Object Detection using YOLOv3

Team Members: Bhagyashri Shetti bss453@nyu.edu Harsha Pattapusetti vsp282@nyu.edu

1. Problem Statement:

Object detection is an important and challenging field in computer vision, one which has been the subject of extensive research. It has been widely used in autonomous driving, pedestrian detection, medical imaging, robot vision, intelligent video surveillance, etc. Even though there exist many detection methods, the accuracy, rapidity, and efficiency of detection are not good enough. $YOLOv3$ is an algorithm that uses deep convolutional neural networks to perform object detection. $YOLOv3$ is an improvement over previous $YOLO$ detection networks, it features multi-scale detection, a stronger feature extractor network, and some changes in the loss function. In this project, we will build a real-time object detector based on the $YOLOv3$ model and then load the model to make predictions for both images and videos.

2. Literature Survey:

- **You Only Look Once: Unified, Real-Time Object Detection.** [1]

  This paper presents $YOLO$, a unified model for object detection where we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. Unlike classifier-based approaches, $YOLO$ is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly. $YOLO$ pushes the state of the art in real-time object detection.

- ***YOLO9000*** **: Better, Faster and Stronger.** [2]

  This paper focuses on improving recall and localization while maintaining classification accuracy by adding batch normalization, high-resolution classifier, and convolution by anchor boxes. The authors propose a mechanism for jointly training on classification and detection data. The joint training allows $YOLO9000$ to predict detections for object classes that don't have labeled detection data.

- ***YOLOv3*** **: An Incremental Improvement.** [3]

  This paper presents updates to $YOLO$. The $YOLOv3$ method directly predicts class probabilities and bounding box offsets from full images with a single feed-forward convolution neural network. It completely eliminates region proposal generation and feature resampling and encapsulates all stages in a single network in order to form a true end-to-end detection system. $YOLOv3$ is fast and accurate in terms of mean average precision (mAP) and intersection over union (IOU) values.

- **Deep Residual Learning for Image Recognition.** [4]

  This paper presents a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

3. Description of Dataset:

A labeled dataset is needed to train the $YOLOv3$ algorithm. We will use the COCO dataset which consists of 80 labels. The COCO dataset is a large scale object detection, segmentation, and captioning dataset that has $330k$ images ($> 200k$ labeled), 1.5 million object instances, 80 object categories, 91 stuff categories, 5 captions per image and $250,000$ people with key points. The pre-trained weights of $YOLOv3$ with the COCO dataset are uploaded as initial weights for the model.

4. Description of Model:

"You Only Look Once" is an algorithm that uses convolutional neural networks for object detection. In comparison to recognition algorithms, a detection algorithm does not only predict class labels but detects locations of objects as well. So, it not only classifies the image into a category, but it can also detect multiple objects within an Image. This algorithm applies a single neural network to the full image. It means that this network divides the image into regions/grids and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

Architecture:

$YOLO$ makes use of only convolutional layers, making it a fully convolutional network. $YOLOv3$ presents a deeper architecture of feature extractor called $Darknet-53$. It contains 53 convolutional layers, each followed by batch normalization layer and Leaky ReLU activation. No form of pooling is used and a convolutional layer with stride 2 is used to downsample the feature maps. This helps in preventing low-level features often attributed to pooling.

This network architecture boasts of Residual blocks and Upsampling layers. The most significant feature of the residual block is the use of a short cut mechanism to alleviate the gradient disappearance problem caused by increasing the depth in the neural network, thereby making the neural network easier to optimize.

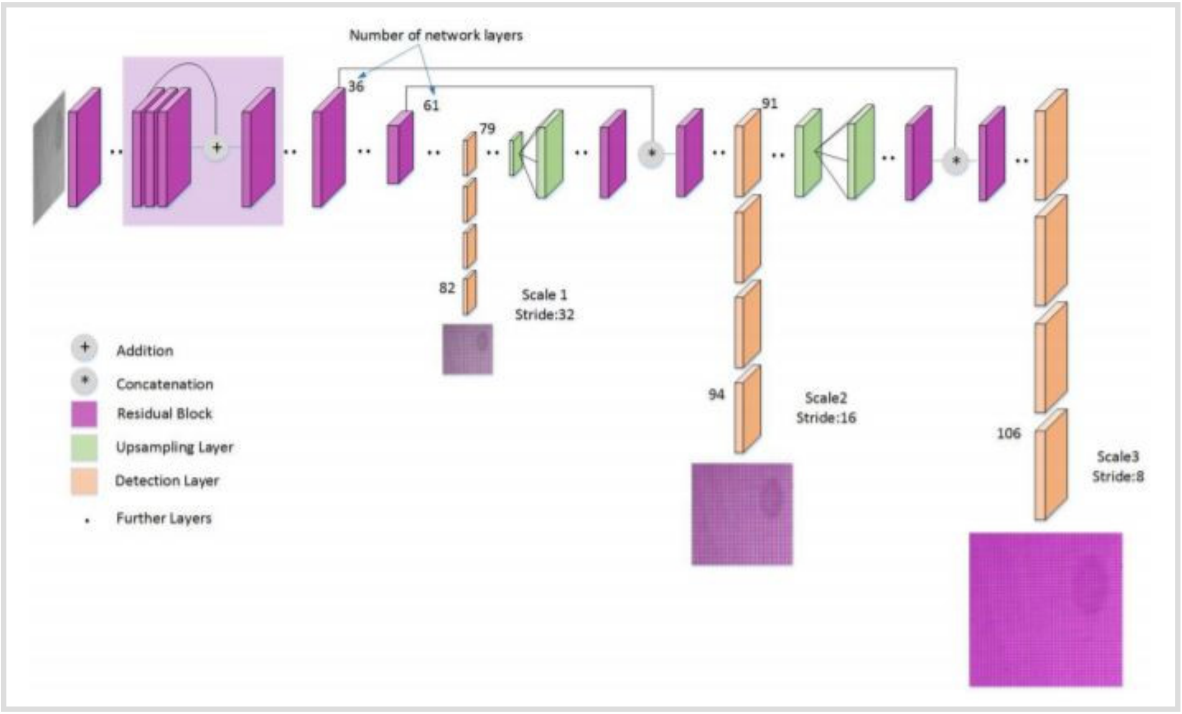


Fig: $YOLOv3$ architecture

As we can see in the figure above, $YOLOv3$ makes detection in 3 different stages. If we feed an input image of size 416×416, $YOLOv3$ uses 3 different prediction scales which split an image into (13×13), (26×26), and (52×52) grid of cells and with 3 anchor boxes for each scale which are chosen using k-means clustering.

For the first scale, $YOLOv3$ downsamples the input image into $13*13$ and makes a prediction at the 82nd layer. The 1st detection scale yields a 3-D tensor of size $13*13*255$. After that, $YOLOv3$ takes the feature map from layer 79 and applies one convolutional layer before upsampling it by a factor of 2 to have a size of $26*26$. This upsampled feature map is then concatenated with the feature map from layer 61. The concatenated feature map is then subjected to a few more convolutional layers until the 2nd detection scale is performed at layer 94. The second prediction scale produces a 3-D tensor of size $26*26*255$.

The same design is again performed one more time to predict the 3rd scale. The feature map from layer 91 is added one convolutional layer and is then concatenated with a feature map from layer 36. The final prediction layer is done at layer 106 yielding a 3-D tensor of size $52*52*255$. Detections at different layers help address the issue of detecting small objects found in YOLOv2.

Therefore, YOLO predicts $([(13*13) + (26*26) + (52*2)]*3) = 10,647$ bounding boxes. According to the research paper, we use the following formula to predict the bounding box predictions.

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w * e^{t_w}$$

$$b_h = p_h * e^{t_h}$$

where:

b_x, b_y, b_w, b_h :- Centre, width, height of predicted bounding box.

t_x, t_y, t_w, t_h :- Outputs of NN.

c_x, c_y :- Cell's top left corner of anchor box.

In order to reduce the detections from 10647 to 1, we first filter boxes based on their objectness score. The objectness score represents the probability that an object is contained inside a bounding box. Generally, boxes having scores below a threshold (for example below 0.5) are ignored.

We then make predictions using the output. $YOLOv3$ uses a multi-label approach which allows classes to be more specific and be multiple for individual bounding boxes. We concatenate the outputs at various scales and use Non-Max suppression (NMS) to get rid of multiple detections of the same object. NMS uses the very important function called "Intersection over Union"or (IOU).

$$IOU = \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|}.$$

IOU measures the overlap between two boundaries. We use that to measure how much our predicted output overlaps with the ground truth.

5. Description of Loss Function:

$YOLOv3$ has 3 detection layers which are also $YOLO$ layers. Each of the 3 $YOLO$ layers is responsible for calculating the loss at three different scales. The losses at 3 different scales are then summed up for back propagation. We only want one of the bounding boxes to be responsible for the object within the image since the $YOLO$ algorithm predicts multiple bounding boxes for each grid cell. To achieve this, we use the loss function to compute the loss for each true positive.

The loss functions looks as follows:

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

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

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

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

$1obj$ is defined as: 1, If an object is present in grid cell i and jth bounding box predictor is responsible for that prediction. 0, Otherwise. The loss function of YOLO v3 can be summarized as follows:

- Confidence loss, determine whether there are objects in the prediction frame;
- Box regression loss, calculated only when the prediction box contains objects;
- Classification loss, determine which category the objects in the prediction frame belong to.

$YOLOv3$ directly optimizes the confidence loss to let the model learn to distinguish the background and foreground areas of the picture. The rule of determination is: if the IoU of a prediction box and all real boxes is less than a certain threshold, then it is determined to be the background, otherwise, it is the foreground (including objects).

$YOLOv3$ predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. The last three terms in $YOLO$ are the squared errors, whereas in $YOLOv3$, they've been replaced by cross-entropy error terms, that is, the classification problem of all categories is reduced to whether it belongs to this category so that multi-classification is regarded as a two classification problem.

The advantage of this is to exclude the mutual exclusion of the categories, especially to solve the problem of missed detection due to the overlapping of multiple categories of objects.In other words, object confidence and class predictions in $YOLOv3$ are now predicted through logistic regression.

This looks as follows:

$$L = -\frac{1}{n}\sum^{x}(y*ln(a) + (1-y)*ln(1-a)$$

6. Description of Training Details:

$YOLO$ can only detect objects belonging to the classes present in the dataset used to train the network. For training the $YOLOv3$ model, we took the weights of a pre-trained model as our starting point. These weights have been obtained by training the network on COCO dataset, and therefore we can detect 80 object categories.

The original $YOLOv3$ weights file is a binary file and the weights are stored in the float data type, hence we first convert it to Tensorflow format. The COCO dataset class names file is placed at ./data/labels/coco.names.

$YOLOv3$ has 5 layers in general, they are: Convolutional layer, Upsample layer, Route layer, Shortcut layer and, $YOLO$ layer. There are two convolutional layer types i.e. with and without batch normalization. The convolutional layer followed by batch normalization uses leaky $ReLU$ activation layer ($alpha = 0.1$), otherwise it uses linear activation. Batch normalization helps regularize the model and thus reduces any kind of over-fitting.

Batch normalization is defined as follows:

Input: Values of x over a mini-batch: $\beta = x_{1....m}$

Parameters to be learned: $\beta\gamma$

Output: $$\hat{x}_i = \frac{x_i - u\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

$$y_i = \gamma\hat{x}_i + \beta \equiv BN_{\gamma\beta}(x_i)$$

The epsilon in the formula is set as 10^{-5} and the momentum which is used for computing moving average and variance is set as 0.9.

In order to concatenate with shortcut outputs from Darknet-53 before applying detection on a different scale, we upsample the feature map by a factor of stride and concatenate it with feature maps of previous layers having identical feature map sizes. To do this, $YOLOv3$ uses bilinear upsampling method. So, if we find upsample block, we retrieve the stride value and add a layer UpSampling2D by specifying the stride value $= 2$.

Since we have S*S grid of cells, after running a single forward pass pass convolutional neural network to the whole image, $YOLOv3$ produces a 3-D tensor with the shape of [S, S, B*(5+C)]. The B is the number of anchors and C is the number of classes.

In the final output layer, we perform our detection and do some refining to the bounding boxes. We use the sigmoid function to convert box_centers, confidence, and classes values into range of 0–1 .

We set the (IOU) and Confidence threshold as 0.5 to test the model and the maximum number boxes detected per image as 100. We also implemented the same for Tiny

$YOLOv3$ which is a lightweight detection algorithm applied to embedded platforms based on $YOLOv3$.

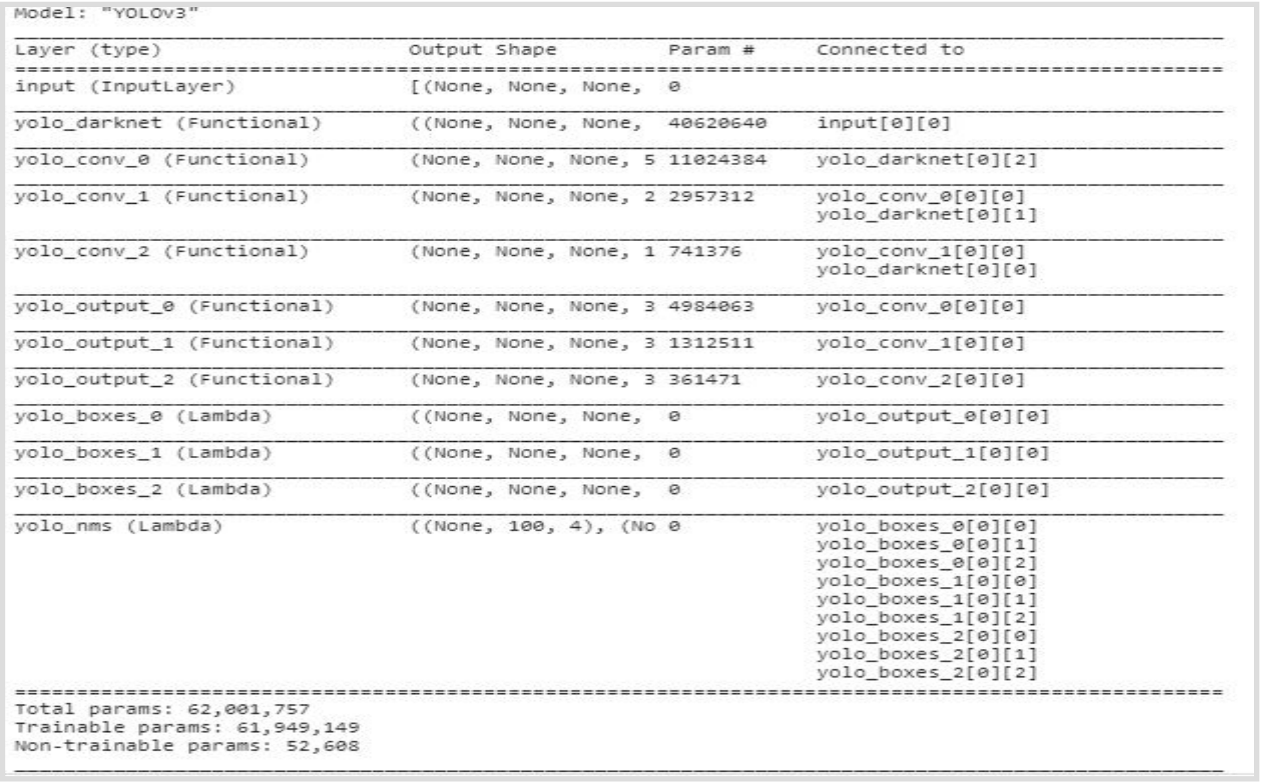
Pseudocode for testing the model on video using OpenCV:

- Capture the video through webcam or any saved video and pass each frame of video through the model.
- Define the threshold for IOU and scores. Get the boxes, scores and classes obtained as output and draw the boxes on the frame accordingly.
- After iterating through each frame, we are able to get the output video with the results we wanted which we can save directly in the detections folder by specifying the output path.

7. Results:

Our model is fast and is able to predict exact bounding boxes with classification and localization of objects in real-time. The upsampled layers concatenated with the previous layers help preserve the fine grained features which help in detecting small objects with good accuracy.

Our model summary looks like:



```
Model: "YOLOv3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input (InputLayer)              [(None, None, None,  0
__________________________________________________________________________________________________
yolo_darknet (Functional)       ((None, None, None,  40620640    input[0][0]
__________________________________________________________________________________________________
yolo_conv_0 (Functional)        (None, None, None, 5 11024384    yolo_darknet[0][2]
__________________________________________________________________________________________________
yolo_conv_1 (Functional)        (None, None, None, 2 2957312     yolo_conv_0[0][0]
                                                                 yolo_darknet[0][1]
__________________________________________________________________________________________________
yolo_conv_2 (Functional)        (None, None, None, 1 741376      yolo_conv_1[0][0]
                                                                 yolo_darknet[0][0]
__________________________________________________________________________________________________
yolo_output_0 (Functional)      (None, None, None, 3 4984063     yolo_conv_0[0][0]
__________________________________________________________________________________________________
yolo_output_1 (Functional)      (None, None, None, 3 1312511     yolo_conv_1[0][0]
__________________________________________________________________________________________________
yolo_output_2 (Functional)      (None, None, None, 3 361471      yolo_conv_2[0][0]
__________________________________________________________________________________________________
yolo_boxes_0 (Lambda)           ((None, None, None,  0           yolo_output_0[0][0]
__________________________________________________________________________________________________
yolo_boxes_1 (Lambda)           ((None, None, None,  0           yolo_output_1[0][0]
__________________________________________________________________________________________________
yolo_boxes_2 (Lambda)           ((None, None, None,  0           yolo_output_2[0][0]
__________________________________________________________________________________________________
yolo_nms (Lambda)               ((None, 100, 4), (No 0           yolo_boxes_0[0][0]
                                                                 yolo_boxes_0[0][1]
                                                                 yolo_boxes_0[0][2]
                                                                 yolo_boxes_1[0][0]
                                                                 yolo_boxes_1[0][1]
                                                                 yolo_boxes_1[0][2]
                                                                 yolo_boxes_2[0][0]
                                                                 yolo_boxes_2[0][1]
                                                                 yolo_boxes_2[0][2]
==================================================================================================
Total params: 62,001,757
Trainable params: 61,949,149
Non-trainable params: 52,608
__________________________________________________________________________________________________
```

Summary of how our model works:

✳ Input image $(416, 416, 3)$

✳ The input image goes through a CNN, resulting in a $(13, 13, 5, 85)$ dimensional output

✳ After flattening the last two dimensions, the output is a volume of shape $(13, 13, 425)$:

- Each cell in a 13*13 grid over the input image gives 225 numbers.

- 225 = 3*85 because each cell contains predictions corresponding to 3 anchor boxes.

- 85 = 5 + 80 where 5 is because bounding box (pc, bx, by, bh, bw) has 5 attributes, and 80 is the number of classes we would like to detect.

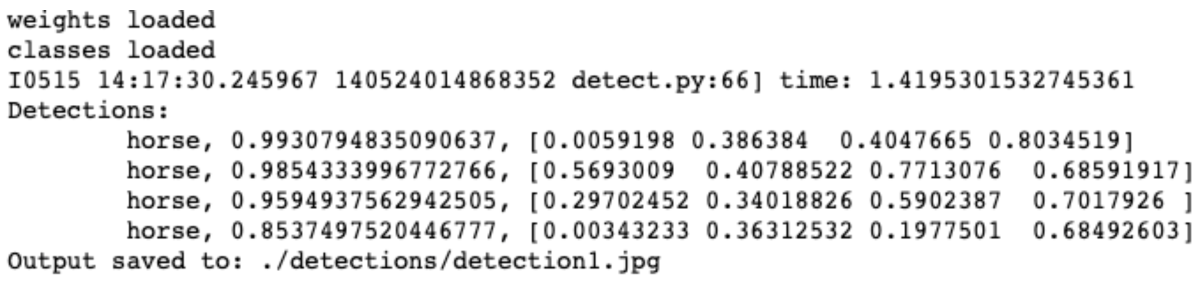
✳ We then select only few boxes based on:

- Score-thresholding: Throw away boxes that have detected a class with a score less than the threshold 0.5.

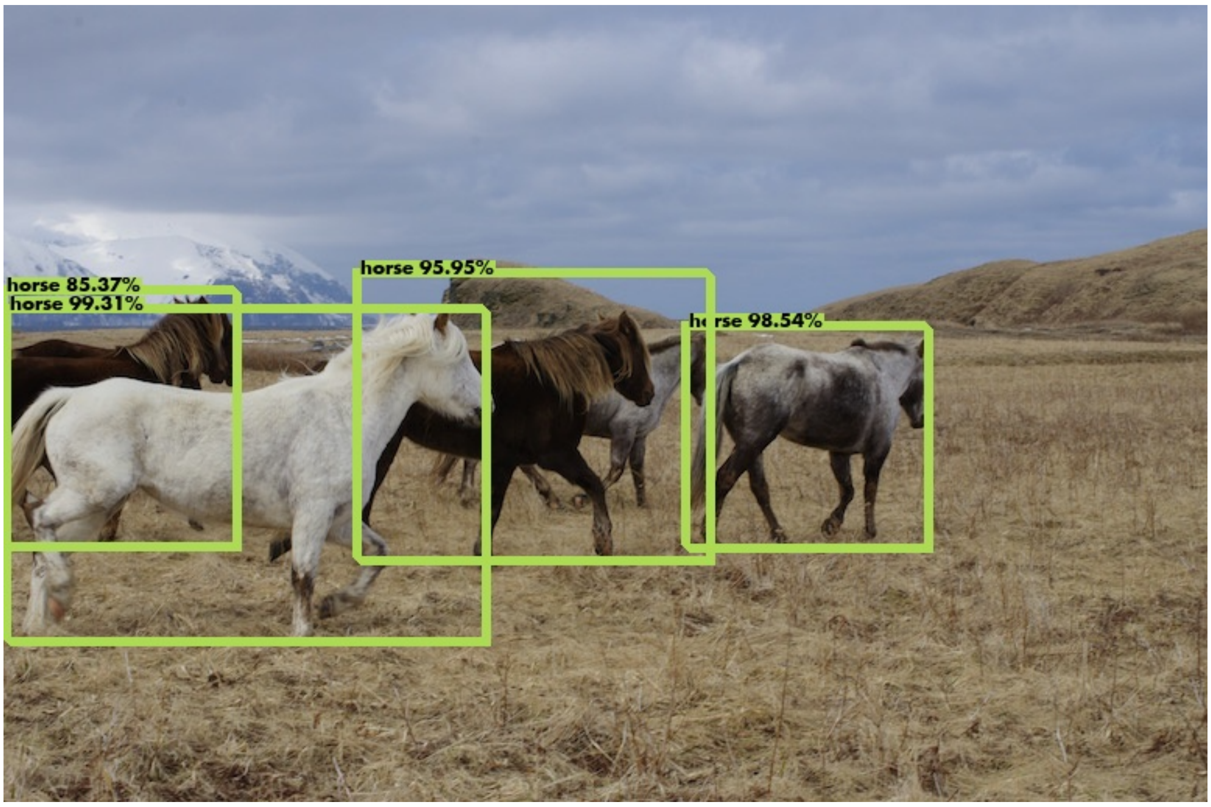
- Non-max suppression: Compute the Intersection over Union and avoid selecting overlapping boxes.

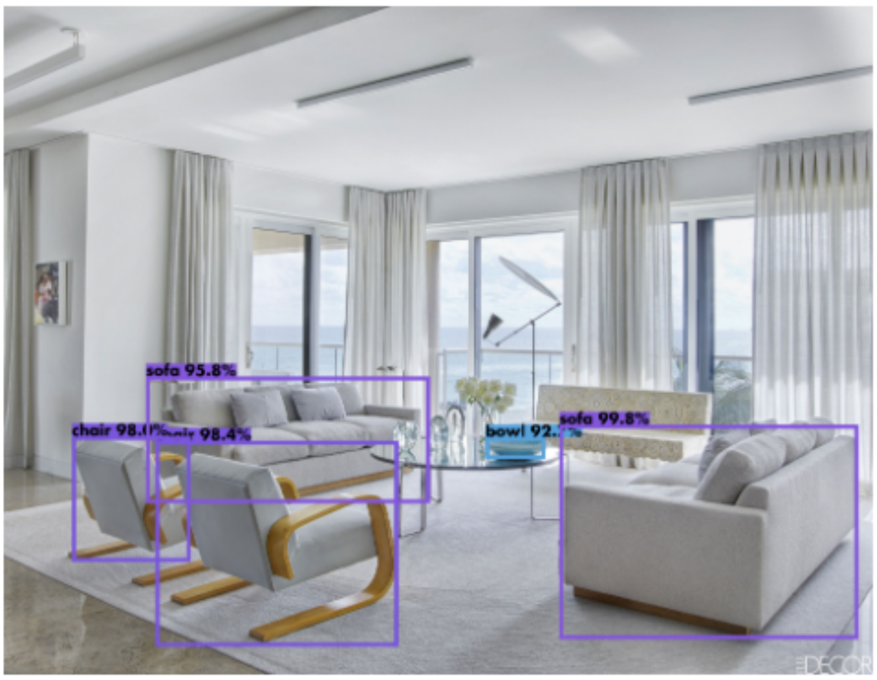
✳ This gives us $YOLO's$ final output which gets saved in the detections folder.

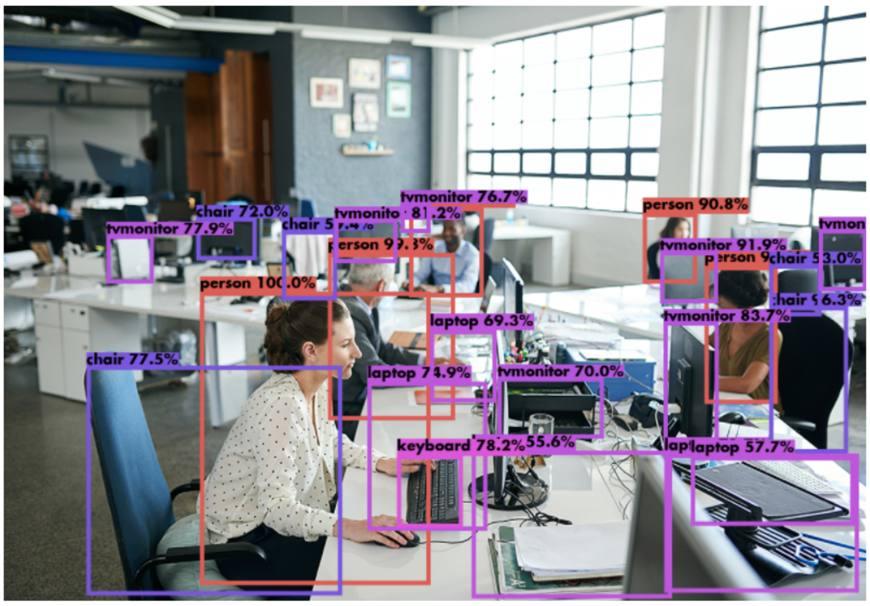
We print the detections i.e. the class names with their prediction values before displaying the image which also automatically gets saved in the detections folder created. This is how our result looks like:

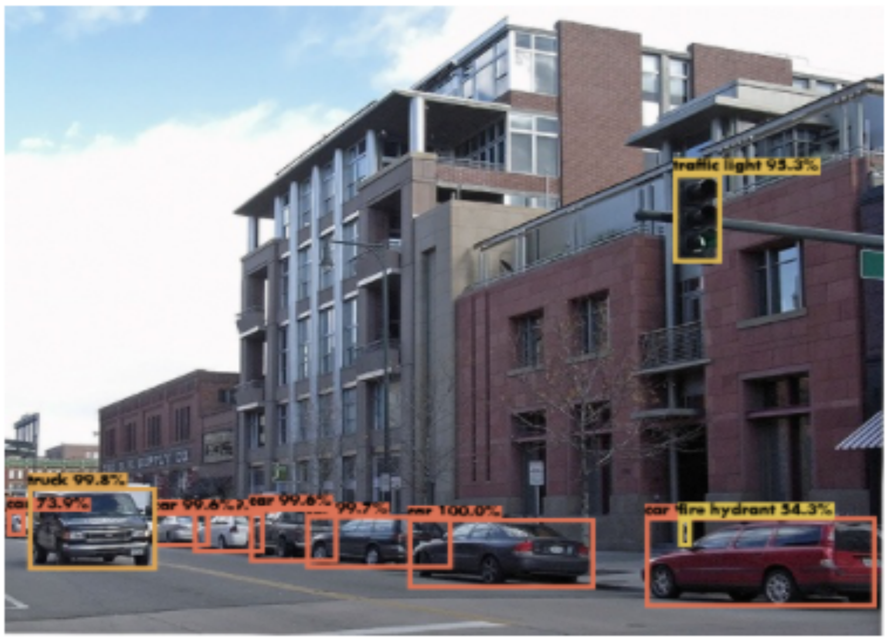


```
weights loaded
classes loaded
I0515 14:17:30.245967 140524014868352 detect.py:66] time: 1.4195301532745361
Detections:
        horse, 0.9930794835090637, [0.0059198 0.386384  0.4047665 0.8034519]
        horse, 0.9854333996772766, [0.5693009  0.40788522 0.7713076  0.68591917]
        horse, 0.9594937562942505, [0.29702452 0.34018826 0.5902387  0.7017926 ]
        horse, 0.8537497520446777, [0.00343233 0.36312532 0.1977501  0.68492603]
Output saved to: ./detections/detection1.jpg
```




sofa 95.8%
chair 98.0%
chair 98.4%
bowl 92.
sofa 99.8%


tvmonitor 76.7%
chair 72.0%
tvmonitor 81.2%
person 90.8%
tvmonitor 77.9%
person 99.3%
tvmonitor 91.9%
chair 53.0%
person 100.0%
laptop 69.3%
tvmonitor 83.7%
chair 77.5%
laptop 74.9%
tvmonitor 70.0%
keyboard 78.2%
55.6%
laptop 57.7%


traffic light 95.3%
truck 99.8%
73.9%
car 99.6%
car 99.6%
99.7%
car 100.0%
fire hydrant 54.3%

We apply the same algorithm to make predictions on videos that is available on the GitHub repository. We can capture the video through webcam or any saved video testing file.

8. Challenges:

- The performance drops as the IOU threshold is increased.
- Loading pre-trained Darknet weights was hard because the layer ordering is different in tf.keras and Darknet. Hence, we created sub-models in keras but since eras is not able to save nested model in h5 format properly we used TensorFlow checkpoint.
- $YOLOv3$ is a fast object detection algorithm, but unfortunately not as accurate as RetinaNet or Faster RCNN. We can use different training heuristics to improve $YOLOv3$. We use the approach of synchronized batch normalization which helps increase the batch size and makes the calculations less noisy.

9. Expected Results vs Actual Results, Conclusion and Future Scope

9.1 Expected Results vs Actual Results

We expected to implement $YOLOv3$ for object detection on images & videos and we implemented it successfully. We constructed and compiled $YOLOv3$ model in TensorFlow and Keras. We transferred weights from original Darknet weights to constructed model and then tested the model to make predictions on images and videos. All the image and video predictions are uploaded on the Github repository. It takes approximately 70ms for detection using $YOLOv3$ on Windows. Our model is fast and accurate and this makes it the best model to choose in applications where speed is important either because the objects need to be real-time or the data is just too big.

9.2 Conclusion

In this project, we used $YOLOv3$, which has good detection speed and good detection results. $YOLOv3$ performs at par with other state of art detectors like RetinaNet, while being considerably faster, at COCO (mAP) 50 benchmark. 50 here corresponds to 0.5 IoU. The average precision for small objects improved in $YOLOv3$, it is now better than Faster RCNN. The accuracy of detecting objects with $YOLOv3$ can be made equal to the accuracy when using RetinaNet by having a larger dataset, which makes it an ideal option for models that can be trained with large datasets. An example of this would be common detection models like traffic detection, where plenty of data can be used to train the model since the number of images of different vehicles is plentiful. It can detect finer details. As mAP increases, localization errors decrease. Predictions at different scales or aspect ratios for same objects improved because of the addition of feature pyramid like method. Thus, a very good accuracy with the best speed makes $YOLOv3$ a go to object detection model.

9.3 Future Scope

$YOLO$ algorithm completely provides an open structure for various target detection tasks. We can completely detect the target feature based on the original $YOLO$ network by cropping, deepening, lightening, combining and determine the number of output layers. This will provide a large application space for all kinds of real-time detection task in any working environment like any public place, station, corporate environment, streets, shopping malls, examination centers, etc., where accuracy and precision are highly desired to serve the purpose. This technique can be used in smart city innovation and it would boost up the development process in many developing countries.

10. Repository Links

- This is the link to our repository: `https://github.com/shettibhagya/Deep-Learning`
- Dataset Link: `https://cocodataset.org/#download`

References:

[1] You Only Look Once: Unified, Real-Time Object Detection. `https://arxiv.org/pdf/1506.02640.pdf`

[2] $YOLO9000$: Better, Faster, and Stronger. `https://arxiv.org/pdf/1612.08242.pdf`

[3] $YOLOv3$: An Incremental Improvement. `https://arxiv.org/pdf/1804.02767.pdf`

[4] Deep Residual Learning for Image Recognition. `https://arxiv.org/pdf/1512.03385.pd`