CS 771 Assignment 3 Writeup

November 2023

Table 1: Contributions

Team Member (lexicographically in last name)	**Contribution**
Ruijia Chen	Section 1, 2, 3
Zhiqi Gao	Section 1, 3
Boyuan Zou	Section 3, 4

1 Model Design

1.1 What is the output of the backbone?

The output of the backbone is a set of feature maps denoted as F_i, where i represents the layer of the backbone network. These feature maps have a size of H×W×C, where H and W are the height and width, and C is the number of classes. The backbone network processes input data into a certain feature representation from the input image, and each F_i represents different aspects of the input data. These feature maps will be passed on to the subsequent layers of the network.

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

FPN considers five levels in total: P3, P4, P5, P6, and P7. These levels are produced by the feature maps C3, C4, C5, P5, and P6, respectively.

To generate its output feature maps, FPN utilizes top-down and lateral connections.

The FPN starts by using convolution to the feature map C5 to transform it to the P5, which gets high-level semantic features.

Next, FPN adds lateral connections to merge the lower-level features (C4, C3) with the upsampled higher-level features (P5, P4) to create P4 and P3 respectively.

For example, features from C4 are transformed to match the number of channels in P5. We then create P4 by adding the upsampled features from P5 to the features from C4. And we repeat the process to create P3 with P4 and C3.

P6 and P7 are produced by applying one 3×3 convolutional layer with the stride being 2 on P5 and P6 with padding 1, respectively. For example, the provided code achieves this by self.p6 = nn.Conv2d (in_channels, out_channels, 3, 2, 1).

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

Location (x, y) is considered as a positive sample if it falls into the center area of any ground-truth box, by following. The center area of a box centered at(c_x, c_y) is defined as the sub-box $(c_x - rs, c_y - rs, c_x + rs, c_y + rs)$, where s is the total stride until the current feature maps and r is a hyper-parameter being 1.5 on COCO. The sub-box is clipped so that it is not beyond the original box. The class label c^* of the location is the class label of the ground-truth box. Otherwise, it is a negative sample and c^* = 0 (background class). If a location falls into the center area of multiple bounding boxes, it is considered as an ambiguous sample. The authors simply choose the bounding box with minimal area as its regression target.

The loss function in FCOS consists of two components: one is the classification loss, and the other is the bounding box regression loss. As for the classification loss, it employs the focal loss from RetinaNet. The loss function in the training is defined as follows:

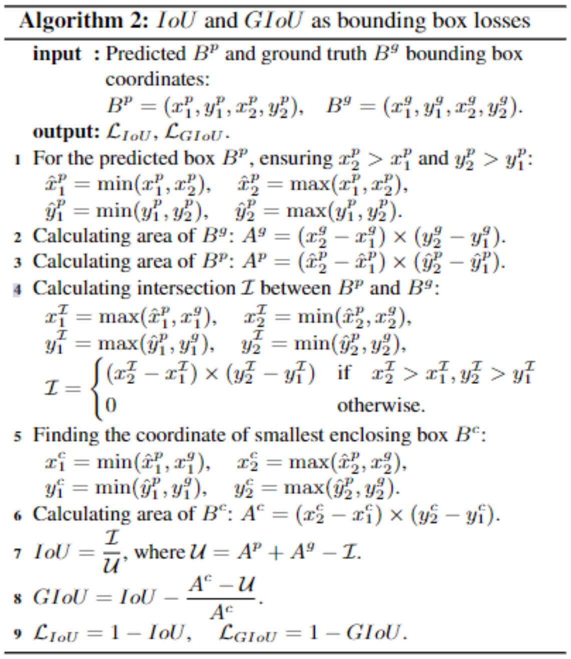
$$L(\{p_{x,y}\}, \{t_{x,y}\}) = \frac{1}{N_{\text{pos}}} \sum_{x,y} L_{\text{cls}}(p_{x,y}, c^*_{x,y}) + \frac{\lambda}{N_{\text{pos}}} \sum_{x,y} \mathbb{1}_{\{c^*_{x,y}>0\}} L_{\text{reg}}(t_{x,y}, t^*_{x,y}) \tag{1}$$

where L_{cls} is focal loss and L_{reg} is the GIoU loss. N_{pos} denotes the number of positive samples and λ being 1 in this paper is the balance weight for L_{reg}. The summation is calculated over all locations on the feature maps F_i. $\mathbb{1}_{\{c_i^*>0\}}$ is the indicator function, being 1 if $c_i^* > 0$ and 0 otherwise.

The focal loss is defined as (focusing parameter $\gamma \geq 0$):

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t) \tag{2}$$

The GIoU loss is defined as:



Algorithm 2: IoU and $GIoU$ as bounding box losses

```
input  : Predicted $B^p$ and ground truth $B^g$ bounding box
         coordinates:
         $B^p = (x_1^p, y_1^p, x_2^p, y_2^p)$,   $B^g = (x_1^g, y_1^g, x_2^g, y_2^g)$.
output: $\mathcal{L}_{IoU}$, $\mathcal{L}_{GIoU}$.
1 For the predicted box $B^p$, ensuring $x_2^p > x_1^p$ and $y_2^p > y_1^p$:
    $\hat{x}_1^p = \min(x_1^p, x_2^p)$,   $\hat{x}_2^p = \max(x_1^p, x_2^p)$,
    $\hat{y}_1^p = \min(y_1^p, y_2^p)$,   $\hat{y}_2^p = \max(y_1^p, y_2^p)$.
2 Calculating area of $B^g$: $A^g = (x_2^g - x_1^g) \times (y_2^g - y_1^g)$.
3 Calculating area of $B^p$: $A^p = (\hat{x}_2^p - \hat{x}_1^p) \times (\hat{y}_2^p - \hat{y}_1^p)$.
4 Calculating intersection $\mathcal{I}$ between $B^p$ and $B^g$:
    $x_1^{\mathcal{I}} = \max(\hat{x}_1^p, x_1^g)$,   $x_2^{\mathcal{I}} = \min(\hat{x}_2^p, x_2^g)$,
    $y_1^{\mathcal{I}} = \max(\hat{y}_1^p, y_1^g)$,   $y_2^{\mathcal{I}} = \min(\hat{y}_2^p, y_2^g)$,
    $\mathcal{I} = \begin{cases} (x_2^{\mathcal{I}} - x_1^{\mathcal{I}}) \times (y_2^{\mathcal{I}} - y_1^{\mathcal{I}}) & \text{if} \quad x_2^{\mathcal{I}} > x_1^{\mathcal{I}}, y_2^{\mathcal{I}} > y_1^{\mathcal{I}} \\ 0 & \text{otherwise.} \end{cases}$
5 Finding the coordinate of smallest enclosing box $B^c$:
    $x_1^c = \min(\hat{x}_1^p, x_1^g)$,   $x_2^c = \max(\hat{x}_2^p, x_2^g)$,
    $y_1^c = \min(\hat{y}_1^p, y_1^g)$,   $y_2^c = \max(\hat{y}_2^p, y_2^g)$.
6 Calculating area of $B^c$: $A^c = (x_2^c - x_1^c) \times (y_2^c - y_1^c)$.
7 $IoU = \frac{\mathcal{I}}{\mathcal{U}}$, where $\mathcal{U} = A^p + A^g - \mathcal{I}$.
8 $GIoU = IoU - \frac{A^c - \mathcal{U}}{A^c}$.
9 $\mathcal{L}_{IoU} = 1 - IoU$,   $\mathcal{L}_{GIoU} = 1 - GIoU$.
```

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

FCOS decodes objects at inference by forwarding the input image through the network and obtaining the predicted bounding boxes by classification scores $p_{x,y}$ and regression prediction $t_{x,y}$ for each location on the feature maps F_i. If the classification score $p_{x,y}$ is greater than 0.05, indicating a high confidence in the presence of an object at that location, then the authors designate the location as a positive sample. The predicted bounding boxes are then derived by inverting Equation 3.

$$\begin{aligned} l^* = (x - x_0^{(i)})/s,\ t^* = (y - y_0^{(i)})/s, \\ r^* = (x_1^{(i)} - x)/s,\ b^* = (y_1^{(i)} - y)/s \end{aligned} \tag{3}$$

Non-Maximum Suppression (NMS) is a necessary post-processing step. It is used to remove duplicate or highly overlapping bounding boxes. During NMS, bounding boxes are sorted by their confidence scores, and boxes with significant overlap are suppressed, leaving only the most confident and non-overlapping bounding boxes. In FCOS, the post-processing hyper-parameters are the same as in RetinaNet except that the authors use NMS threshold 0.6 instead of 0.5 in RetinaNet. As claimed in the abstract, NMS is the only post-processing step.

2 Model Inference

2.1 Classification and Regression Heads

For the forward function in FCOSClassificationHead, we iterate through each feature map in the input x, apply convolution using self.conv, and then pass the results through the classification layer self.cls_logits to obtain classification logits. The logits are reshaped as (N, HxW, C) and stored in a list called classification_logits. The function returns this list containing classification logits for each input feature map.

For the forward function in FCOSRegressionHead, we do similar things. We iterate through each feature map in the input x, apply convolution using self.conv, and then extract regression outputs and center-ness scores. The regression outputs list contains tensors of size (N, HxW, 4), while the center-ness scores list contains tensors of size (N, HxW).

We do some re-arrangement of the outputs here for later training and inference.

2.2 Decoding the Objects

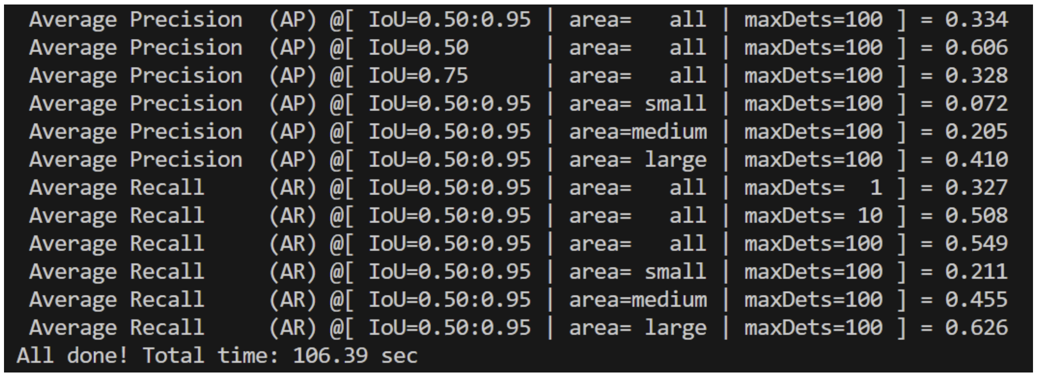
First of all, we apply the sigmoid activation function to the classification logits (cls_logits) and centerness logits (ctr_logits). This activation transforms the raw model outputs into probability-like values between 0 and 1. Then we follow the instructions in the comments. We loop over every image and every pyramid level, compute the object scores with the equation $\sqrt{cls\_logits \times ctr\_logits}$. Then we filter out boxes with object scores below the threshold. We select the top K boxes with self.topk_candidates, decode the boxes, and clip boxes outside of the image boundaries and remove small boxes. Our criterion for small boxes is that x2 is less than or equal to x1 or y2 is less than or equal to y1. As for the decoding, we compute with regression outputs and strides as follows:

```
# (4) Decode the boxes
box_x0=per_locations[:, 1]-per_box_regression[:, 0]*strides[level]
box_y0=per_locations[:, 0]-per_box_regression[:, 1]*strides[level]
```

```
box_x1=per_locations[:, 1]+per_box_regression[:, 2]*strides[level]
box_y1=per_locations[:, 0]+per_box_regression[:, 3]*strides[level]
```

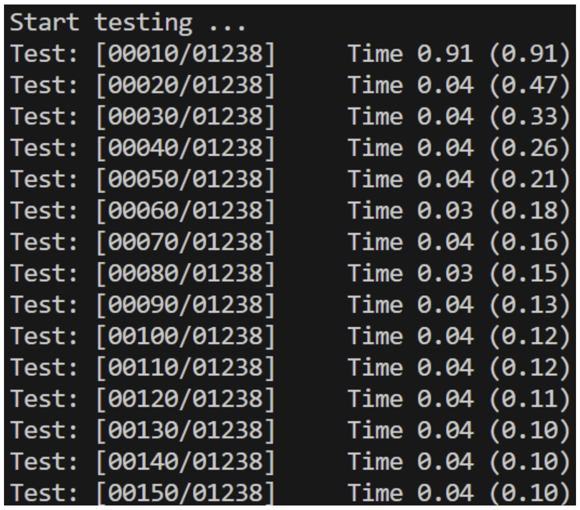
Then we collect all candidate boxes across all pyramid levels using torch.cat. We run non-maximum suppression to remove duplicated boxes within each category with 'batched_nms' function and keep the top K boxes after NMS. Finally, we append the detections for each image to a list and return the list.

We conducted our experiments on a laptop equipped with 32GB of RAM and a NVIDIA GeForce RTX 4080 Laptop GPU. Figure 1 shows our mAP scores, matching the expected mAP@0.5 of 60.6%. The total inference time is 106.39 seconds. The actual inference time for some mini-batches are shown in Figure 2.



```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.334
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.606
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.328
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.072
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.205
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.410
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.327
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.508
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.549
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.211
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.455
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.626
All done! Total time: 106.39 sec
```

Figure 1: the mAP scores for inference



```
Start testing ...
Test: [00010/01238]     Time 0.91 (0.91)
Test: [00020/01238]     Time 0.04 (0.47)
Test: [00030/01238]     Time 0.04 (0.33)
Test: [00040/01238]     Time 0.04 (0.26)
Test: [00050/01238]     Time 0.04 (0.21)
Test: [00060/01238]     Time 0.03 (0.18)
Test: [00070/01238]     Time 0.04 (0.16)
Test: [00080/01238]     Time 0.03 (0.15)
Test: [00090/01238]     Time 0.04 (0.13)
Test: [00100/01238]     Time 0.04 (0.12)
Test: [00110/01238]     Time 0.04 (0.12)
Test: [00120/01238]     Time 0.04 (0.11)
Test: [00130/01238]     Time 0.04 (0.10)
Test: [00140/01238]     Time 0.04 (0.10)
Test: [00150/01238]     Time 0.04 (0.10)
```

Figure 2: the actual inference time for some mini-batches

Figure 3 shows our sample detection results. We successfully detected the box corresponding to the category.

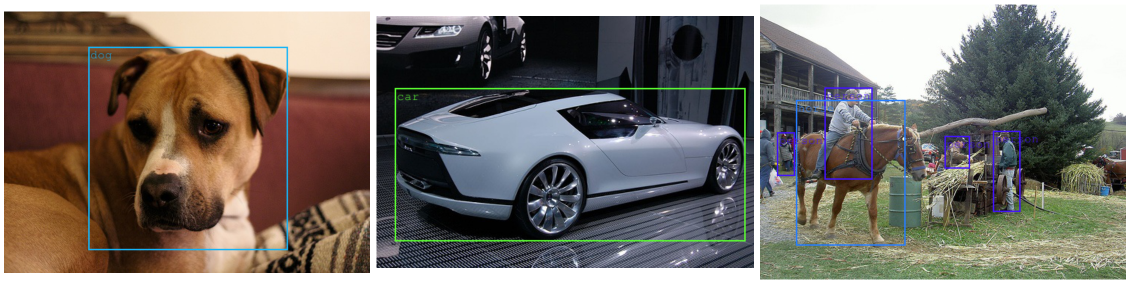
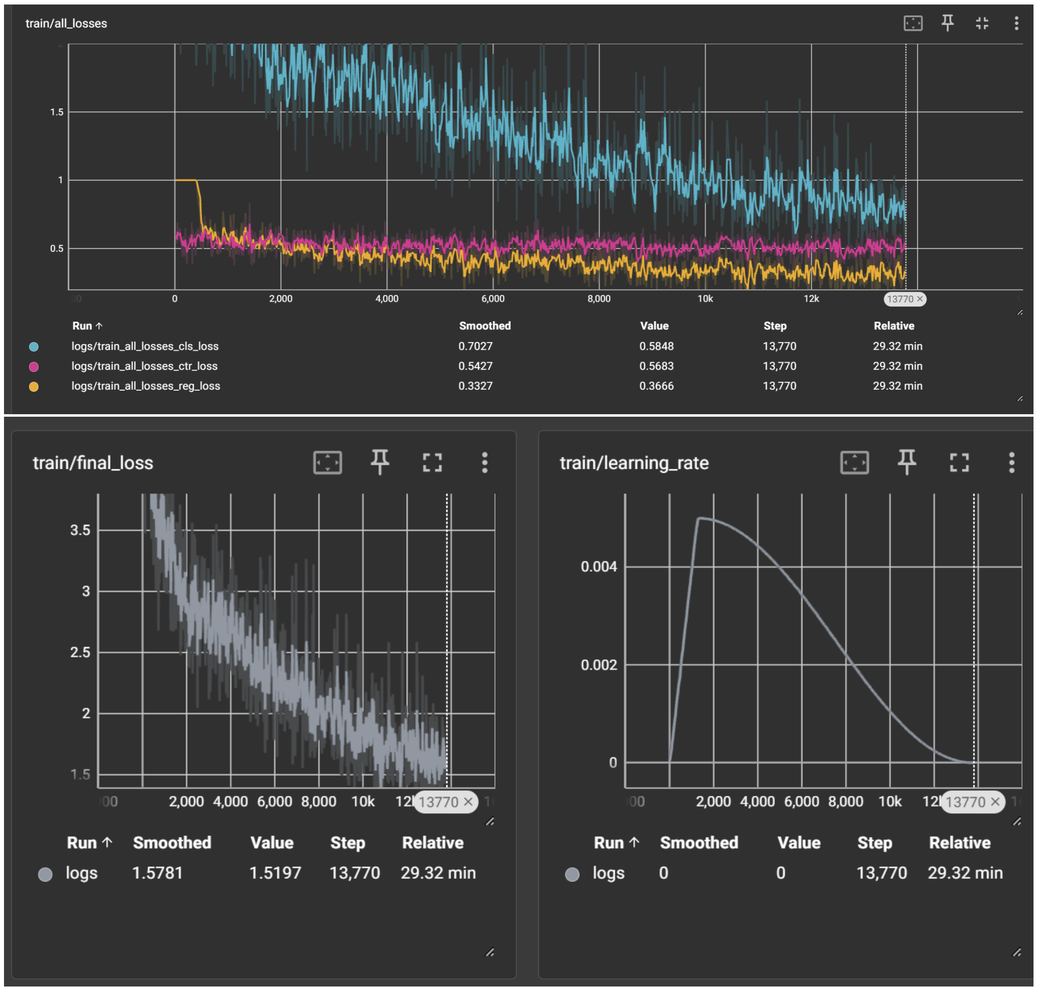


Figure 3: sample detection results for inference

3 Model Training

Firstly, we reshape the 2D coordinate points for each feature level as (Level,HxW,2) for subsequent operations. Then we collect information about ground truth bounding boxes, labels, and areas from the target dictionary. We perform center sampling by determining whether each point lies within the center of a bounding box, and we select valid foreground points based on conditions of being inside the bounding box, within the regression range, and in the center region. We encode regression targets based on the distance between points and bounding box boundaries, according to Equation 3. Then we flatten and concatenate various tensors to prepare them for loss computation. For regression loss and center-ness loss, we only choose the foreground (positive) points for loss computation. We compute the classification, regression, and center-ness losses using focal loss, GIoU loss, and binary cross-entropy loss, respectively. Lastly we compute the final loss as the sum of classification, regression, and center-ness losses.

We conducted our experiments on a laptop equipped with 32GB of RAM and a NVIDIA GeForce RTX 4080 Laptop GPU. Figure 4 shows training curves of our model, and Figure 5 shows our testing mAP scores. The total test time is 134.88 seconds.

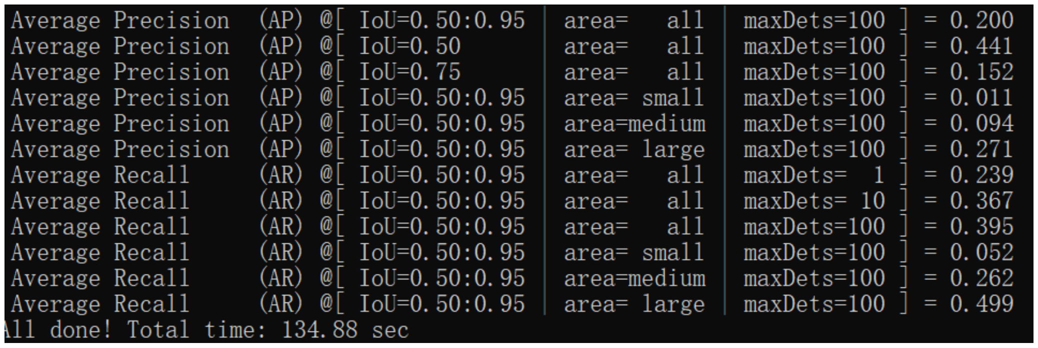


Run ↑	Smoothed	Value	Step	Relative
● logs/train_all_losses_cls_loss	0.7027	0.5848	13,770	29.32 min
● logs/train_all_losses_ctr_loss	0.5427	0.5683	13,770	29.32 min
● logs/train_all_losses_reg_loss	0.3327	0.3666	13,770	29.32 min

Run ↑	Smoothed	Value	Step	Relative
● logs	1.5781	1.5197	13,770	29.32 min

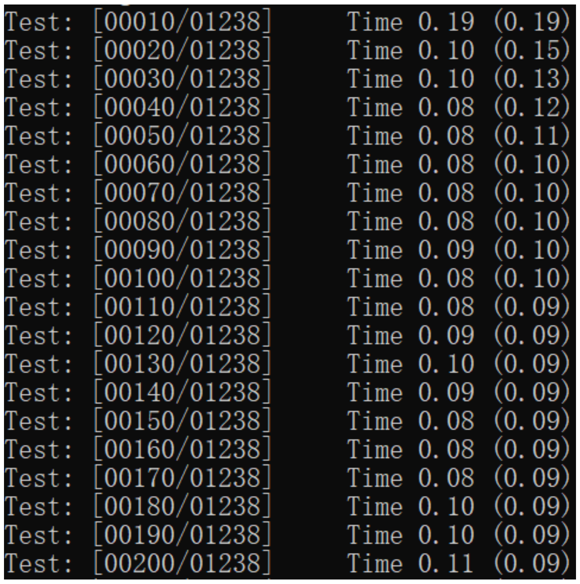
Run ↑	Smoothed	Value	Step	Relative
● logs	0	0	13,770	29.32 min

Figure 4: training curves of our model



```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.200
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.441
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.152
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.011
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.094
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 ] = 0.239
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.367
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.395
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.052
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.262
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.499
ll done! Total time: 134.88 sec
```

Figure 5: the mAP scores for model training



```
Test: [00010/01238]    Time 0.19 (0.19)
Test: [00020/01238]    Time 0.10 (0.15)
Test: [00030/01238]    Time 0.10 (0.13)
Test: [00040/01238]    Time 0.08 (0.12)
Test: [00050/01238]    Time 0.08 (0.11)
Test: [00060/01238]    Time 0.08 (0.10)
Test: [00070/01238]    Time 0.08 (0.10)
Test: [00080/01238]    Time 0.08 (0.10)
Test: [00090/01238]    Time 0.09 (0.10)
Test: [00100/01238]    Time 0.08 (0.10)
Test: [00110/01238]    Time 0.08 (0.09)
Test: [00120/01238]    Time 0.09 (0.09)
Test: [00130/01238]    Time 0.10 (0.09)
Test: [00140/01238]    Time 0.09 (0.09)
Test: [00150/01238]    Time 0.08 (0.09)
Test: [00160/01238]    Time 0.08 (0.09)
Test: [00170/01238]    Time 0.08 (0.09)
Test: [00180/01238]    Time 0.10 (0.09)
Test: [00190/01238]    Time 0.10 (0.09)
Test: [00200/01238]    Time 0.11 (0.09)
```

Figure 6: the actual test time for some mini-batches

Figure 7 shows our sample detection results. We successfully detected the box corresponding to the category. Our model training config file is default './config/voc fcos.yaml'.

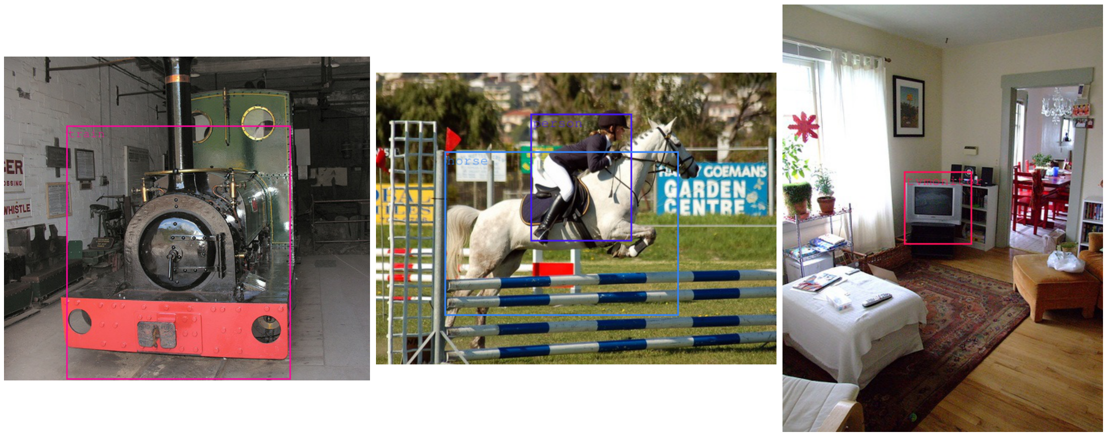


Figure 7: sample detection results for model training