BMI / CS 771 Homework Assignment 3
FCOS: Fully Convolutional One-Stage Object Detector

Siddharth Baskar sbaskar2@wisc.edu
Karan Vikyath Veeranna Rupashree veerannarupa@wisc.edu
Anudeep Kumar kumar256@wisc.edu

1 Introduction

This assignment was focused on learning object detection by implementing FCOS - a scene recognition model. At first we went through the research paper on FCOS titled 'FCOS: Fully Convolutional One-Stage Object Detection' to have a complete understanding of the working and implementation of FCOS. This helped in answering the theoretical section of the assignment. After that, we started with model inference which is explained in detail in section 4. Following which we trained different models with having the bonus marks in consideration too in section 5. In the last section we have depicted the results in a table and have made conclusions regarding this assignment.

2 Dataset

The dataset used for this is the PASCAL VOC 2007 from University of Oxford. It is a dataset for image recognition. It is fundamentally a supervised learning learning problem in that a training set of labelled images is provided and it has a total of twenty classes. Some of the example images are shown in fig 1.

(a) Dirt Bike

(b) A child

(c) Living Room

(d) A cat

(e) Boats

(f) Airplane

Figure 1: Sample from Dataset

3 Understanding FCOS

3.1 Model Design

a. What is the output of the backbone?

The output of the backbone are feature maps that capture various information at different scales.These features are crucial for subsequent stages namely FPN and Bounding box detection stage. These feature maps capture hierarchical and spatial information from the image. The captured information include

- Low level features like edges, texture basic patterns.
- High Level features as we move deeper like semantic information like complex patterns, context of scene.[2]

In the journal [1] they have considered 4 feature maps $\{C_3, C_4, C_5\}$. With respect to an input of 800 $\times$ 1024 image, the size of the feature maps are calculated based on the down-sampling ratio. For an instance for C_3, $H = 800/8 = 100; W = 1024/8 = 128$ Similarly for C_4 ratio is 16 and for C_5 it is 32 which makes the size 50×64 and 25×32 respectively.

b. How many levels are considered in FPN? And how does the FPN generates its output feature maps?

In the journal [1] implementation they have considered five levels in the FPN. namely $\{P_3, P_4, P_5, P_6, P_7\}$.

FPN generates its output in a simple yet effective manner saving on number of parameters. To understand this lets delve in to how FPN has bottom up and top down pathways. Our backbone network already used bottom up pathway, with high resolution low semantic layer at the beginning and as we go up spatial resolution decreases and semantic value increases. Now using top down pathway FPN takes semantically rich layers and creates high resolution layers. It applies a 1 × 1 convolution filter to reduce C_5 channel depth to 256-d to create M5. This becomes the first feature map layer used for object prediction.

As it goes down the top-down path, it upsamples the previous layer by 2. It again apply a 1 × 1 convolution to the corresponding feature maps, for instance C_4 in the bottom-up pathway. Then we add them element-wise.[3] The attached figure provides an visual cue of it all.

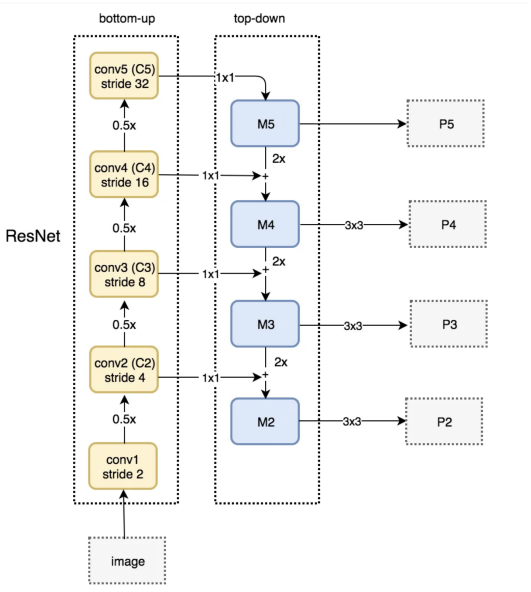


Figure 2: Top Down Pathway [3]

Now P_6 and P_7 are produced by applying 3 × 3 convolutional layer with a stride of 2 to P_5 and P_6 respectively. This procedure performs similar to if P_6 and P_7 were produced from backbone feature maps but it saves on number of parameters.

c. How does FCOS assign positive / negative samples during training? What are the loss functions used in the training?

FCOS computes regression targets l^*, t^*, r^*, b^* for each location on all feature levels. If a location at feature level $i \in [2, 7]$ satisfies the condition

$$m_{i-1} \leq max(l^*, t^*, r^*, b^*) \leq m_i \tag{1}$$

then it is assigned positive sample during training. If it is asssigned negative sample, then it is not required to regress a bounding box anymore. $m_i = [0, 64, 128, 256, 512, \infty]$

There are three types of loss used in training

- **Classification Loss** L_{cls}: For classification task FCOS uses Focal loss. It is designed to address the class imbalance problem that often occurs in object detection, where the majority of anchor points are negative. The focal loss assigns higher weights to hard-to-classify examples, focusing the training difficult samples. It is divided by N_{pos} the number of positive samples. It is calculated for every location and summed on the feature map.
- **Regression Loss** L_{reg} : GIoU Loss is implemented , which performs better than IoU. It is multiplied by λ/N_{pos} where $\lambda = 1$ .Let B_{gt} and B_{pred} represent the ground truth bounding box and the predicted bounding box, respectively.
  The GIoU is computed as follows[4]:

$$\text{IoU} = \frac{\text{Area}(B_{gt} \cap B_{pred})}{\text{Area}(B_{gt} \cup B_{pred})} \tag{2}$$

$$\text{C} = \text{Area}(\text{convex hull}(B_{gt}, B_{pred})) \tag{3}$$

$$\text{GIoU} = \text{IoU} - \frac{C - \text{Area}(B_{gt} \cup B_{pred})}{C} \tag{4}$$

- **Center-ness Loss** L_{center} : Uses Binary Cross Entropy loss.

Final equation :

$$L(p_{x,y}, t_{x,y}) = \frac{1}{N_{pos}} \sum_{x,y} L_{cls}(p_{x,y}, c^*_{x,y}) + \frac{\lambda}{N_{pos}} \sum_{x,y} \mathbb{1}_{\{c^*_{x,y}>0\}} L_{reg}(t_{x,y}, t^*_{x,y}) + \frac{1}{N_{pos}} \sum_{x,y} \mathbb{1}_{\{c^*_{x,y}>0\}} L_{center} \tag{5}$$

d. How does FCOS decode objects at inference? What are the necessary post-processing steps (e.g., non-maximum suppression)?

The inference of FCOS is straightforward. Given an input images, it is forwarded through the network and the classification scores $p_{x,y}$ and the regression prediction $t_{x,y} = (l, t, r, b)$ for each location on the feature maps Fi. After this it follows RetinaNet [5] It chooses the location with $p_{x,y}$ > 0.05 as positive samples and to obtain the predicted bounding boxes it uses formula :

$$x_0^{(i)} = x - ls \tag{6}$$

$$y_0^{(i)} = y - ts \tag{7}$$

$$x_1^{(i)} = rs + x \tag{8}$$

$$y_1^{(i)} = bs + y \tag{9}$$

$$\tag{10}$$

RetinaNet forms a single FCN comprised of a ResNet-FPN backbone,a classification sub-net,and a box regression subnet. In this case there will also be a centerness prediction. This centerness prediction $o_{x,y}$ and the classification score $p_{x,y}$ are multiplied and taken a square root to calcualte final score $s_{x,y}$

$$s_{x,y} = \sqrt{p_{x,y} \times o_{x,y}} \tag{11}$$

Consequently, center-ness can down-weight the scores of bounding boxes far from the center of an object. As a result, with high probability, these low-quality bounding boxes might be filtered out by the final non-maximum suppression (NMS) process, improving the detection performance remarkably[1]. Also it uses NMS threshold of 0.6 instead of 0.5

4 Model Inference

4.1 Classification and Regression Head

In this part er discuss the Forward pass for Classification and regression head. In the function we obtain 'x' which is a batch of images. For the 5 levels the H and W were $[[80, 80], [40, 40], [20, 20], [10, 10], [5, 5]]$.Batch size $N = 4$ and we pass it through a sequential network predefined in the code which consists of a 2D Convolutional layer , a Group Norm and a ReLU . Next we pass it through a Classification Head which is a 2D Conv layer then we initialize the weights of the convolutional layer using a normal distribution with a standard deviation of 0.01 and the bias of the convolutional layer using a constant value based on the focal loss formula, as described in the paper "Focal Loss for Dense Object Detection." This gives a tensor of shape $C \times H \times W$.

We can append this to a dictionary and make it a list of tensors of shape $N \times C \times H \times W$. But we decide to permutate this result for easier inference. So we convert the output to a list of tensor each of shape $HW \times C$ In classification case our output has list of shapes looking like : $[[6400, 20], [1600, 20], [400, 20], [100, 20], [25, 20]]$.

The process is same as above for Regression. The only change is we have C as 4 for box regression outputs and 1 for centerness output.

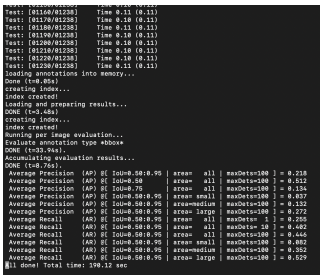
4.2 Decoding the Object

Drawing inspiration from the FCOS pytorch implementation [6] we started with declaring a list of dictionary. Now we obtain a argument *points* which is a list of length 5 containing tensors each corresponding to layers of shape =$[[80, 80, 2], [40, 40, 2], [20, 20, 2], [10, 10, 2], [5, 5, 2]]$ Now for our inference we had to modify it for our shapes and alos becuase it will be more efficient. So we reshape and append points to make them of the shape : $[[6400, 2], [1600, 2], [400, 2], [100, 2], [25, 2]]$. Now we start a for loop for the N images (4 in this case) . Loaded the 5 layer corresponding regression outputs for every image. Now we looped thru each layer. First we calculated score per level using the equation (11). Based on the given threshold we find the indexes to keep and this returns a list of True and False values in the corresponding index. We extract scores based on this. We also ensures that the k-value used in the top-k operation does not exceed the size of the input tensor. We now have top-k indexes. Since we have a flattened representation of class-specific predictions, and we want to convert those indices into indices corresponding to the boxes. So we divide each element of the top-k indexes tensor by number of classes, with the rounding mode set to "floor". We offset the label after calculating it by finding modulo of top-k values with number of classes.

The thing that was starkly different from [6] was they used anchorbox generation and some abstract functions to calculate and decode boxes. But we had to work out the math ourselve which helped us understand the implementation and develop a stronger grasp of the concept. We started with laoding the 4 points (l, t, r, b) from the argument reg_output. After loading using equation (6-8) We calculate the predicted bounding box coordinates. Initially we left it at this and were getting horrible results. Then we figured it out that we need to transform the points to real image scale. We used the formula $x_{image} = \lfloor s/2 \rfloor + x_{layer}s$. After implementing this we were able to get correct bounding boxes. We stacked the points obtained to recreate $[HW, 4]$ shaped tensor and then clipped it to image making sure it doesnt go out. Now we append this in a list for every layer. After doing this for all layer for an image we concatenate it along 0th dimension and sent it to $batched_nms()$ function provided by pytorch. We append the corresponding values to keep after NMS for each image and return the detections.

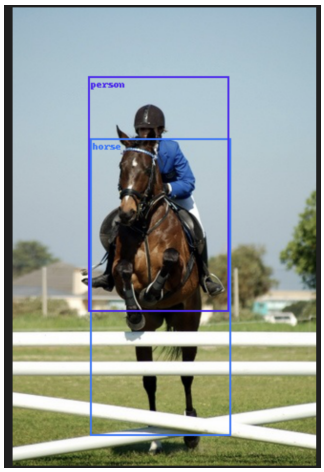
4.3 Testing your inference Code

Below are the results obtained after running *python ./eval.py ../pretrained/voc res18.yaml ../pretrained/voc res18.pth.tar*.We got an $mAP@50 = 51.2\%$ The results are a bit lower than expected. But it is important to note that the inference time is just **190.12 seconds** with an average time for *0.11 seconds* for every 10 image in Test. Before our final implementation we concatenated all layers together along 2nd dimension resulting in a tensor of $[8525, C]$ shape, where $8525 = 6400 + 1600 + 400 + 100 + 25$ and was iterating thru every point and it took *12 seconds* per image. I believe the accuracy could be increased with better permutation or NMS optimization technique.



(a) Inference Results

(b) Single Detection



(c) Multiple detection

5 Training the Model

5.1 Compute Loss

The compute loss function calculates three essential loss components for training the model: classification loss, regression loss, and centerness loss. These losses are later aggregated to form the final loss, which serves as the optimization objective during the training process. The function takes various inputs, including ground truth information (targets), feature map points (points), strides for different layers (strides), a range for regression, and the logits for classification, regression, and centerness. Initially a loop is ran which transposes the dimensions of the logits for classification, regression, and centerness. The logits are originally in the shape $N\_images \times 20 \times H \times W$, and after this operation, they become $N\_images \times H \times W \times 20$, $N\_images \times H \times W \times 4$, and $N\_images \times H \times W \times 1$, respectively.

Then anchor box is calculated based on the target centers and a sampling radius. These anchor boxes are used for positive/negative sample determination and after this the distances are calculated between each point on the feature map and the corresponding target bounding boxes. Now, a mask is created based on various conditions. It checks whether each point satisfies criteria related to being within target subboxes, target boxes, and having a max distance within a specified regression range and distinctions between the points lying inside and outside the bounding boxes are made. The points lying inside the bounding boxes are termed as foreground points and these points are counted towards the total count of positive samples.

Then the three losses are calculated starting with classification loss using sigmoid focal loss. This loss measures the difference between predicted and ground truth class probabilities. Followed by regression loss using generalized IoU (GIOU) loss. It measures the dissimilarity between predicted and ground truth bounding boxes. After that centerness loss is calculated which Centerness is a measure of how well a predicted bounding box's size matches the object it contains. The calculation involves extracting the distances for each foreground point, computing the centerness from these distances, and then calculating the loss using binary cross-entropy between predicted and target centerness. Finally the total loss is determined which is the sum of the three losses and then normalized by the number of positive samples to obtain the final loss.

5.2 Basic Model

In our initial assessment, the basic model (voc_fcos.yaml) was executed using default parameters to evaluate our compute loss function. Figure 4. illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 5 presents the evaluation results on the trained

model alongside examples of object detection in Figure 6. This baseline model achieved a mean Average Precision (mAP) at 50% threshold of **45.4%** , indicating a moderate level of accuracy in object detection tasks. The inference time, denoted as **142.81** seconds, reflects the model's response time during these tasks. Considering these outcomes as our base model benchmark, we proceeded with further optimizations, including hyperparameter tuning and strategic model design modifications, to enhance the model's performance and efficiency.

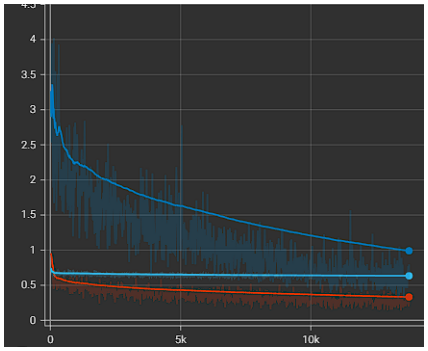


Figure 4: Training Curves

```
DONE (t=8.11s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.189
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.454
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.110
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.017
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.079
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.256
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.237
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.365
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.402
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.026
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.267
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.511
All done! Total time: 142.81 sec
```

Figure 5: Inference Results

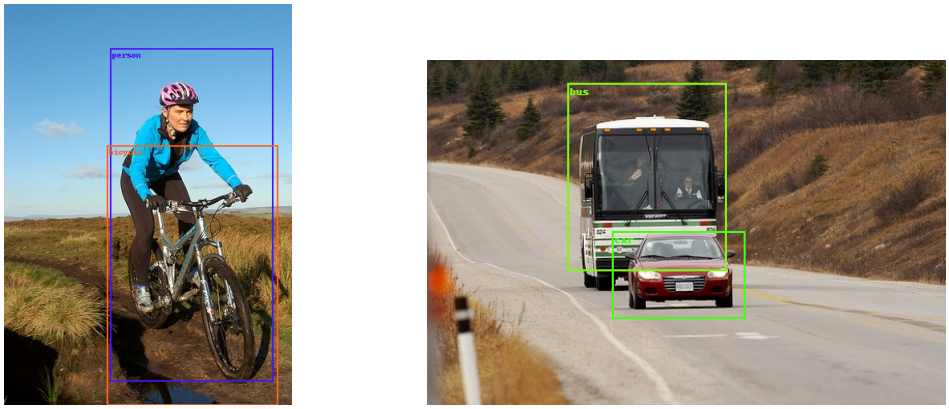


Figure 6: Object Detection Sample

5.3 ResNet-34

The ResNet-34 model with default parameters was run for 8 epochs. Figure 7 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 8 presents the evaluation results on the trained model alongside examples of object detection in Figure 9. This

model achieved a mean Average Precision (mAP) at 50% threshold of **48.9%** . This demonstrated an increase in mAP by 3.5% as compared to the base model.

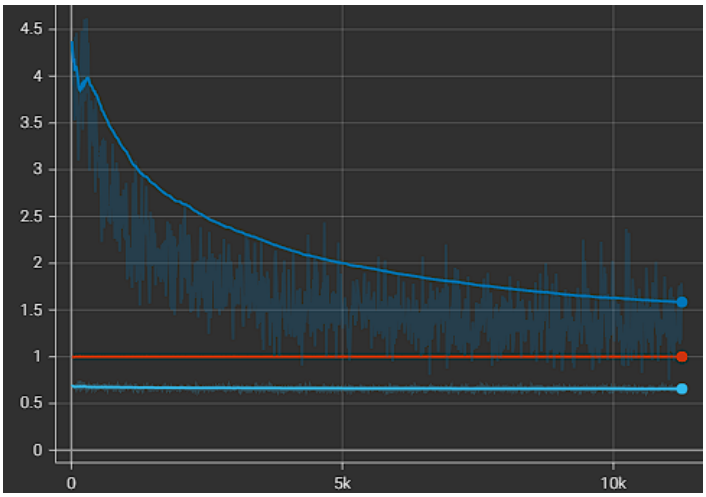


Figure 7: Training Curves

```
DONE (t=10.31s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.219
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.513
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.131
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.032
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.139
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.271
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.254
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.400
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.443
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.078
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.353
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.527
All done! Total time: 213.00 sec
```

Figure 8: Inference Results

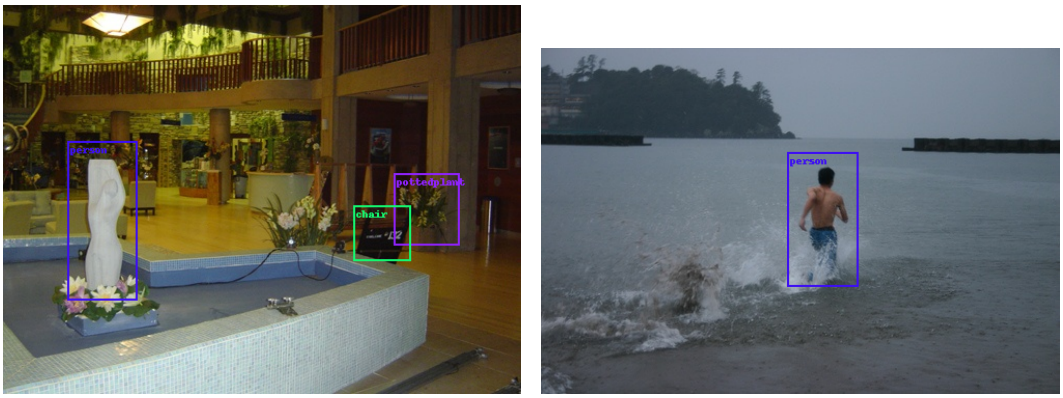


Figure 9: Object Detection Sample

5.4 ResNet-50

The ResNet-34 model with lr: 0.01 was run for 12 epochs. Figure 10 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 11 presents the

evaluation results on the trained model alongside examples of object detection in Figure 12. This model achieved a mean Average Precision (mAP) at 50% threshold of **56.2%** . This demonstrated an increase in mAP by 3.5% as compared to the base model.

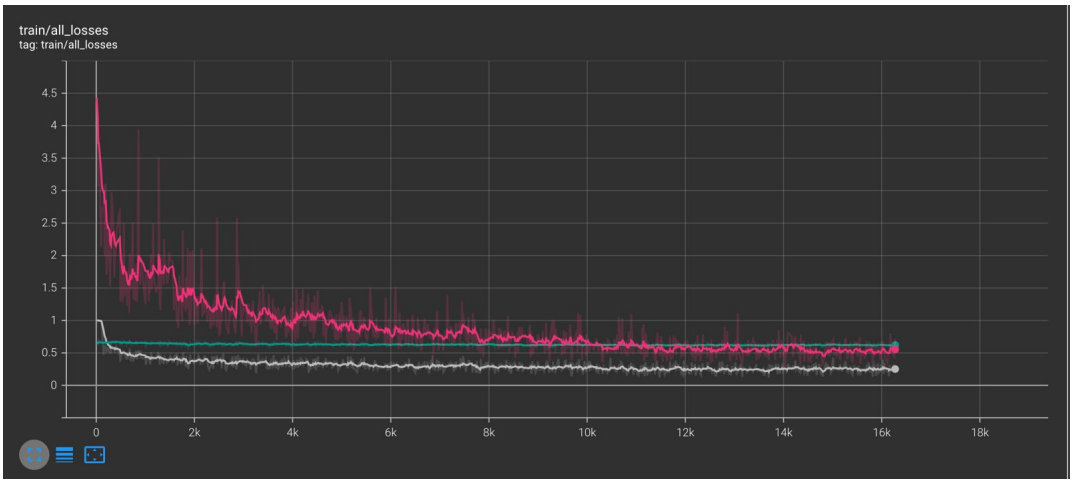
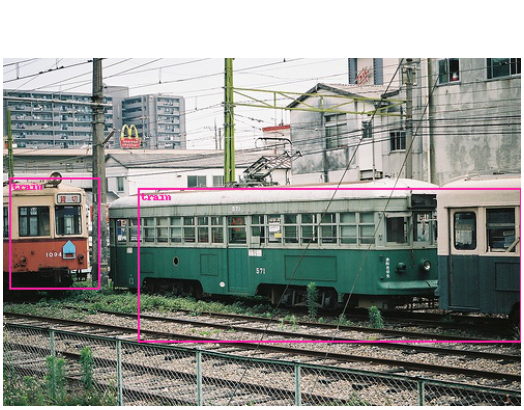


Figure 10: Training Curves

```
Accumulating evaluation results...
DONE (t=8.87s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.245
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.562
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.157
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.046
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.168
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.299
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.265
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.415
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.458
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.091
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.382
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.537
All done! Total time: 195.24 sec
```

Figure 11: Inference Results



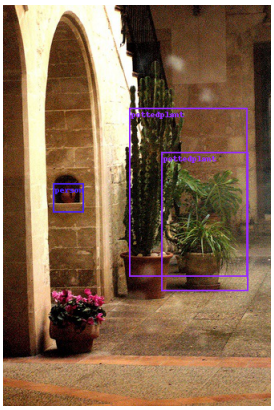


Figure 12: Object Detection Sample

5.5 ResNet-101

The ResNet-34 model with lr: 0.001 was run for 12 epochs. Figure 13 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 14 presents the evaluation results on the trained model alongside examples of object detection in Figure 15. This model achieved a mean Average Precision (mAP) at 50% threshold of **0.01%** . This was a failed experiment as observed from the results, including its increased training time.

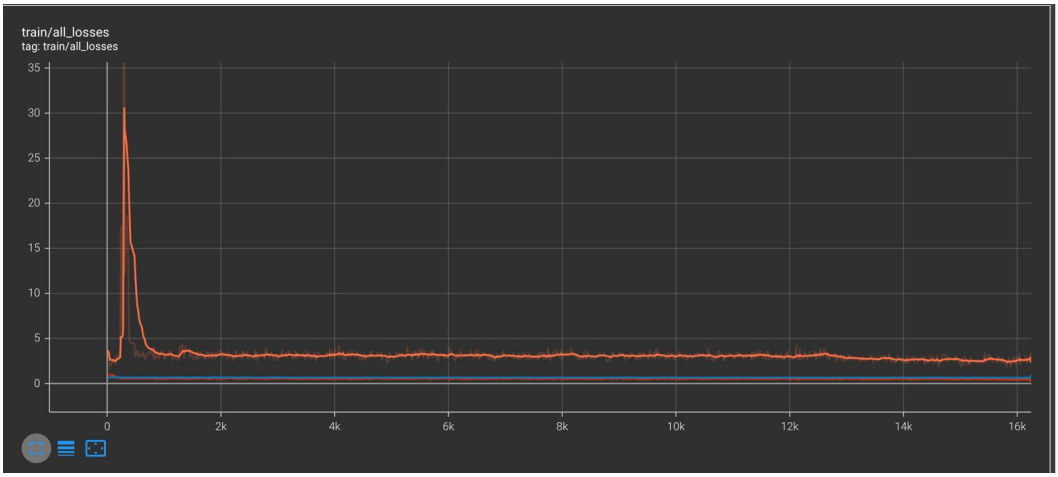


Figure 13: Training Curves

```
Accumulating evaluation results...
DONE (t=4.85s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.001
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.001
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.005
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.011
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.003
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.018
All done! Total time: 154.32 sec
```

Figure 14: Inference Results

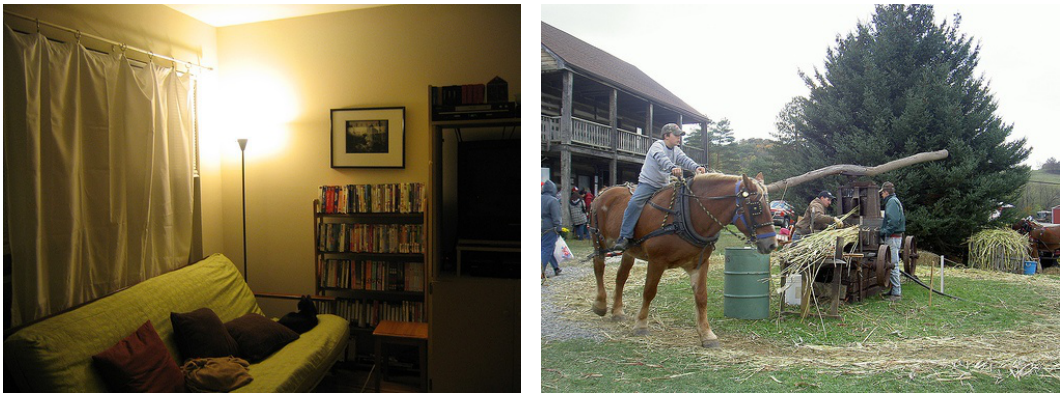


Figure 15: Object Detection Sample

5.6 ResNet-18 Hyperparameter Tuning

5.6.1 Hyperparameter Tuning 1

The resnet18 model with lr: 0.0025 was run for 15 epochs. Figure 16 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 17 presents the evaluation results on the trained model alongside examples of object detection in Figure 18. This model achieved a mean Average Precision (mAP) at 50% threshold of **48.9%** . This demonstrated an increase in mAP by 3.5% as compared to the base model.

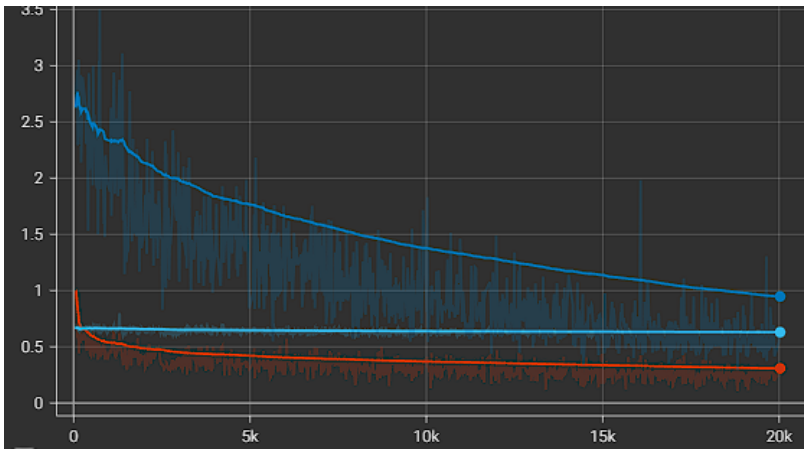


Figure 16: Training Curves

```
DONE (t=8.38s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.206
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.489
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.123
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.031
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.121
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.259
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.248
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.389
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.431
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.076
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.332
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.517
All done! Total time: 152.62 sec
```

Figure 17: Inference Results

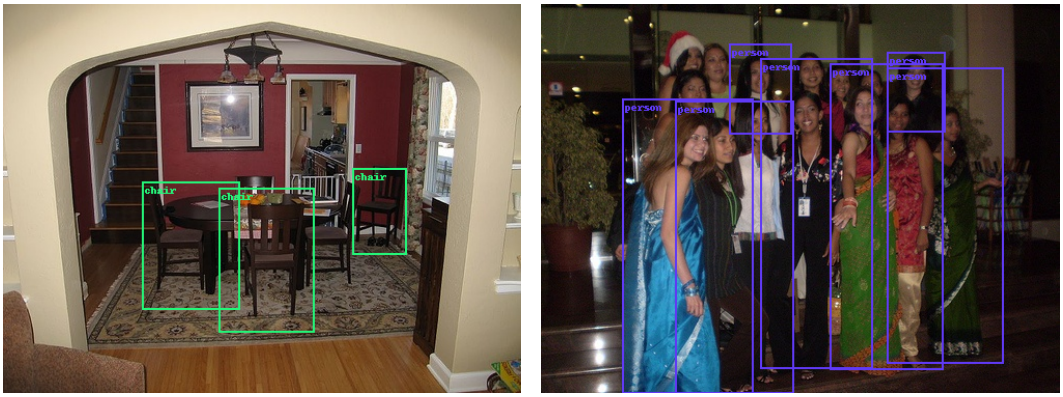


Figure 18: Object Detection Sample

5.6.2 Hyperparameter Tuning 2

The resnet18 model with lr: 0.001 was run for 20 epochs. Figure 19 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 20 presents the evaluation results on the trained model alongside examples of object detection in Figure 21. This model achieved a mean Average Precision (mAP) at 50% threshold of **49.1%** . This demonstrated an increase in mAP by 3.7% as compared to the base model and 0.2% to the previous model.

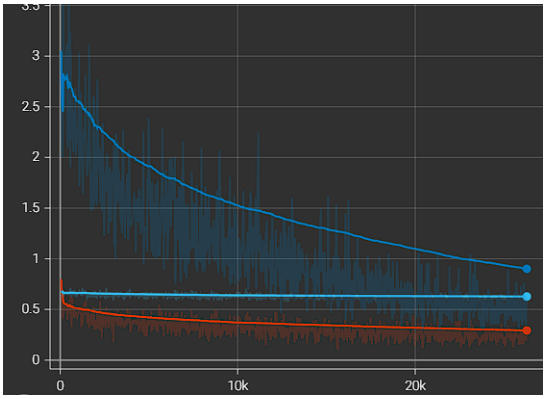


Figure 19: Training Curves

```
DONE (t=10.51s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.214
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.491
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.135
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.037
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.127
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.267
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.254
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.395
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.434
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.079
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.331
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.521
All done! Total time: 163.68 sec
```

Figure 20: Inference Results

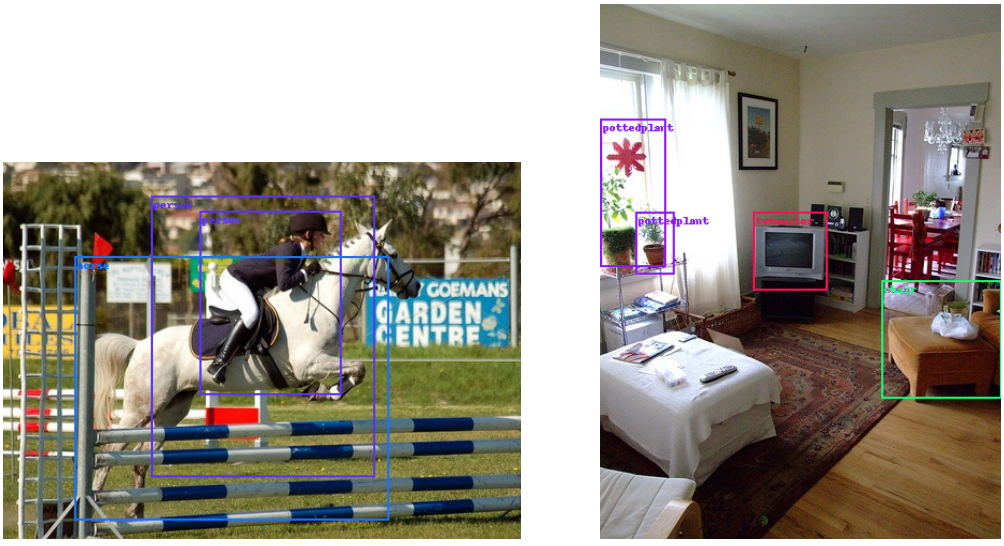


Figure 21: Object Detection Sample

5.6.3 Hyperparameter Tuning 3

The resnet18 model with lr: 0.003 was run for 20 epochs. Figure 22 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 23 presents the evaluation results on the trained model alongside examples of object detection in Figure 24. This model achieved a mean Average Precision (mAP) at 50% threshold of **49.9%** . This demonstrated an increase in mAP by 4.5% as compared to the base model and 0.8% to the previous model.

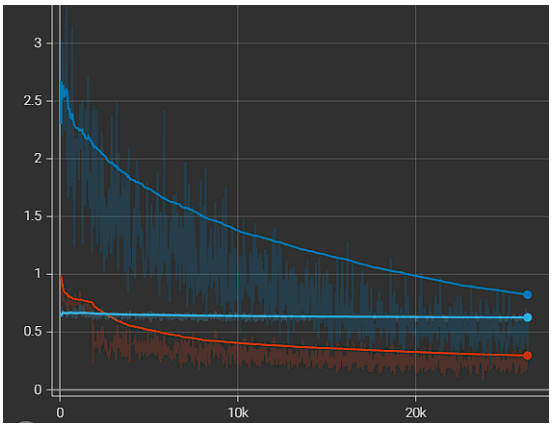


Figure 22: Training Curves

```
DONE (t=8.20s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.215
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.498
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.137
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.038
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.132
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.269
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.252
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.396
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.436
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.085
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.332
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.524
All done! Total time: 139.45 sec
```

Figure 23: Inference Results

Figure 24: Object Detection Sample

5.6.4 Hyperparameter Tuning 4

The resnet18 model with lr: 0.001, weight_decay: 0.0005 and batch_size: 32 was run for 50 epochs. Figure 25 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 26 presents the evaluation results on the trained model alongside examples of object detection in figure 27. This model achieved a mean Average Precision (mAP) at 50% threshold of **51.2%** . This demonstrated an increase in mAP by 5.8% as compared to the base model and 1.3% to the previous model.

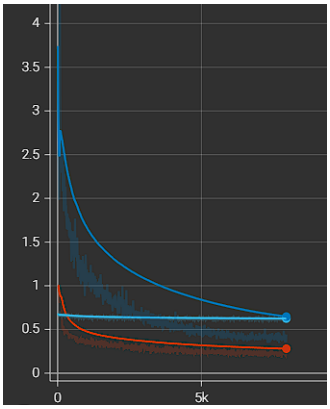


Figure 25: Training Curves

```
DONE (t=6.94s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.218
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.511
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.133
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.036
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.129
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.272
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.252
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.392
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.431
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.076
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.324
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.518
All done! Total time: 137.88 sec
```

Figure 26: Inference Results

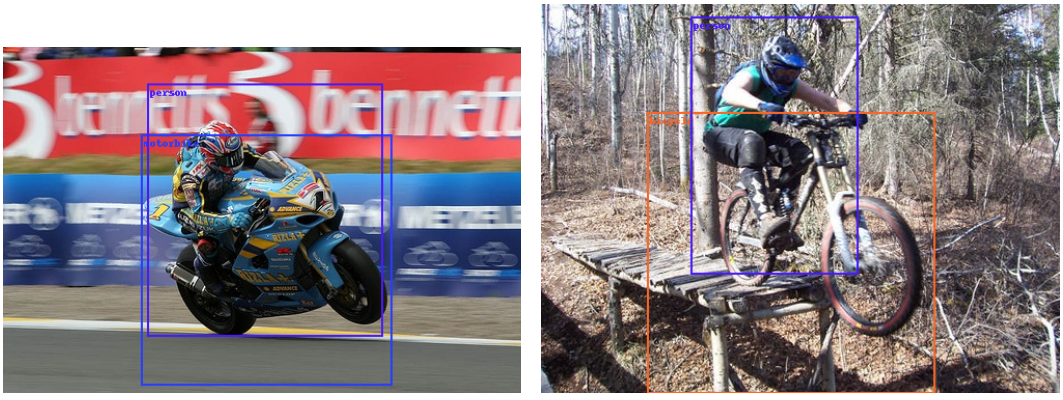


Figure 27: Object Detection Sample

5.6.5 Hyperparameter Tuning 5

The resnet18 model with lr: 0.001, weight_decay: 0.0005 and batch_size: 32 was run for 100 epochs. Figure 28 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 29 presents the evaluation results on the trained model alongside examples of object detection in Figure 30. This model achieved a mean Average Precision (mAP) at 50% threshold of **49.6%** and achieved a max mAP of **52.2%** when evaluated with 60th epoch weights This demonstrated an increase in mAP by 6.8% as compared to the base model and 1% to the previous model.

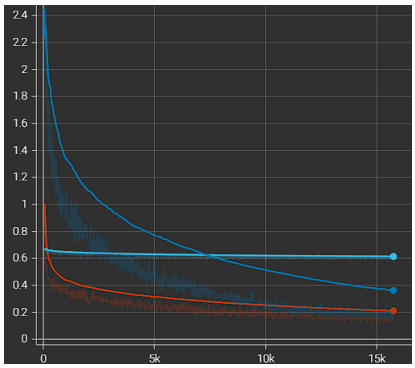


Figure 28: Training Curves

```
DONE (t=5.54s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.215
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.503
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.136
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.034
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.122
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.271
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.246
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.376
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.407
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.071
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.297
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.491
All done! Total time: 127.37 sec
```

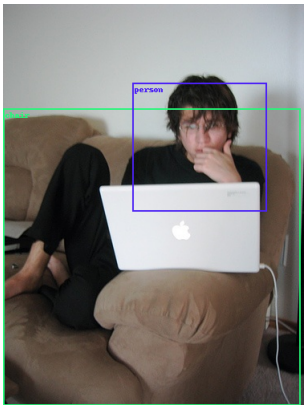
(a) Inference for 100th epoch

```
DONE (t=8.53s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.230
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.522
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.152
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.038
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.140
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.286
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.257
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.397
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.432
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.085
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.332
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.514
All done! Total time: 150.85 sec
```

(b) Inference for 60th epoch

Figure 29: Inference Results

Figure 30: Inference Results



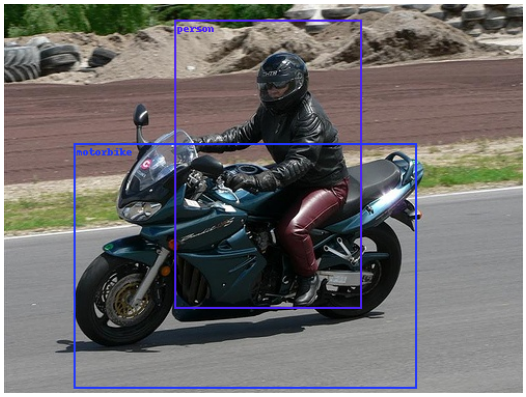


Figure 31: Object Detection Sample

5.7 Data Augmentation

Although we didn't find any success in our implementation of data augmentations, here are the different methods which were used.

5.7.1 Augmentation 1

The resnet18 model with default parameters was run for 8 epochs with the following augmentations commented in lines 97-103 in dataset.py. Figure 31 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 32 presents the evaluation results on the trained model alongside examples of object detection in Figure 33. This model achieved a mean Average Precision (mAP) at 50% threshold of **40.8%** . This demonstrated a decrease in mAP by 4.6% as compared to the base model.

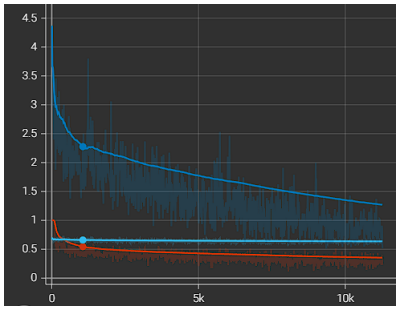


Figure 32: Training Curves

```
DONE (t=10.70s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.167
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.408
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.089
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.020
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.091
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.215
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.229
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.371
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.410
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.044
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.293
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.505
All done! Total time: 182.14 sec
```

Figure 33: Inference Results

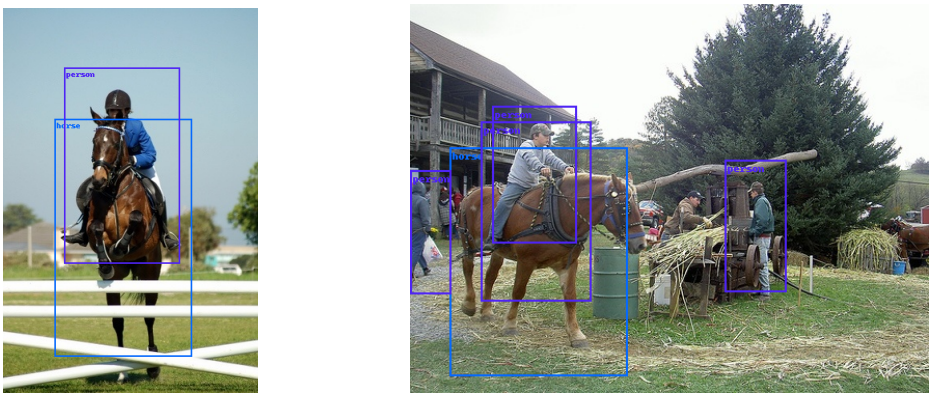


Figure 34: Object Detection Sample

5.7.2 Augmentation 2

The resnet18 model with default parameters was run for 8 epochs with the following augmentations commented in lines 108-114 in dataset.py. Figure 35 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 36 presents the evaluation results on the trained model alongside examples of object detection in Figure 37. This model achieved a mean Average Precision (mAP) at 50% threshold of **41.9%** . This demonstrated a decrease in mAP by 3.5% as compared to the base model and an increase of **1.1%** as compared to the previous model.

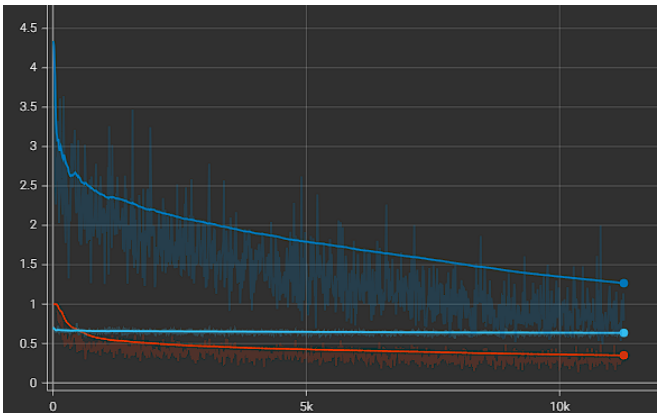


Figure 35: Training Curves

```
DONE (t=9.61s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.173
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.419
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.095
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.021
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.095
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.222
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.233
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.374
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.415
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.045
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.302
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.508
All done! Total time: 174.43 sec
```

Figure 36: Inference Results

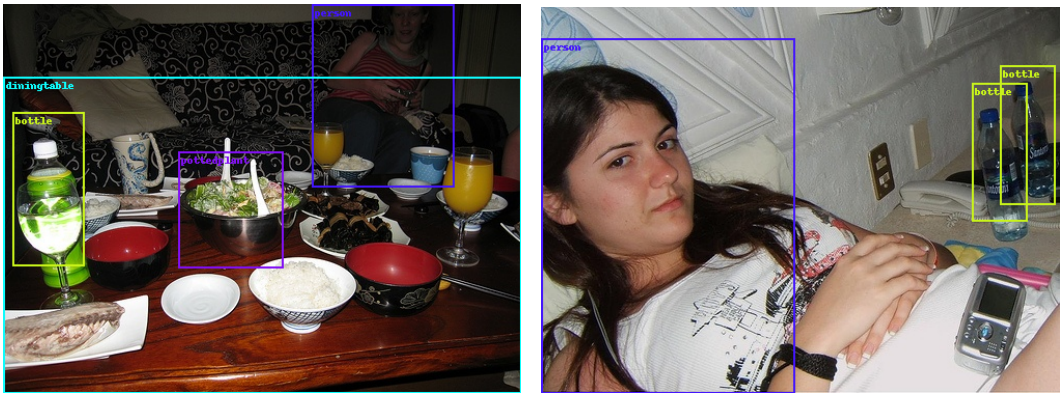


Figure 37: Object Detection Sample

5.7.3 Augmentation 3

The resnet18 model with default parameters was run for 8 epochs with the following augmentations commented in lines 119-129 in dataset.py. Figure 38 illustrates the training curve graphs, offering a detailed view of the model's learning progress over time. Figure 39 presents the evaluation results on the trained model alongside examples of object detection in Figure 40. This model achieved a mean Average Precision (mAP) at 50% threshold of **43.3%** . This demonstrated a decrease in mAP by 2.1% as compared to the base model and an increase of **1.4%** as compared to the previous model.

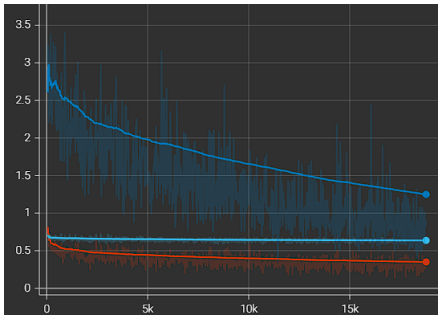


Figure 38: Training Curves

```
DONE (t=8.64s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.178
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.431
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.100
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.022
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.096
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.228
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.234
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.376
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.416
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.046
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.304
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.509
All done! Total time: 166.28 sec
```

Figure 39: Inference Results

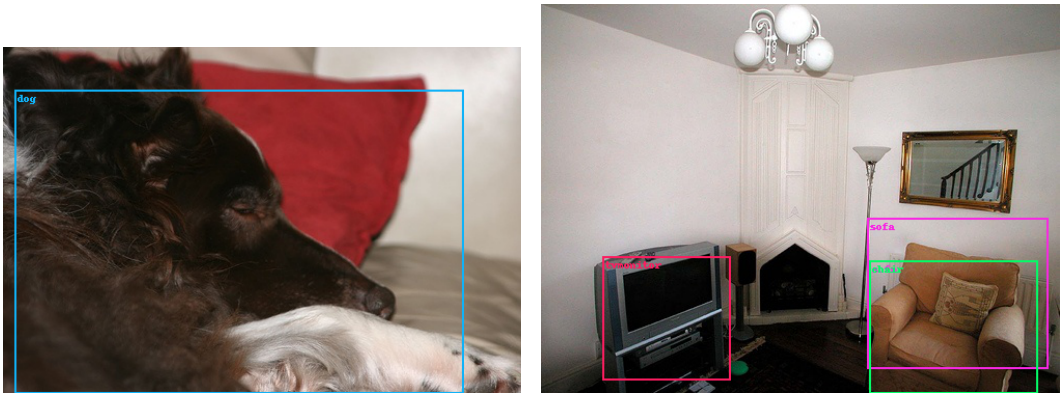


Figure 40: Object Detection Sample

6 Results and Conclusion

Table 1 depicts the results obtained for this assignment which is explained in detail in the following subsections

Table 1: Comparison of Object Detection Models

Method	AP	AP_{50}	AP_{75}	Inference Time(s)	Training Time(mins)
Voc Base Model	18.9	45.4	11.0	142.81	51.21
ResNet-34	21.9	51.3	13.1	213.0	52
ResNet-100	0	0.1	0	154.32	13hr43
ResNet-18 w/ Aug 1	16.7	40.8	8.9	182.14	44
ResNet-18 w/ Aug 2	17.3	41.9	9.5	174.43	1hr5
ResNet-18 w/ Aug 3	17.8	43.1	10.0	166.28	1hr12
ResNet-18 HPT 1	20.6	48.9	12.3	152.62	1hr15
ResNet-18 HPT 2	29.4	49.1	13.5	163.68	1hr38
ResNet-18 HPT 3	21.5	49.8	18.7	139.45	1hr37
ResNet-18 HPT 4	21.8	51.2	13.3	137.88	3hr33
ResNet-18 HPT 5	21.5	50.3	13.6	127.37	6hr52
ResNet-18 HPT 5 @60epochs	23.0	52.2	15.2	150.85	
ResNet-50	24.5	56.2	15.7	195.24	1hr9

6.1 Aggressive data augmentations

Based on the comprehensive analysis and outcomes derived from our experimental procedures, it has become evident that the data augmentation techniques implemented did not yield the anticipated enhancement in the model's performance. Despite the application of various augmentation strategies intended to introduce robustness and diversity in the training dataset, there was no measurable improvement in the model's accuracy or efficiency. This conclusion is substantiated by a careful comparison of the performance metrics before and after the application of these techniques. Consequently, this finding necessitates a reevaluation of our approach to data augmentation, and it prompts further investigation into alternative methods that could potentially lead to more favorable results in the model's ability to learn and generalize from the data provided.

6.2 Better model design for improved accuracy

The data presented in the table clearly indicates an upward trend in accuracy among the evaluated models, especially when compared to the base model. This improvement is most notably exemplified by the ResNet50 model, which achieved the highest accuracy with a mean Average Precision (mAP)@50 of 56.2%. This represents a significant advancement in model performance. What makes the achievement of the ResNet50 model particularly noteworthy is its efficiency. The model not only attains superior accuracy but does so within a training time frame comparable to that of the base model. This efficiency suggests that the architectural features of ResNet50 are highly effective for this specific task, enhancing accuracy without necessitating additional training time.

6.3 Better model design for improved efficiency

The analysis reveals an interesting aspect of the models' performance: despite having similar training times, there's a notable improvement in their inference times. This improvement is a critical factor in evaluating the efficiency of these models. Remarkably, the model with Hyperparameter Tuning 5 (HPT 5) stands out by demonstrating the most significant reduction in inference time.

6.4 Hyperparameter Tuning for improved performance

In the comparative analysis of the models, a notable trend emerges: each model exhibits an increase in accuracy relative to the base model. This is evident from the data, where the model trained

with hyperparameter tuning (HPT) demonstrates a leap in performance, achieving a mean Average Precision (mAP)@50 of 52.2%. This is a substantial improvement, underscoring the effectiveness of hyperparameter optimization.

7 Teammate Contributions

Anudeep Kumar	**Karan Vikyath Veeranna Rupashree**	**Siddharth Baskar**
Understanding FCOS	Understanding FCOS	Understanding FCOS
Model Inference	Model Training	Model Training
Bonus	Bonus	Bonus

References

[1] Tian, Zhi, Chunhua Shen, Hao Chen, and Tong He. "FCOS: A simple and strong anchor-free object detector." IEEE Transactions on Pattern Analysis and Machine Intelligence 44, no. 4 (2020): 1922-1933.

[2] FCOS Article LearnOpenCV

[3] FPN explained Medium

[4] Review-GIoU Medium

[5] Lin, Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection." In Proceedings of the IEEE international conference on computer vision, pp. 2980-2988. 2017.

[6] PyTorch FCOS