CS771 Homework Assignment 3 Write-up

Moniek Smink
smink2@wisc.edu

Venkata Saikiranpatnaik
vbalivada@wisc.edu

Sourav Suresh
sourav.suresh@wisc.edu

1. Team Member Contributions

Name	Contribution
Moniek Smink *smink2@wisc.edu*	4: Model Inference
Venkata Saikiranpatnaik *vbalivada@wisc.edu*	3: Understanding FCOS 6: Bonus
Sourav Suresh *sourav.suresh@wisc.edu*	5: Model Training 6. Bonus

2. Overview

The main objectives of the assignment are three-fold. First, we understand FCOS, see Section 3. Next, we implement the regression and classification heads as well as the inference prediction decoding in Section 4. Finally, we implement the multi-task loss and train the model in Section 5. As a bonus, we explore possible extensions to FCOS that can significantly increase the performance in Section 6.

3. Understanding FCOS

In this section, we discuss some major details of the FCOS design and implementation.

The high-level overview of the FCOS [4] architecture is shown in Figure 1. As can be seen from the figure, there are three major components that make up the architecture. The backbone, the feature pyramid network, and, finally, the classification/regression heads. We try to understand these components in the following subsections.

3.1. Backbone

In this subsection, we answer the question:

What is the output of the backbone?

First, we decide which backbone to use. The journal paper [4] uses Resnet-50 in the Experiments section. However, we assume Resnet-18 since that is the default configuration in the provided skeleton code. The C3, C4, and C5 in figure 1 correspond to the named layers, 'layer2', 'layer3', and 'layer4' of Pytorch's built-in Resnet-18 pretrained model. What is the shape of each of these outputs? The height and width of the feature maps are variable and dependent on the input image size. This is because we do not scale images to a fixed input size in a fully convolutional network (because we do not need to). Now, say that the original image has height H and weight W. The layers $C3$, $C4$, and $C5$ have shapes $(128, \lfloor H/8 \rfloor, \lfloor W/8 \rfloor)$, $(256, \lfloor H/16 \rfloor, \lfloor W/16 \rfloor)$, and $(512, \lfloor H/32 \rfloor, \lfloor W/32 \rfloor)$ respectively. We call the down sampling ratio between the input shape and the layer shape, the stride.

3.2. Feature Pyramid Network (FPN)

The outputs of the backbone are input to the FPN. The FPN employs convolutions, upsampling, and merging the output of the upper layers. In this subsection, we answer the questions:

How many levels are considered in FPN?

There are 5 levels in total as can be seen in Figure 1. Three of these layers correspond to the output of the backbone. Moreover, we have two additional convolutional layers on top of the last layer in FPN to make up a total of 5 levels.

How does FPN generate its output feature maps?

The main operation of FPN is shown in Figure 2. The paper [1] describes the operation in more detail. We upsample the output from the upper layer using nearest interpolation. We also apply 1x1 convolution on the output of the backbone network to make the number of output channels uniform across the FPN layers. After adding these two tensors, finally, we apply a 3x3 convolution to produce the output for the classification/regression heads. The number of channels here is a hyperparameter controlled by the `model.fpn_feats_dim` config. The height and width are determined similarly to the backbone. Hence, the output shape is $(num\_channels, \lfloor \frac{H}{2^{level}} \rfloor, \lfloor \frac{W}{2^{level}} \rfloor)$.

Levels $P6$, and $P7$ behave differently from the rest of the FPN network in how they compute their outputs. They are convolutional layers with kernel size 3, stride 2, and padding 1. There is a ReLU non-linearity between $P6$ and

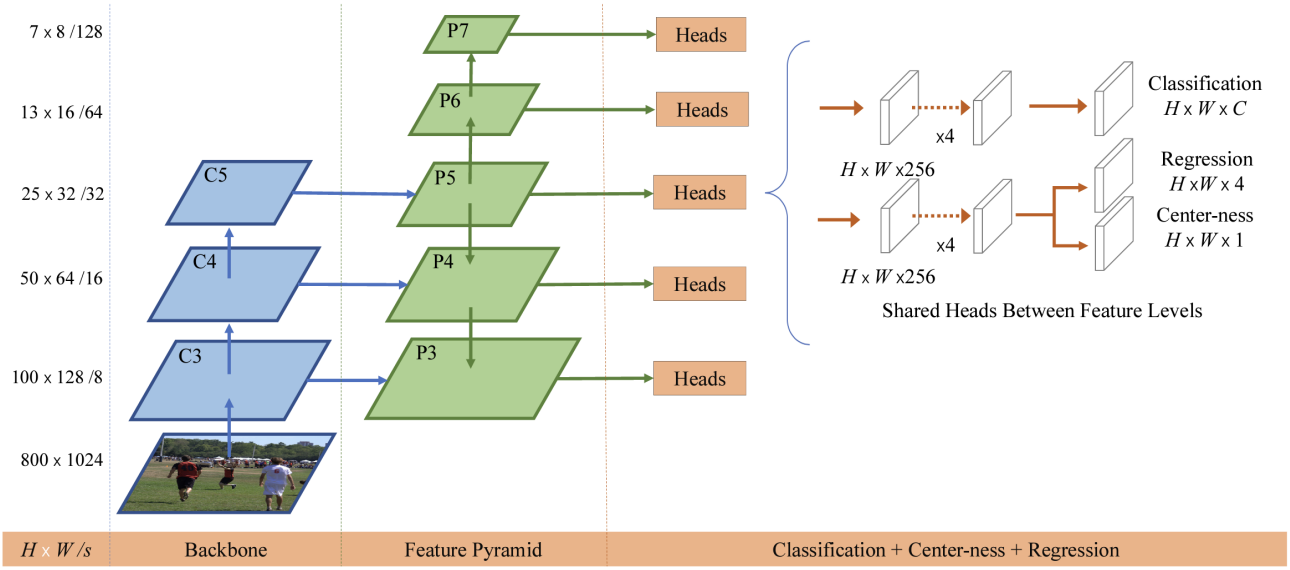

Figure 1. FCOS Architecture

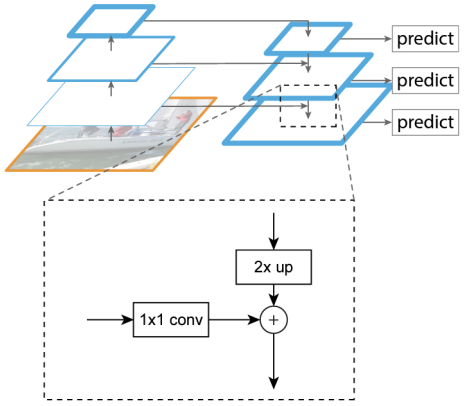

Figure 2. FPN Operation

$P7$ too. However, other than that, there are no norm or pooling layers, unlike a traditional convolutional network. The explicit stride helps with downsampling the feature map in the absence of pooling layers. The FPN network is now ready to handle a wide range of object sizes.

3.3. Training

The outputs of the FPN network are passed through classification and regression heads shown on the right of Figure 1. The outputs at each level pass through the same network heads. In this subsection, we further understand how these heads (and the FPN weights) are trained.

How does FCOS assign positive/negative samples during training?

FCOS uses dense labeling and hence does classification for each point. However, it only considers the regression loss and the centerness loss for positively labeled points. Now, a point (x, y) at level l with stride s is a positive point if and only if the point falls within the central area $(c_x - rs, c_y - rs, c_x + rs, c_y + rs)$. We also normalize the losses by the number of positive points.

What are the loss functions used in the training?

First, we discuss the classification loss. We use binary classification instead of multi-class classification and have one channel for each class. FCOS uses the sigmoid focal loss

as described in [2] because dense labeling has an extreme imbalance in classes. Most points are either background or belong to another class. We combat this imbalance by first multiplying the cross entropy loss with α_t (see section 3.1 of the original paper). In our case, α_t is α for a classified point and $1-\alpha$ otherwise. α is usually smaller than $\frac{1}{2}$ to downweigh the large losses from the less frequent class. We also multiply the loss by $(1-p_t)^\gamma$. Here, p_t directly corresponds to how confident the model is in its prediction. We put less emphasis on examples that the model is certain of and put more weight on those examples that the model is uncertain about. Finally, we normalize using the number of positive examples. The final loss is $\frac{1}{N_{pos}} \sum_{x,y} \sum_c \alpha_t (1-p_t)^\gamma BCE(p_{x,y,c}, y_c)$, where $BCE(\cdot)$ is the binary cross-entropy loss. To clarify, both positive and negative examples are valid training points for our dense labeling classification problem.

Quick note: we wish to prior the network towards predicting background. To that end, we initialize the biases to negative enough values to output small probabilities (sigmoid outputs $\frac{1}{2}$ with a zero input).

Next, we consider the regression loss that predicts the distance to the bounding box on all sides. For each positive point (and only on positive points), we compute the GIoU loss as mentioned in the paper [3]. The GIoU loss, unlike the traditional IoU loss, does not plateau for non-overlapping boxes, making it a much better loss for optimization purposes. Figure 3 demonstrates this behavior. However, unlike the classification, we calculate regression loss only for positive examples. The loss term is $\frac{1}{N_{pos}} \mathbb{1}_{c>0} \times \mathcal{L}_{GIoU}(l, r, t, b, l^*, r^*, t^*, b^*)$, where the regression targets are strided.

Finally, we consider the centerness loss. The centerness is computed using the formula $\sqrt{\frac{min(l^*,r^*)}{max(l^*,r^*)} \times \frac{min(t^*,b^*)}{max(t^*,b^*)}}$, where the distances from the current point are computed based on the ground truth bounding box. We now use the binary cross-entropy loss to learn this centerness metric. Like the regression loss, we only compute this for the positive examples and normalize it by the number of positive examples. The total loss is just the sum of these three losses.

3.4. Inference

How does FCOS decode objects at inference?

First, we collect all points that are potential centers of bounding boxes via a threshold. We do this by multiplying the object score with the centerness score. We retain the best candidates, use the regression head to compute the bounding boxes and project them onto the original image. Later we run batched non-maximum suppression to bounding boxes collected from all FPN levels and retain the best detections. For more details on the inference implementation please see Section 4.2.

What are the necessary post-processing steps?

After the non-maximum suppression step, which systematically removes overlapping boxes, we need to denormalize the bounding boxes to the original image dimension. The image may be scaled before passing to the model and the bounding boxes must be rescaled back to the original image dimension.

4. Model Inference

4.1. Classification and Regression Heads

In FCOS, the feature maps created by the different levels of the FPN are used in the two prediction heads to create the final prediction logits. The classification head outputs a list of class logit tensors, one tensor for each FPN level, of shape N x HW x C, where N is the batch size, HW is the number of center points possible on this FPN level, and C is the number of classes. The regression head outputs a list of bounding box offset regression tensors and a list of centerness score tensors, one pair of tensors for each FPN level. A bounding box regression tensor is of shape N x HW x 4 for the four offsets used to create a bounding box around that center point. A centerness score tensor is of shape N x HW x 1 for one centerness score per center point location.

For each input FPN feature map for the different FPN levels, both of the heads first use three blocks made up of a 3x3 convolution layer, a group normalization layer, and a ReLU activation. Next, in the classification head, an extra 3x3 convolution layer with a learned prior probability bias compute the final class logits. In the regression head, an extra 3x3 convolution layer and ReLU activation compute the bounding box offsets while a different 3x3 convolution layer computes the centerness scores. Final tensors are reshaped to the desired output format.

4.2. Decoding the Objects

To decode the prediction head output logits to actual bounding boxes in the scale of the original input image, an inference function was implemented. For each input image, at each FPN level, the top-k prediction candidates are kept based on object scores. These object scores are computed by $\sqrt{\sigma(logits_{class}) * \sigma(logits_{centerness})}$. For each of these top-k candidates, raw bounding box coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$ are computed from the bounding box offsets (l, t, r, b), center points (x_{center}, y_{center}), and FPN level stride s according to the following equations.

1. $x_{min} = x_{center} - l * s$
2. $y_{min} = y_{center} - t * s$
3. $x_{max} = x_{center} + r * s$
4. $y_{max} = y_{center} + b * s$

Raw bounding box coordinates are then clamped to the input image size. Finally, the top-k predictions for

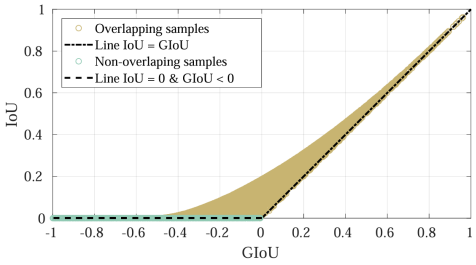

Figure 3. GIoU loss does not plateau unlike IoU loss

each FPN level are aggregated and undergo class-specific batched non-maximum suppression. A max number of best predictions for each image are kept and returned to the user.

4.3. Evaluation

A pre-trained FCOS model using a ResNet18 backbone was tested using the prediction head and inference-time decoding implementations detailed above. With a total testing time of 206.25 seconds and an average inference time per 10 images of 0.13 seconds, the final mAPs can be seen in Table 1 and some sample detection results can be seen in Figure 4.

IoU	Area	Max Detections	mAP (%)
0.50:0.95	all	100	33.3
0.50	all	100	60.5
0.75	all	100	32.5
0.50:0.95	small	100	7.1
0.50:0.95	medium	100	20.3
0.50:0.95	large	100	40.9

Table 1. The mAPs of a pre-trained FCOS model (ResNet18 backbone) with different IoU thresholds, target area sizes, and maximum number of detections per image.

5. Model Training

5.1. Multitask Loss Function

This loss function for the FCOS object detector computes the desired training supervision signals of center-ness, bounding box regression, and classification. The implementation loops over all images in a batch and vectorizes computations across all spatial locations in each feature pyramid layer.

Initial ground truth assignment designates each location as either a positive or negative sample. The criteria for positive samples are:

1. The location resides within a ground truth bounding box.
2. It is enclosed by the central region of the ground truth box.
3. The maximum distance to the target box boundaries should lie within the configured regression range for the FPN level.

Subsequently, the points satisfying all the above conditions each get a classification label derived from the ground truth box, while the remaining locations are designated as background points. When a location satisfies the three conditions for more than one ground truth box, we assign the label with the minimum area to reduce ambiguity.

For classification loss, the labels of positive samples are one-hot encoded to match the output shape of the classification header(H*W*number_classes). We use a zero vector for background points. Sigmoid focal loss is computed between the predicted and ground truth labels. We leverage the sum reduction technique (as opposed to mean) with $\alpha = 0.25$ while computing the loss.

To assign ground truth boxes to suitable feature layers, each layer is designated a receptive field range in ascending order. For each spatial location, the maximum regression coordinates of enclosing ground truth boxes are evaluated to determine the highest pyramid level whose receptive field contains the location. In the 5-layer implementation, the receptive field ranges were [(0, 64), (64, 128), (128, 256), (256, 512), (512, 1000)] for the respective layers, which is configurable.

For the regression loss, we calculate the *l*, *t*, *r*, *b* for the positive samples, which represents the left, top, right and bottom distance from the center. They are computed as follows:

$$l^* = x - x_0^i$$
$$t^* = y - y_0^i$$
$$r^* = x_1^i - x$$
$$b^* = y_1^i - y$$

These predicted values from the above equations along with the target regression outputs are passed to the *GIoU* loss function to calculate the loss. For the centerness loss, we compute a target centerness score as follows (for positive samples):

$$centerness = \sqrt{\frac{min(l^*,r^*)}{max(l^*,r^*)} \times \frac{min(t^*,b^*)}{max(t^*,b^*)}}$$

We compare the predicted centerness score from the

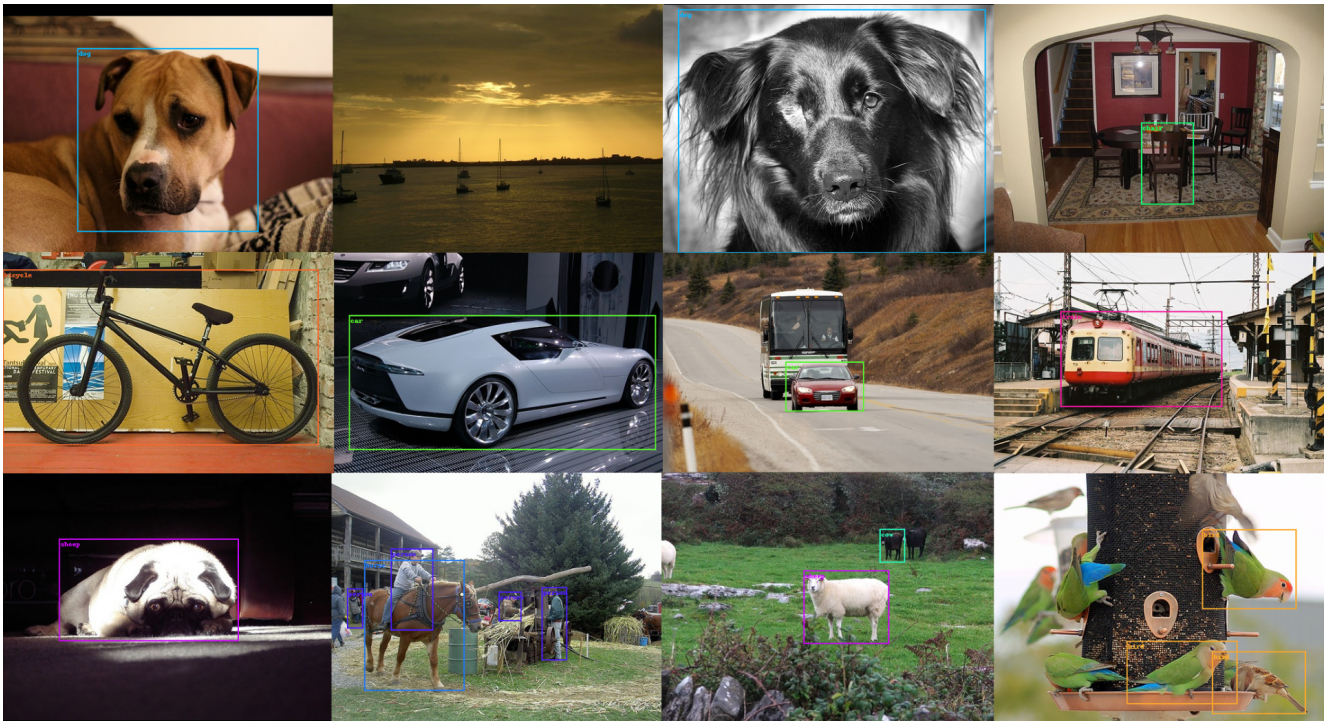

Figure 4. Sample predictions for the pre-trained FCOS model with a ResNet18 backbone.

`FCOSRegressionHead` with the target centerness score using the *binary_cross_entropy_with_logits* function to calculate the loss. Finally, we take the sum of all the above 3 losses which are normalized by the number of positive samples in the batch which would be the final loss.

5.2. Training & Evaluation

We now train the FCOS model using a ResNet18 backbone with the default config as mentioned with the pretrained FCOS model. The model was trained with 12 epochs, and it took around 1 hour to train the model with learning rate as show in Fig. 8. Respective losses can be observed at Fig. 6 and the final combined loss can be observed at Fig. 7. The final mAPs can be seen in Table 2 and some sample detection results can be seen in Figure 5.

We can clearly observe from Table 1 representing the reference pretrained model v/s Table 2 from the FCOS model trained with our computed loss, the results seem comparable and for some we can see slight increase.

IoU	Area	Max Detections	mAP (%)
0.50:0.95	all	100	**33.9**
0.50	all	100	60.5
0.75	all	100	**33.8**
0.50:0.95	small	100	**7.2**
0.50:0.95	medium	100	**20.6**
0.50:0.95	large	100	**41.9**

Table 2. The mAPs of our FCOS model (ResNet18 backbone) trained with same configuration of pretrained reference model, with different IoU thresholds, target area sizes, and maximum number of detections per image.

6. Bonus

6.1. Data Augmentation

During training, we choose a minimum size for the image at random. However, at test time, we only pick the largest from the specified values. This is a clear discrepancy between training and test time. To combat this, we consider all possible image size minimum values provided in the config. We use these different variations to augment the inference step. To achieve this, we first define a separate transform for each minimum value during initialization. Now, during inference, we apply each transform one by one in the forward step. Next, we collect the bounding boxes out of each transformation on images. Finally, we run non-maximum suppression to collect bounding boxes from different image aspect ratios and discard overlapping bounding boxes. Note that we run the NMS step only after the post-processing step to gather the bounding boxes in the same scale.

Now, we discuss the results. We extend the mAPs table with mAPs after data augmentation at test time. Please refer to Table 3 for a comparison of mAP before (Old) and

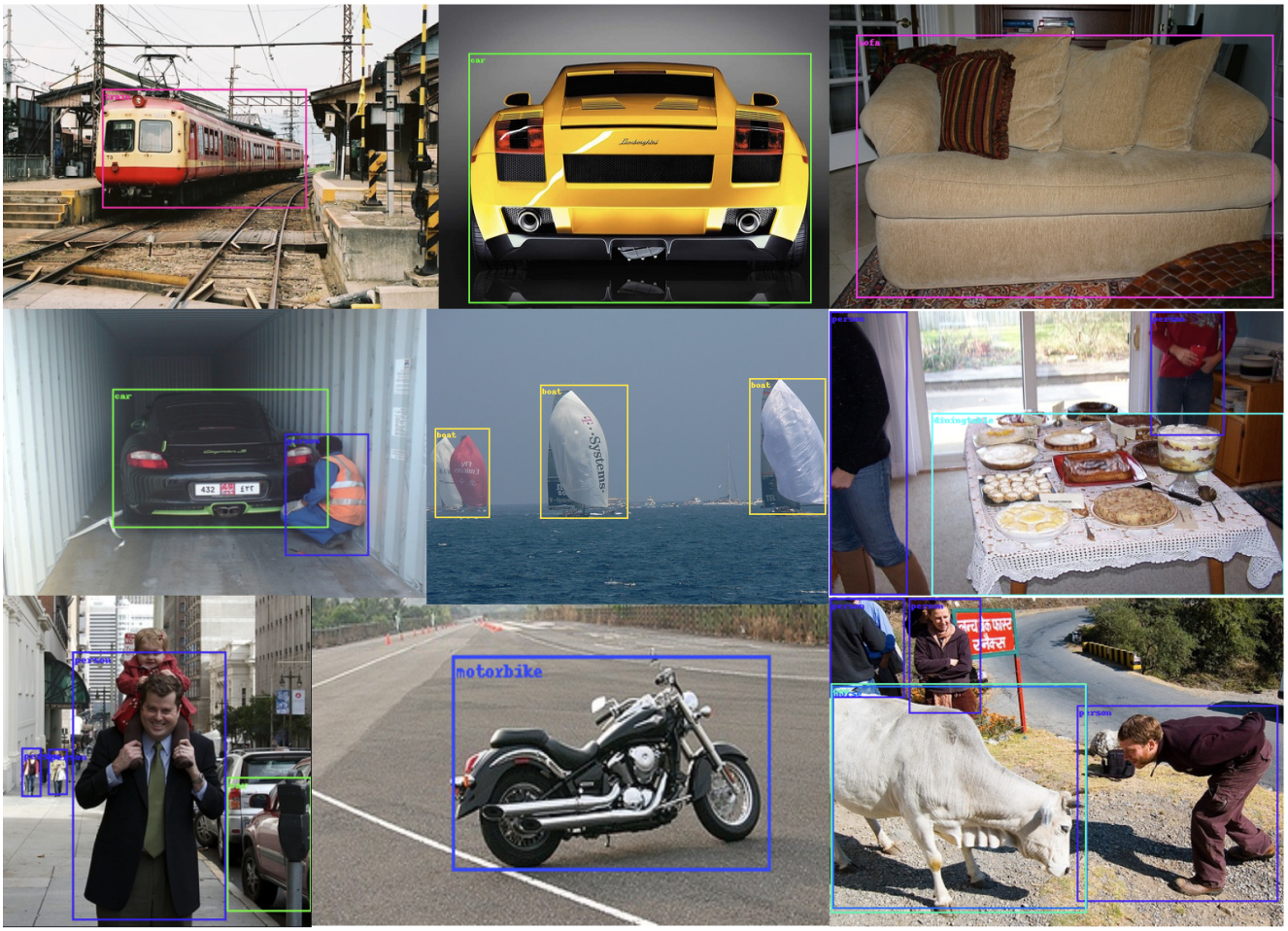

Figure 5. Sample predictions for the FCOS model with a ResNet18 backbone trained with same configuraton of pretrained reference model.

after augmentation (New). We see decent improvements, sometimes even more than 8%, except for one case where there was a slight reduction in performance. On the flip side, the inference time scales linearly with the number of image sizes we consider as can be seen in figure 9. To run this experiment for yourself, use the same pretrained model as used in 4.3 with a different configuration file: *augment.yaml*.

IoU	Old (%)	New (%)	Impr. (%)
0.50:0.95 (all)	33.3	35.1	5.4
0.50 (all)	60.5	62.1	2.6
0.75 (all)	32.5	35.3	8.6
0.50:0.95 (s)	7.1	7.2	1.4
0.50:0.95 (m)	20.3	20.1	-1
0.50:0.95 (l)	40.9	44	7.6

Table 3. A comparison between the mAPs of the provided pretrained FCOS model before and after data augmentation.

6.2. Deeper Models

We tried to evaluate with deeper models. Table 4 shows the deeper models give increase performance over our default model as mentioned in 5. With Fig. 10 we can compare visually the performance of different models.

IoU	resnet18	resnet34	resnet50	resnet101
0.50:0.95 (all)	33.9	38.2	39.2	**39.5**
0.50 (all)	60.5	65.4	67.3	**67.4**
0.75 (all)	33.8	39.2	**40.2**	**40.2**
0.50:0.95 (s)	7.2	8.6	**9.7**	**9.7**
0.50:0.95 (m)	20.6	24.7	**27.6**	27.1
0.50:0.95 (l)	41.9	46.3	46.8	**47.3**

Table 4. The mAPs (in %) of our FCOS model with different backbones trained with same configuration of pretrained reference model, with different IoU thresholds, target area sizes, and maximum number of detections per image. (s=small, m=medium, l=large)

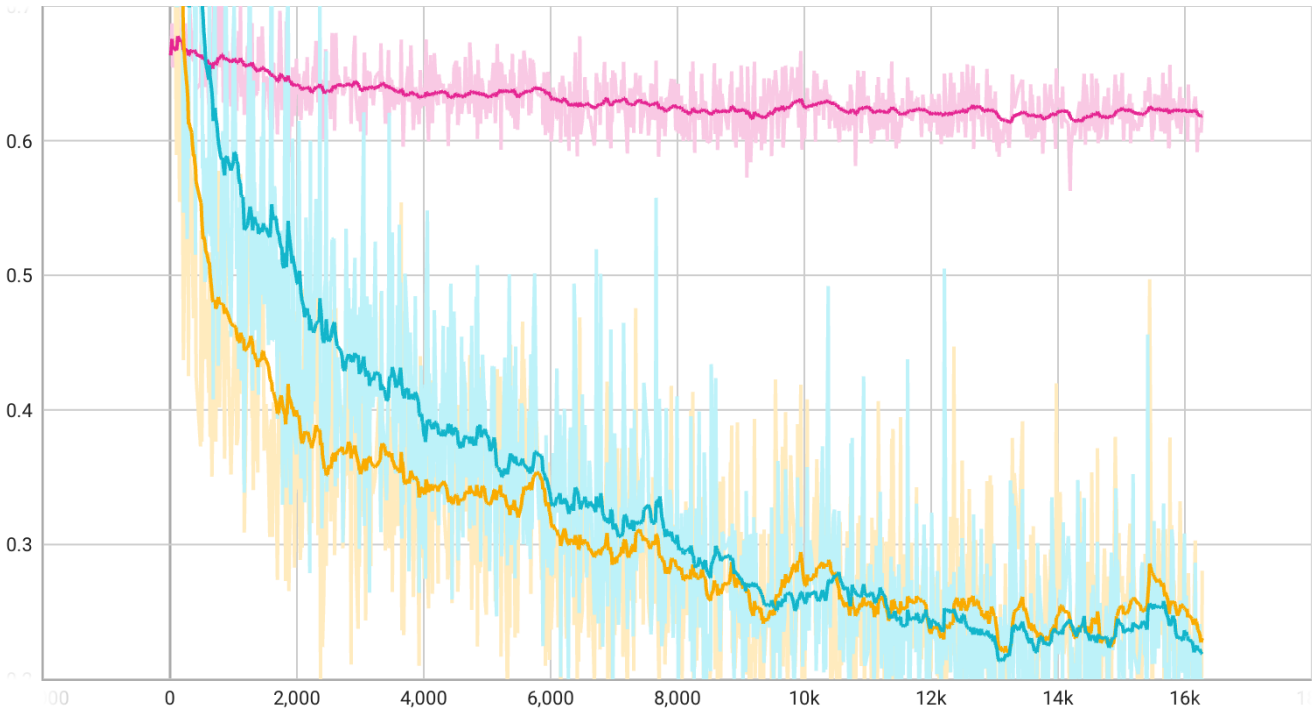

Figure 6. FCOS with ResNet18 backbone - Classification loss (blue), Regression Loss (orange), Centerness Loss (pink)

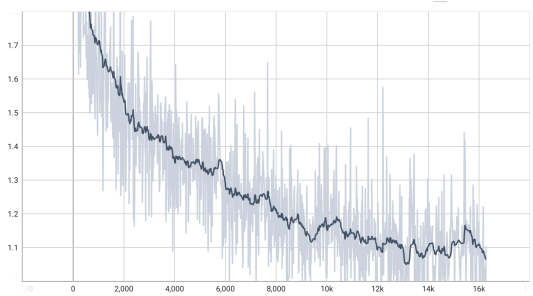

Figure 7. FCOS with ResNet18 backbone - Final combined loss

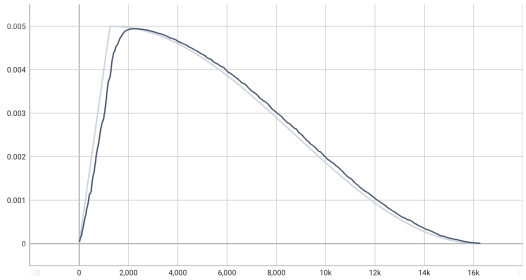

Figure 8. FCOS with ResNet18 backbone - Learning Rate

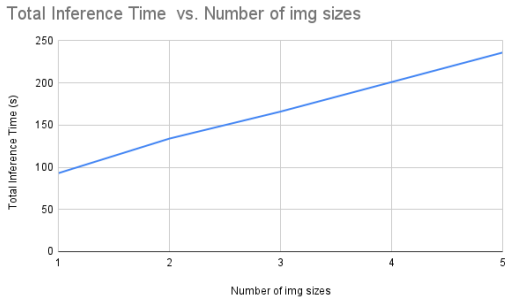

Figure 9. Inference time as a function of number of augmentations.

6.3. Hyper parameter Tuning

6.3.1 NMS Threshold

We tried to infer with different NMS threshold on our ResNet 18 backbone trained as mentioned in 5 and validate if varying the threshold value would affect the performance. From Table 5 we can observe that the lower values are better and as we increase the threshold the performance degrades.

6.3.2 Epochs

Clearly, if we observe from Fig. 11, there is a scope for training even further since loss is not saturating. So, we

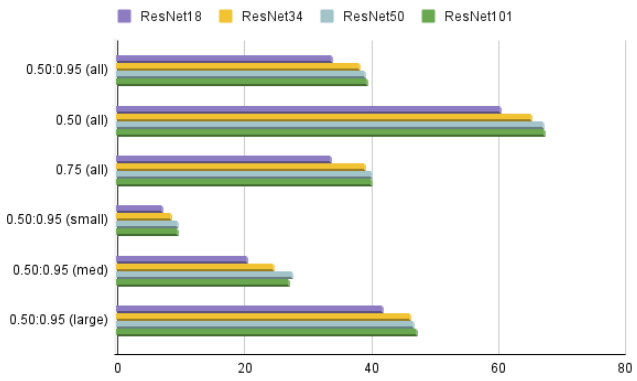

Figure 10. Deeper model comparison

IoU	Area	(nms=0.5)	(nms=0.6)	(nms=0.7)
0.50:0.95	all	33.8	33.9	33.6
0.50	all	61.5	60.5	58.5
0.75	all	32.9	33.8	34.8
0.50:0.95	small	7.1	7.2	7.0
0.50:0.95	medium	20.4	20.6	20.3
0.50:0.95	large	41.7	41.9	41.6

Table 5. The mAPs of our FCOS model (ResNet34 backbone) trained with same configuration of pretrained reference model, with different IoU thresholds, target area sizes with different NMS threshold values.

trained the model with more iterations and we observed improvements in the results. The results are captured with Table 6. We can also see the loss is almost saturating after 8k iterations in Fig. 12.

IoU	Area	(epochs=12)	(epochs=30)
0.50:0.95	all	38.2	**38.7**
0.50	all	65.4	**67.4**
0.75	all	39.2	**39.2**
0.50:0.95	small	8.6	8.6
0.50:0.95	medium	24.7	**25.6**
0.50:0.95	large	46.3	**47.0**

Table 6. The mAPs of our FCOS model (ResNet34 backbone) trained with same configuration of pretrained reference model, with different IoU thresholds, target area sizes with multiple epochs.

6.3.3 Number of Convolution Layers

We also evaluated if the number of convolution layers has any effect on performance. This refers to the `num_convs` parameter in `FCOSClassificationHead` and `FCOSRegressionHead`. We can observe from Table 7 that having fewer conv layers with 3 layers over 4, showed a slight increase in performance.

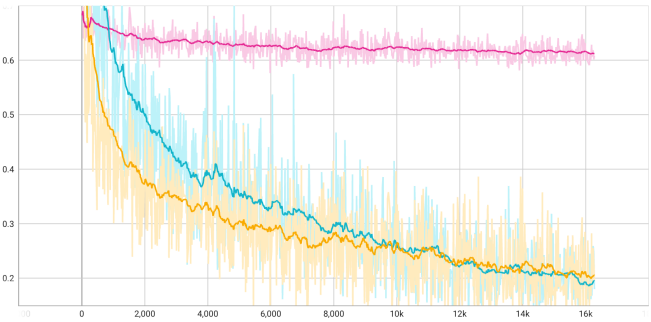

Figure 11. FCOS (ResnNet34 backbone) loss visualization with epochs=12

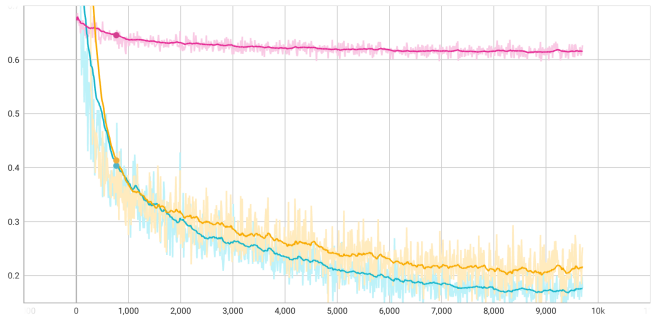

Figure 12. FCOS (ResnNet34 backbone) loss visualization with epochs=30

IoU	Area	(num_conv=4)	(num_conv=3)
0.50:0.95	all	38.0	38.2
0.50	all	64.8	65.4
0.75	all	39.0	39.2
0.50:0.95	small	8.0	8.6
0.50:0.95	medium	24.6	24.7
0.50:0.95	large	46.3	46.3

Table 7. The mAPs of our FCOS model (ResNet34 backbone) trained with same configuration of pretrained reference model, with different IoU thresholds, target area sizes with different number of conv layers.

References

[1] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017. 1

[2] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018. 3

[3] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized in-

tersection over union: A metric and a loss for bounding box regression, 2019. 3

[4] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: A simple and strong anchor-free object detector. *CoRR*, abs/2006.09214, 2020. 1