Yamini Kota
194102309

YOLO V3-“You Only Look Once” Architecture

EE-525
Project-2

Introduction

Object Detection is one of the cutting-edge topics in Computer Vision and Data Science. It recognizes “what” the object is and “where” it is located. Object Detection involves “Object Localization”, i.e., identifying the position of one or more objects in the image and “Image Classification”, i.e., prediction of the class of the object in the image. Many methods and algorithms have been proposed and have gained huge popularity for Object Detection like RCNN, Fast RCNN, YOLO, etc. However, YOLO has found its place in many real time applications due to its exceptional high speed. As the name suggests, YOLO processes the entire image in one forward pass to output multiple detected objects along with their confidence scores.
YOLO V3 is an improvement over previous YOLO detection networks. Compared to prior versions, it features multi-scale detection, stronger feature extractor network, and some changes in the loss function. As a result, this network can now detect many more targets from big to small. And, of course, just like other single-shot detectors, YOLO V3 also runs quite fast and makes real-time inference possible on GPU devices.

Architecture

YOLO V3 Architecture is shown in the figure below:

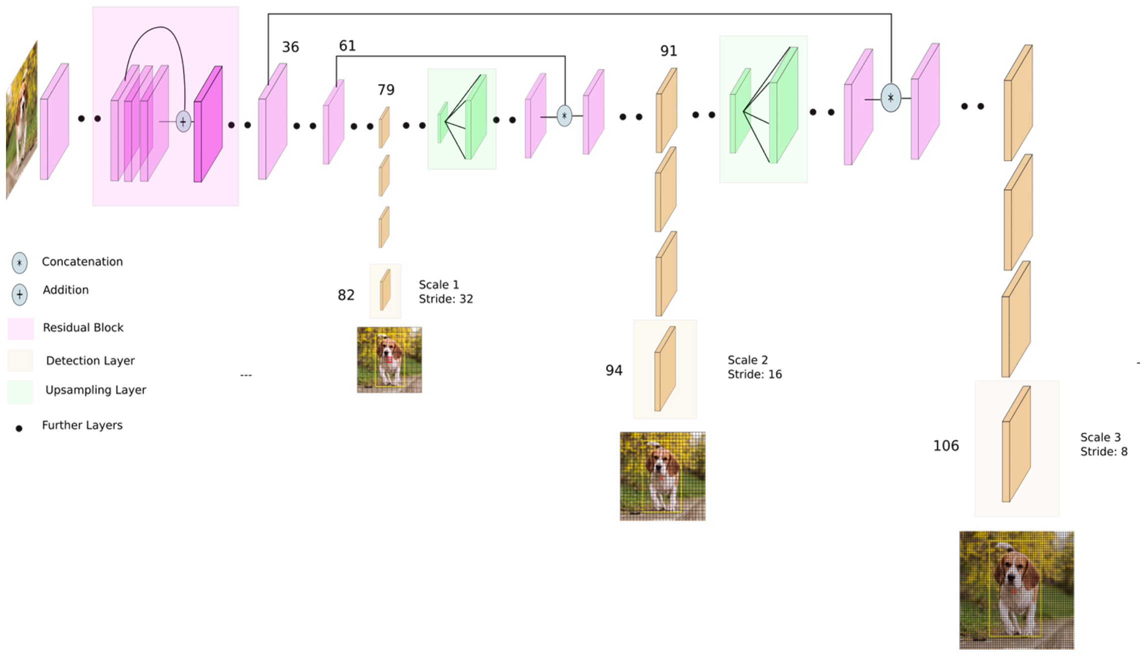
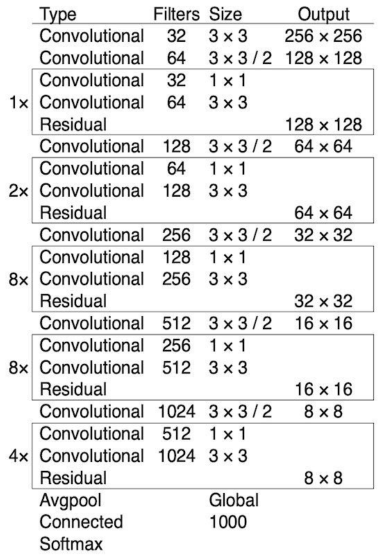


Figure 1: YOLO V3 Architecture

The whole system can be divided into two major components: Feature Extractor and Detector; both are multi-scale. When a new image comes in, it goes through the feature extractor first so that the feature embeddings at three (or more) different scales can be obtained. Then, these features are feed into three (or more) branches of the detector to get bounding boxes and class information.

Feature Extractor

The feature extractor YOLO V3 uses is called Darknet-53. The structure of Darknet-53 is shown below:



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2: Darknet-53

YOLOv2 uses Darknet-19, which mostly uses 3×3 filters to extract features and 1×1 filters to reduce output channels. It also uses global average pooling to make predictions. Darknet-53 borrows the idea of skip connections to help the activations to propagate through deeper layers without gradient diminishing from ResNet and successfully extends the network from 19 to 53 layers.
The whole network is a chain of multiple blocks with some strides 2 Convolutional layers in between to reduce dimension. Inside the block, there's just a bottleneck structure (1×1 followed by 3×3) plus a skip connection.
Consider layers in each rectangle as a residual block. Features from last three residual blocks are all used in the later detection as YOLO V3 is designed to be a multi-scaled detector. The input image size is 416×416, so three scale vectors would be 52×52, 26×26, and 13×13.

Multi-scale Detector

Once the three features vectors are obtained, they can be feed into the detector. The detection layer is used make detection at feature maps of three different sizes, having strides 32, 16, 8 respectively. These are done at 86^{th}, 94^{th} and 106^{th} layers respectively. Assuming the input image is $(416, 416, 3)$, the final output of the detectors will be in shape of $[(52, 52, 3, (4+1+num\_classes)), (26, 26, 3, (4+1+num\_classes)), (13, 13, 3, (4+1+num\_classes))]$. The three items in the list represent detections for three scales. The network downsamples the input image until the first detection layer (layer 86), where a detection is made using feature maps of a layer with stride 32. Further, layers are upsampled by a factor of 2 and concatenated with feature maps of a previous layers having identical feature map sizes. By doing so, small scale detection can also benefit from the result of large scale detection. Another detection is now made at layer 94 with stride 16. The same upsampling procedure is repeated, and a final detection is made at the layer 106 of stride 8. Anchor boxes concept explains what the cells in this $52 \times 52 \times 3 \times (4 + 1 + num\_classes)$ matrix mean.

Anchor Box

Anchor box is a prior box that could have different pre-defined aspect ratios. These aspect ratios are determined before training by running K-means on the entire dataset. Anchor boxes are used for faster convergence of the network.
When convolution kernels run in paraller, the output is a square matrix of feature values. This matrix is defined as a "grid". Anchor boxes anchor to each cell of the grids, and they share the same centroid. And once the anchors are defined, the overlap between the ground truth box and the anchor box can be determined and the one with the best IOU is picked and they are coupled together, where IOU is the "Intersection over Union" ratio. Now, instead of predicting coordinates from the wild west, the offsets to these bounding boxes can be predicted. This gives a good headstart in training. For each anchor box, 3 things are to be predicted:

1. The 4 location offsets against the anchor box: tx, ty, tw, th. (Note: These are not the actual coordinates of the bounding box.)
2. The objectness score to indicate if this box contains an object.
3. The num_class number of class probabilities to tell us which class this box belongs to.

In total, we are predicting $4 + 1 + num\_classes$ values for one anchor box, and therefore, the output is a matrix in shape of $52 \times 52 \times 3 \times (4 + 1 + num\_classes)$. In YOLO V3, there are three anchor boxes per grid cell, and three scales of grids. Therefore 9 anchors for 3 scales are provided.

Loss Function

The loss function consists of four parts:

1. Centroid Loss: It is the loss for bounding box centroid. tx and ty is the relative centroid location from the ground truth. tx' and ty' is the centroid prediction from the detector directly. The smaller this loss is, the closer the centroids of prediction and ground truth are. Mean square error is used here. Centroid loss is given by:

   $$xy\_loss = Lambda\_Coord * Sum(Mean\_Square\_Error((tx, ty), (tx', ty')) * obj\_mask)$$

   obj_mask indicates if there's an object or not, i.e., $obj\_mask = 1$ when the object is present, and zero otherwise. And $Lambda\_Coord$ is the weight for the loss in the boundary box coordinates. This is to be kept more than 1 (default: 5).

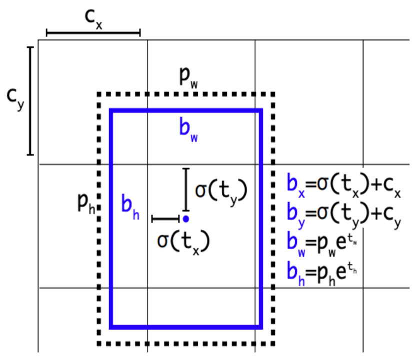
   

   Figure 3: Bounding boxes with priors and predictors

   The predictions correspond to:

   $$bx = sigmoid(tx) + Cx$$
   $$by = sigmoid(ty) + Cy$$

   where bx and by are the absolute values that are usually used as centroid location. As the centroids of the grid cell and anchor box are shared, the difference to grid cell can represent the difference to anchor box. Therefore, the centroid is predicted relative to the top-left corner of the grid cell. $sigmoid(tx)$ and $sigmoid(ty)$ are the centroid location relative to the grid cell. Cx and Cy represents the absolute location of the top-left corner of the current grid cell. These values of bx and by need to be normalized by dividing by the grid size. These formulas are inverted to get tx and ty.

2. Width and Height Loss: It is given by:

   $$wh\_loss = Lambda\_Coord * Sum(Mean\_Square\_Error((tw, th), (tw', th')) * obj\_mask)$$

Here we get tw and th by inverting the following:

$$bw = exp(tw) * pw$$
$$bh = exp(th) * ph$$

where bw and bh are absolute width and height to the whole image. And pw and ph are the width and height of the anchor box.

3. Objectness Score Loss: Objectness indicates how likely is there an object in the current cell. Binary cross-entropy is used instead of mean square error here. Objectness or no-objectness losses are given by:

$$obj\_loss = Sum(Binary\_Cross\_Entropy(obj, obj') * obj\_mask)$$
$$noobj\_loss =$$
$$Lambda\_Noobj * Sum(Binary\_Cross\_Entropy(obj, obj') * (1 - obj\_mask) * ignore\_mask)$$

   obj and obj' are the object scores, which represent the probability that an object is contained inside a bounding box. $noobj\_loss$ penalizes the false positives while training. False positives are masked with $1 - obj\_mask$. The $ignore\_mask$ is used to make sure predictions are only penalized when the current box doesn't have much overlap with the ground truth box. If the bounding box prior is not the best but does overlap a ground truth object by more than some threshold we ignore the prediction. And finally, $Lambda\_Noobj = 0.5$ to make sure the network won't be dominated by cells that don't have objects.

4. Classification Loss: Binary cross-entropy is applied for each class one by one and are summed up because they are not mutually exclusive. And then this is multiplied with obj_mask so that only those cells are counted that have a ground truth object. Classification loss is given by:

$$class\_loss = Sum(Binary\_Cross\_Entropy(class, class') * obj\_mask)$$

   $class$ and $class'$ are the one-hot encoding vectors of Class confidences, which represent the probabilities of the detected object belonging to a particular class. Sigmoid is used because if an object belongs to one class, then it cannot belong to another class.

All the losses are added to get the total loss:

$$total\_loss = Lambda\_Coord * Sum(Mean\_Square\_Error((tx, ty), (tx', ty')) * obj\_mask)$$
$$+ Lambda\_Coord * Sum(Mean\_Square\_Error((tw, th), (tw', th')) * obj\_mask)$$
$$+ Sum(Binary\_Cross\_Entropy(obj, obj') * obj\_mask)$$
$$+ Lambda\_Noobj * Sum(Binary\_Cross\_Entropy(obj, obj') * (1 - obj\_mask) * ignore\_mask)$$
$$+ Sum(Binary\_Cross\_Entropy(class, class') * obj\_mask)$$

Non-Maximal Suppression

YOLO can predict multiple bounding boxes for the same object while testing. Generally, boxes having scores below a threshold (for example below 0.5) are ignored. Then, Non-Maximal Suppression is used. Starting from the predictions with the highest confidence, discard current prediction if it has IOU>0.5 with any of the previous bounding boxes having the same class. Repeat this until all the predictions are checked. Non-Maximal Suppression ensures only one bounding box is predicted for each object.

Other State-of-Art Methods

R-CNN

Selective search is used to extract just 2000 regions from the image, called region proposals, instead of huge number of regions.
Selective Search Algorithm:

1. Generate initial sub-segmentation, many candidate regions are generated
2. Use greedy algorithm to recursively combine similar regions into larger ones
3. Use the generated regions to produce the final candidate region proposals

These 2000 candidate region proposals are warped into a square and fed into a Convolutional Neural Network, which acts as a feature extractor that produces a 4096-dimensional feature vector as output. The extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four offset values, which help in adjusting the bounding box of the region proposal.

Problems with R-CNN

1. It takes a huge amount of time to train the network as 2000 region proposals per image are to be classified.
2. It cannot be implemented real time.
3. The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

Fast R-CNN

Instead of feeding the region proposals to the CNN, the input image is fed to the CNN to generate a convolutional feature map. From the convolutional feature map, the region of proposals are identified and warped into squares and by using a RoI pooling layer reshaped into a fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, a softmax layer is used to predict the class of the proposed region and also the offset values for the bounding box. The reason “Fast R-CNN” is faster than R-CNN is because 2000 region proposals don’t need to be fed to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it. Fast R-CNN is significantly faster in training and testing sessions over R-CNN. But region proposals become bottlenecks in Fast R-CNN algorithm affecting its performance.

Faster R-CNN

Faster R-CNN is an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals. Rest of the detection is done similar to Fast R-CNN. Faster R-CNN is much faster than it’s predecessors. Therefore, it can even be used for real-time object detection.

All these methods are region based, i.e., use regions to localize the object within the image. However, YOLO V3 processes the entire image in one go using a single CNN network.

SSD-Single Shot MultiBox Detector

SSD speeds up the process by eliminating the need of the region proposal network. The SSD object detection composes of 2 parts:

1. Extract feature maps. Multi-scale feature maps are used for object detection.
2. Apply convolution filters to detect objects.

For each cell (also called location), it makes 4 object predictions. Each prediction composes of a boundary box and 21 scores for each class (one extra class for no object), and the highest score is picked as the class for the bounded object. SSD reserves a class “0” to indicate it has no objects. Making multiple predictions containing boundary boxes and confidence scores is called multibox. It computes both the location and class scores using small 3×3 convolution filters. Separate filters are used for default boxes to handle the difference in aspect ratios. SSD can run at real-time speed and the accuracy is higher as well.

FPN-Feature Pyramid Network

Feature Pyramid Network (FPN) is a feature extractor. A pyramid of the same image at different scale is used to detect objects. Or a pyramid of feature can be created and use them for object detection. FPN composes of a bottom-up and a top-down pathway. The bottom-up pathway is the usual convolutional network for feature extraction. As we go up, the spatial resolution decreases. FPN provides a top-down pathway restores resolution with rich semantic information. As we go down the top-down path, the previous layer is upsampled by 2 using nearest neighbors upsampling. Lateral connections are added between reconstructed layers, which add more precise object spatial information back and also acts as skip connections. Top-down pathway plus lateral connections improve accuracy.

RetinaNet

RetinaNet is a single, unified network composed of a backbone network and two task-specific subnetworks. The backbone, generally FPN structure, is responsible for computing a conv feature map over an entire input image. The first subnet performs classification on the backbones output; the second subnet performs convolution bounding box regression. RetinaNet uses Focal loss to reduce loss for well-trained class. So whenever the model is good at detecting background, it will reduce its loss and reemphasize the training on the object class.

YOLO V3 is as accurate as SSD, RetinaNet or FPN; but three times faster than SSD. It also runs almost four times faster than RetinaNet or FPN.

References

[1] Dive Really Deep into YOLO v3: A Beginner's Guide - https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e

[2] YOLO v3 theory explained - https://medium.com/analytics-vidhya/yolo-v3-theory-explained-33100f6d193

[3] R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms - https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e