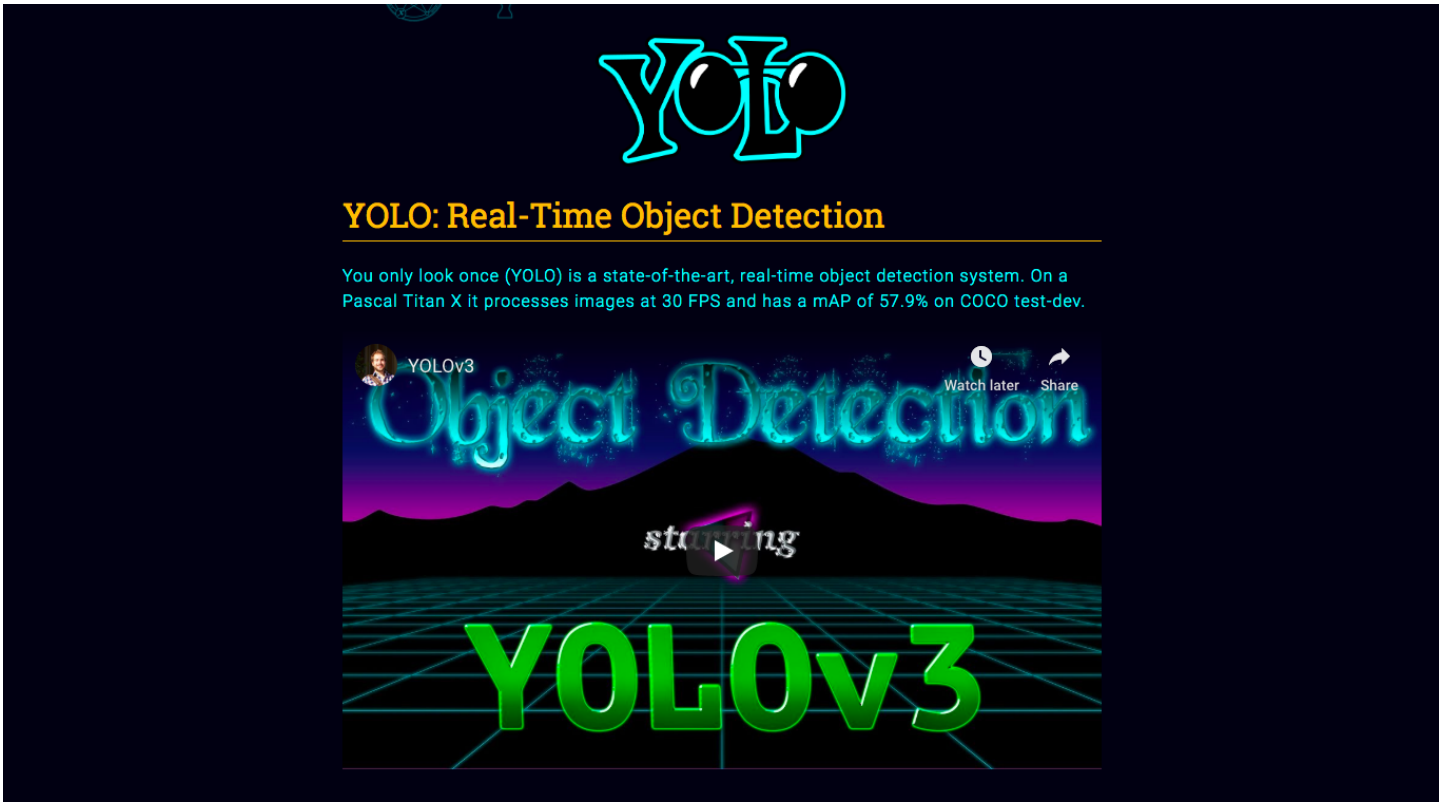
All About YOLO Object Detection and its 3 versions (Paper Summary and Codes!!)

Mehul Gupta
Sep 6, 2019 · 6 min read



OVERVIEW:

Object Detection has been amongst the hottest streams in Data Science. A lot of models have been explored and gained tremendous success. But the first & foremost that comes to our mind is YOLO i.e. You Look Only Once. Due to its tremendous speed, it has found its application in a number of live applications. The name itself explains a lot about itself, **that it just goes through the entire image just once**!! Hence we will be exploring how YOLO works and a comparative study of its different versions.

ABOUT YOLO:

YOLO has been amongst the most popular Object Detection Algorithm. Some important things to know are:

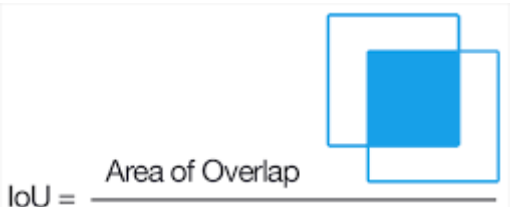
- It Intakes an image and divides it in a grid of S X S (where S is a natural number)
- Each pixel in the image can be responsible for a finite number of (5 in our case) bounding box predictions. A pixel is taken responsible for prediction when it is the center of the object detected. Out of all detected boxes, It is taken responsible for the detection of only one object and other detections are rejected.
- It predicts **C conditional class probabilities** (one per class for the likeliness of the object class).

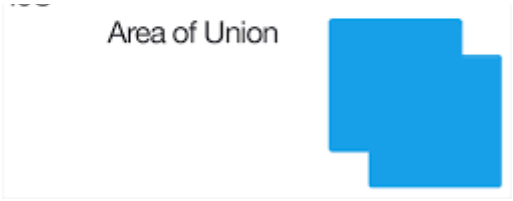
> *Total detections to be done per image=**S X S((B*5)+C)***
>
> *Where*
>
> ***S X S**= Total number of images yolo divides the Input*
>
> ***B*5**=*B is the Number of Bounding boxes Detected all over the image(Without any threshold consideration).For each bounding box, 5 elements are detected:*
>
> *Detected Objects Centre coordinates(x,y),*
>
> *Height and Width*
>
> *Confidence score.*
>
> ***C**=Conditional probability for Number of Classes.*

Hence if the image is divided in a 2 x 2 grid, and 10 boxes are predicted with 3 classes(Dog, Cat, Mouse), we will have 2*2(10*5+3) predictions=212 predictions

- Two thresholds are taken into consideration for selecting final detection

1. **IOU threshold**: The IOU threshold can be better understood using the below image:





Here the two boxes represent Predicted and True object

2. **Confidence threshold**: Threshold for minimum confidence the model has on a detected object(box confidence score)

- Below are certain scores calculated over the detected object.

$$\begin{aligned}\text{box confidence score} &\equiv P_r(object) \cdot IoU \\ \text{conditional class probability} &\equiv P_r(class_i|object) \\ \text{class confidence score} &\equiv P_r(class_i) \cdot IoU \\ &= \text{box confidence score} \times \text{conditional class probability}\end{aligned}$$

where

$P_r(object)$ is the probability the box contains an object.
IoU is the IoU (intersection over union) between the predicted box and the ground truth.
$P_r(class_i|object)$ is the probability the object belongs to $class_i$ given an object is presence.
$P_r(class_i)$ is the probability the object belongs to $class_i$

P(object) is 1 if a box is detected else 0.We must make this clear that IoU isn't the IOU threshold mentioned above but the predicted IoU.

Hence the entire process of detection can be summed up as:

- Dividing the image in a grid of S X S
- Detecting bounding box
- Calculating various scores(mentioned in the above image)
- Depending on the threshold for confidence and IoU, select some boxes
- **Use Non-Max suppression**: It helps us to avoid duplicate detections of the same object by rejecting multiple predictions for the same object. It takes a list of predicted boxes for the same image and accepts detection with maximum confidence.

- If the detection is over both IOU and Confidence threshold, It is taken as the final prediction

mAP:

Mean Average Precision is the metric used for YOLO's evaluation which can be explored here.

Loss function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

It has 4 major parts:

- Error in X, Y coordinates of the detected object
- Error in Height, width of the detected object(root is taken to give less weightage)
- Error in Classification. Lambda has been used to give more weightage to Confidence error.
- Error in confidence of object detected(similar to log loss)

Different Versions

YOLO requires a Neural Network framework for training and for this we have used DarkNet.The first version has 26 layers in total, with 24 Convolution Layers followed by 2 Fully Connected layers. The major problem with YOLOv1 is its inability to detect very small objects.

After the first version, 2 more versions for YOLO released that are:

YOLO9000 / YOLOv2:

- Inclusion of batch Normalization layers after each Conv Layer
- It has 30 layers in comparison to YOLO v1 26 layers.
- Anchor Boxes were introduced.

Anchor boxes are predefined boxes provided by the user to Darknet which gives the network an idea about the relative position and dimensions of the objects to be detected. It has to be calculated using the training set Objects.

- No fully connected layer present
- Random dimensions were taken for training images ranging from 320–608
- Multiple labels might be provided to the same objects, but still a multiclass problem(WordTree concept) i.e either the parent or child be the final label and not both.
- Still bad with small objects

YOLOv3:

- 106 layers neural network
- Detection on 3 scales for detecting objects of small to very large size
- 9 anchor boxes taken; 3 per scale. Hence more bounding boxes are predicted than YOLO9000 & YOLOv1
- MultiClass problem turned in MultiLabel problem
- Certain changes in the Error function.
- Quite good with small objects

Implementation:

You can find out Vehicle Number Plate Detection using YOLOv3 and Darknet at the below link:

Vehicle_Number_Plate_Detection for your reference

Some brief description of the major changes you need to do for custom training:

- Label your training dataset as given below:

> The training/validation set has the following components:
>
> - Images with objects(that has to be detected)
> - Text file corresponding each image which has the following composition:
>
> > *"Label_ID X_CENTER Y_CENTER WIDTH HEIGHT"*
>
> **Lets decode this first**
>
> - Label_ID is the numeric ID given to different classes that has to be determined starting from 0 i.e. if 3 classes has to be determined, cats,dogs and monkeys, IDs can be 0,1,2.
> - X_CENTER is the X coordinate of the centre of the object that has to be detected/Image_width
> - Y_CENTER is the Y coordinate of the centre of the object that has to be detected/Image_height
> - WIDTH is the width of the object that has to be detected/Image_width
> - HEIGHT is the height of the object that has to be detected/Image_height
>
> Hence all the values(except the ID) is in the range 0–1

- For multiple objects, take the same format for all objects in the same file, one below the other with correct Label_ID.Also, the text file has the same name as the

image name.

- For creating a labelled dataset, you might need to know YOLO_MARK.

Once done with the dataset,

- Git clone: https://github.com/AlexeyAB/darknet.git

Follow the below steps(all this within yolo-obj.cfg in cfg folder)

Training Yolo v3:

1. Create file `yolo-obj.cfg` with the same content as in `yolov3.cfg` (or copy `yolov3.cfg` to `yolo-obj.cfg)` and:

- change line batch to `batch=64`
- change line subdivisions to `subdivisions=8`
- change line max_batches to (`classes*2000`), f.e. `max_batches=6000` if you train for 3 classes
- change line steps to 80% and 90% of max_batches, f.e. `steps=4800,5400`
- change line `classes=80` to your number of objects in each of 3 `[yolo]` -layers:

- Change the 'filter' values as mentioned below not for all variables but the variable just above 'classes' as shown below.

So if `classes=1` then should be `filters=18` . If `classes=2` then write `filters=21` .

(Do not write in the cfg-file: filters=(classes + 5)x3)

(Generally `filters` depends on the `classes` , `coords` and number of `mask` s, i.e. filters= `(classes + coords + 1)*` `<number of mask>` , where `mask` is indices of anchors. If `mask` is absence, then filters= `(classes + coords + 1)*num`)

So for example, for 2 objects, your file `yolo-obj.cfg` should differ from `yolov3.cfg` in such lines in each of **3** [yolo]-layers:

```
[convolutional]
filters=21

[region]
classes=2
```

2. Create file `obj.names` in the directory `build\darknet\x64\data\` , with objects names - each in new line

3. Create file `obj.data` in the directory `build\darknet\x64\data\` , containing (where **classes = number of objects**):

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

4. Put image-files (.jpg) of your objects in the directory `build\darknet\x64\data\obj\`

- Create train.txt and test.txt having path_location for train/validation images.

- Once done with this, download initial weights for training from the below link:

darknet53.conv.74

Edit description

drive.google.com

- Now, we are ready with all our guns loaded. now we need to place these stuff at their right places:

Now do the following

1. Place yolo-obj.cfg & weights in /darknet
2. Place train.txt, test.txt, obj.data & obj.names in /darknet/data
3. Place the dataset in /darknet/data
4. Once done, run-make command in CLI in /darknet & chmod +x *.sh for making darknet.sh executable
5. Once done the cloned repository can be used for training,testing and calculating mAP for object detection.Confidence and threshold are set to 0.2 & 0.3 respectively for detection purpose

Codes to execute darknet for different purposes(for Linux):

- ./darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.74 -dont_show >> yolo_rotate_1.log

This trains your data using obj.data,yolo-obj.cfg and initial weights downloaded. -dont-show helps us logging.

- ./darknet detector test data/obj.data yolo-obj.cfg backup/yolo-obj_final.weights -ext_output -dont_show -out result.json < data/valid.txt

This helps us to test our model using final_weights stored in backup folder.-ext_output gives us the output coordinates of the detected objects and data/valid.txt is the validation data

- ./darknet detector calc_anchors data/obj.data -num_of_clusters 9 -width 416 -height 416

This helps us to calculated anchors using the training dataset

- Remember that the paths used in train.txt,test.txt should be relative to the darknet folder.
- Do get a GPU else your life would be quite hectic. If you get one, set GPU=1 in makeFile
- If GPU unavailable, set CUDNN and CUDA as 1 for faster training on CPU(not at all fast😅)

Conclusion:

YOLO has its ups and its downs. The trained model size goes up to 250MB which can be considerable. It has some exceptional performance over large objects but still requires some modification for small objects. Though I would suggest you use YOLOv3 and let the detection game begin!!!

Looking For more, explore below:

- **Reinforcement for beginners**
- **Tensorflow for Beginners**
- **Starting off with Time Series**
- **Preprocessing Time Series (With codes)**
- **Kaggle for beginners**
- **Data Analytics for beginners**
- **Statistics for beginners**
- **Data Science For Free!!!**

Machine Learning Object Detection Yolo Data Science Artificial Intelligence