOv4-tiny is the compressed version of YOLOv4 designed to train on machines that have less computing power. Its model weights are around 16 megabytes large, allowing it to train on 350 images in 1 hour when using a Tesla P100 GPU. YOLOv4-tiny has an inference speed of 3 ms on the Tesla P100, making it one of the fastest object detection models to exist.

YOLOv4-Tiny utilizes a couple of different changes from the original YOLOv4 network to help it achieve these fast speeds. First and foremost, the number of convolutional layers in the CSP backbone are compressed with a total of 29 pretrained convolutional layers. Additionally, the number of YOLO layers has been reduced to two instead of three and there are fewer anchor boxes for prediction.

References:

Model: https://models.roboflow.com/object-detection/yolov4-tiny-darknet

Dataset: https://public.roboflow.com/object-detection/vehicles-openimages

Detection Results:

The dataset being used, Vehicle OpenImages Dataset, is split into train, test and valid parts.

The train part of the dataset is used to train the tiny YOLOv4 model, the valid part to validate and the test part to test.

The Number of Classes is 5.

The classes are Ambulance, Bus, Car, Motorcycle, Truck.

Results of Training:

```
 Calculation mAP (mean average precision) …
128
 detections_count = 659, unique_truth_count = 223
class_id = 0, name = Ambulance, ap = 83.41% (TP = 20, FP = 15)
class_id = 1, name = Bus, ap = 67.59% (TP = 27, FP = 13)
class_id = 2, name = Car, ap = 40.18% (TP = 47, FP = 46)
class_id = 3, name = Motorcycle, ap = 22.07% (TP = 5, FP =10)
class_id = 4, name = Truck, ap = 28.80% (TP = 9, FP = 10)
 for conf_thresh = 0.25, precision = 0.53, recall = 0.48, F1-score = 0.51
 for conf_thresh = 0.25, TP = 108, FP = 94, FN = 115, average IoU = 42.35 %
 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 Mean average precision (mAP@0.50) = 0.484108 or 48.41 %
```

The model successfully identifies classes in the test part of the dataset.

A random example from the test part shows a Bus identified correctly in an image.

Links to the Google Colab Notebooks

Part 1:

https://colab.research.google.com/drive/1ntgiF4_RPqALsDtwsMAKdAHPri-hjDZw?usp=sharing

Part 2:

https://colab.research.google.com/drive/1z-nFmTv5M1bE-6nyCOpfgIQVcxk4lZkg?usp=sharing