16824 homework 2

jiaqigen

March 2021

Task 0

0.1 What classes does the image at index 2020 contain

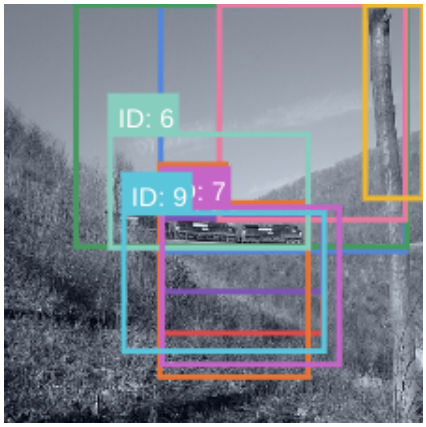
The train class

0.2 What is the easiest way to select the most informative regions

If we want n most informative regions, we simply select the top n proposals with highest scores.

0.3 Use Wandb to visualize the ground-truth bounding box and the class for the image at index 2020

0.4 Use Wandb to visualize the top ten bounding box proposals for the image at index 2020



1 Task 1

1.1 In the report, for each of the TODO, describe the functionality of that part

TODOs in task1.py

1: Define BCELoss and SGD optimizer, plug in parameters like weight decay and learning rate.

2: Set up the VOCDataset (resize image to 512 * 512, load top 300 proposals).

3: Initialize wandb

4: Potentially change the arguments for the train function

5: Get inputs and targets from the train loader

6: Get output from model

7: Perform any necessary functions on the output such as clamping

8: Compute loss for train

9: Compute gradient and do SGD step

10 and 11: Visualization

12: Get inputs and targets from the val loader

13: Get output from model

14: Perform any necessary functions on the output such as clamping

15: Compute loss for val

16 and 17: Visualization

18: save the trained model (potentially modify the parameters for the saving function)

TODOs in AlexNet.py

1: Define the architecture for LocalizerAlexNet (similar to AlexNet)

2: Perform forward pass (data will go through features and classifier)

3: Load pretrained AlexNet weights to our model

1.2 What is the output resolution of the model

The output resolution is (batch size $* 20 * 29 * 29$), where 20 is the number of classes.

1.3 Use wandb to plot the training loss curve; Use wandb to plot images and the rescaled heatmaps for only the GT classes for 2 batches

Training Loss Curve:

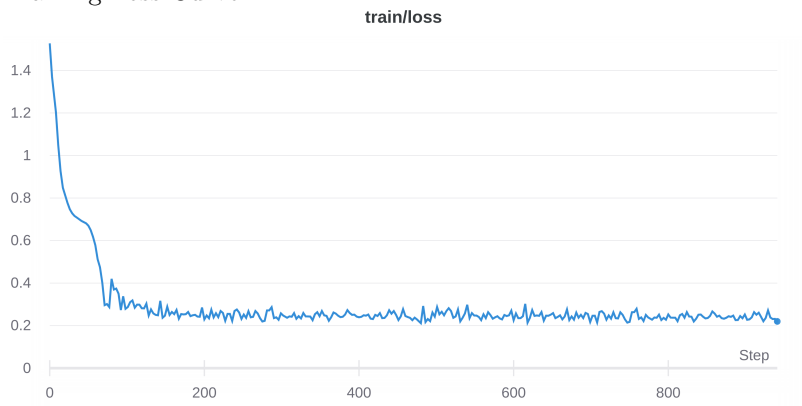
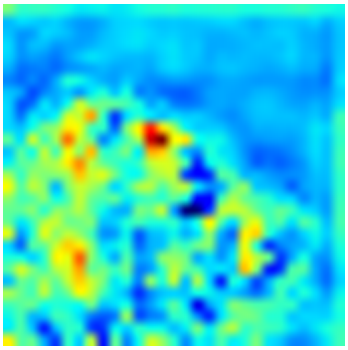


Image 1:

First epoch heatmap:



Second epoch heatmap:

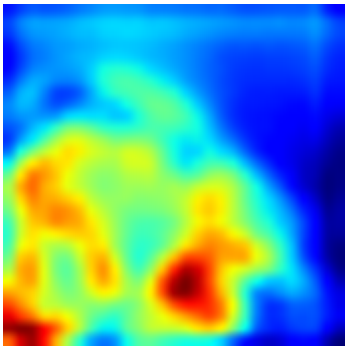
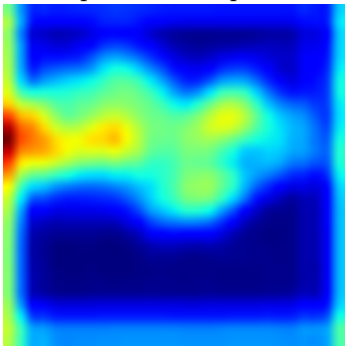
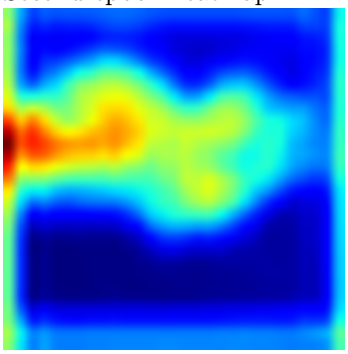


Image 2:

First epoch heatmap:



Second epoch heatmap:



1.4 In the first few iterations, you should observe a steep drop in the loss value. Why does this happen?

I printed out the model outputs for the first iterations and it looks like the model tries to predict everything with values close to zero. I think this is because our ground truth contains many zeros and only a few ones. Therefore, predicting everything with zero will quickly decrease the loss value. After the first few iterations, the loss can only be decreased further by actually learning from the ground truth class.

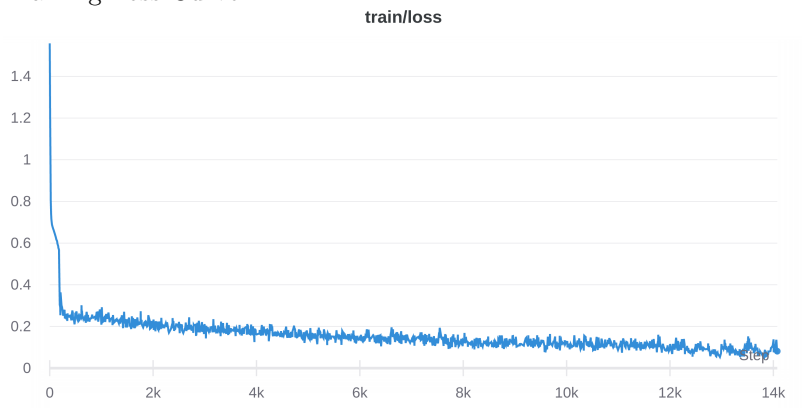
1.5 We will log two metrics during training to see if our model is improving progressively with iterations

Metric 1 is mAP, and metric 2 is Recall.

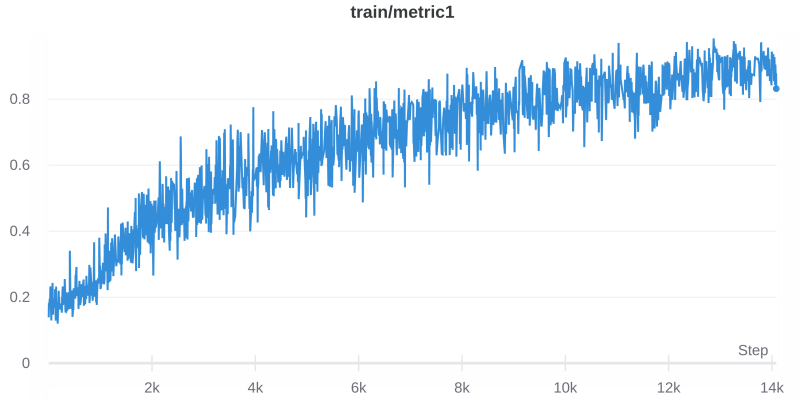
See the code for details.

1.6 Train the model using batchsize=32, learning rate=0.01, epochs=30. Evaluate every 2 epochs.

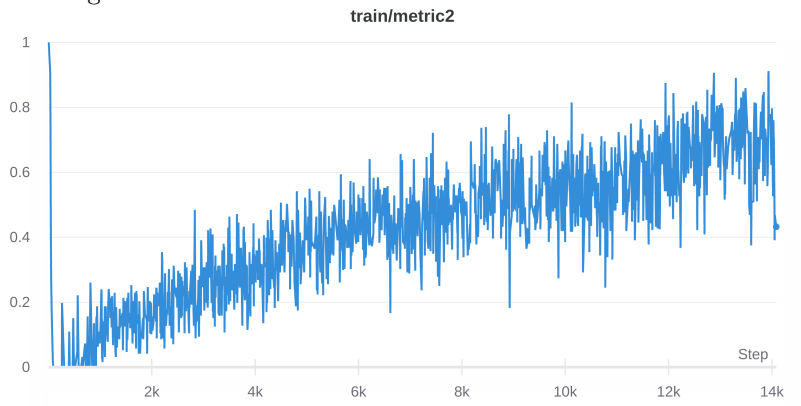
Training Loss Curve:



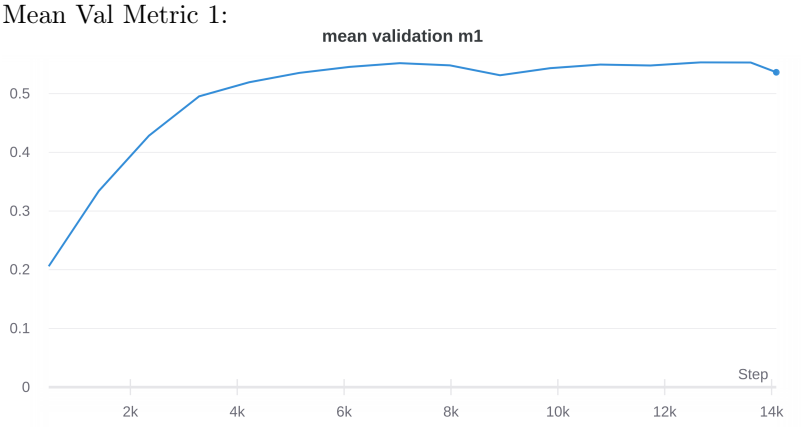
Training Metric 1:



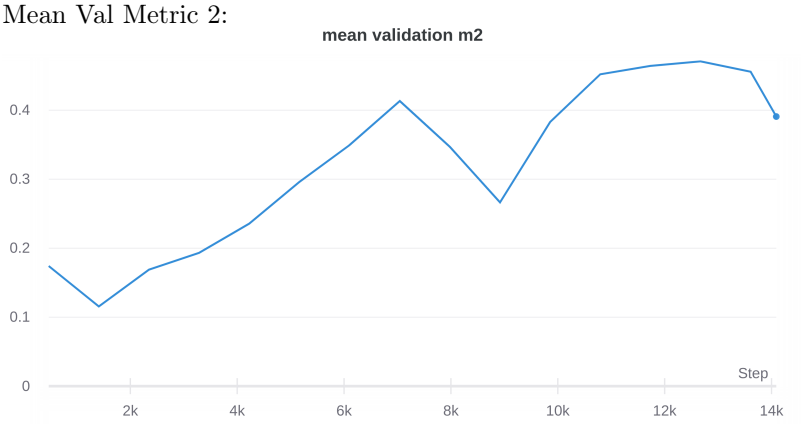
Training Metric 2:



Mean Val Metric 1:



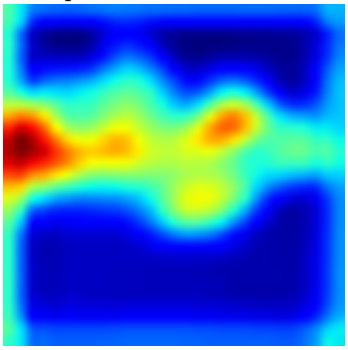
Mean Val Metric 2:



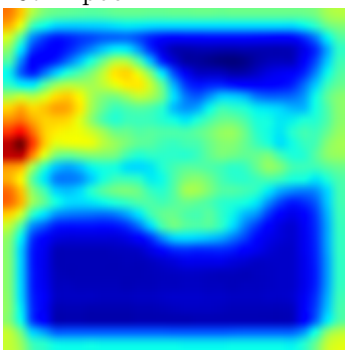
Final train metric1 (average): 0.903
Final train metric2 (average): 0.690
Final train loss (average): 0.0861
Final val metric1 (average): 0.537
Final val metric2 (average): 0.393

Image:

1st Epoch:



15th Epoch:



30th Epoch:

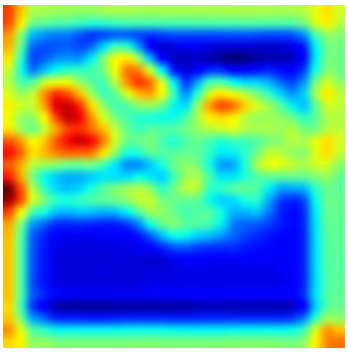
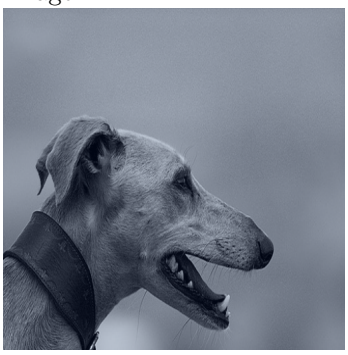
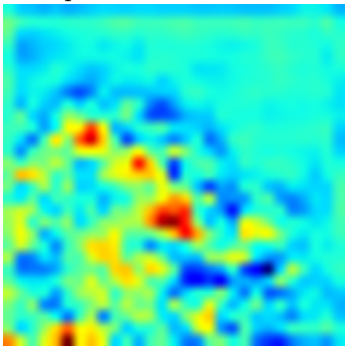


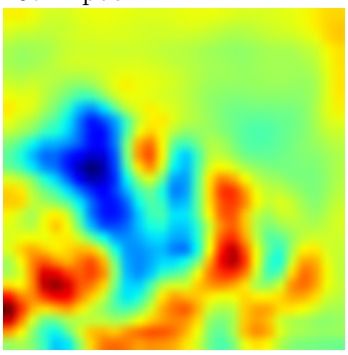
Image:



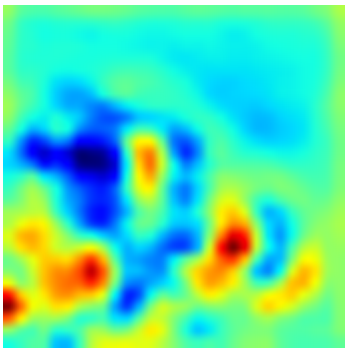
1st Epoch:



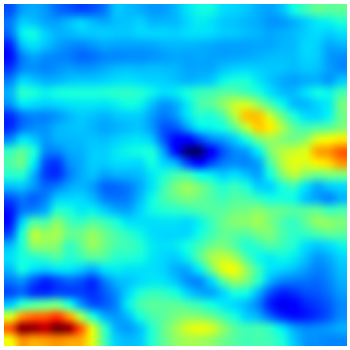
15th Epoch:



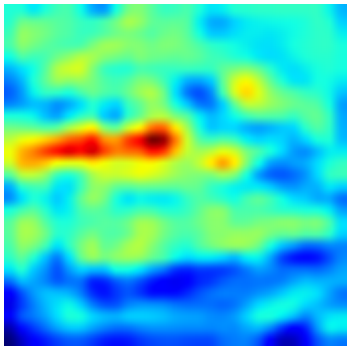
30th Epoch:



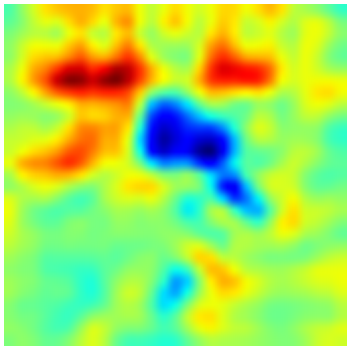
Random Image 1:

Random Image 2:

Random Image 3:

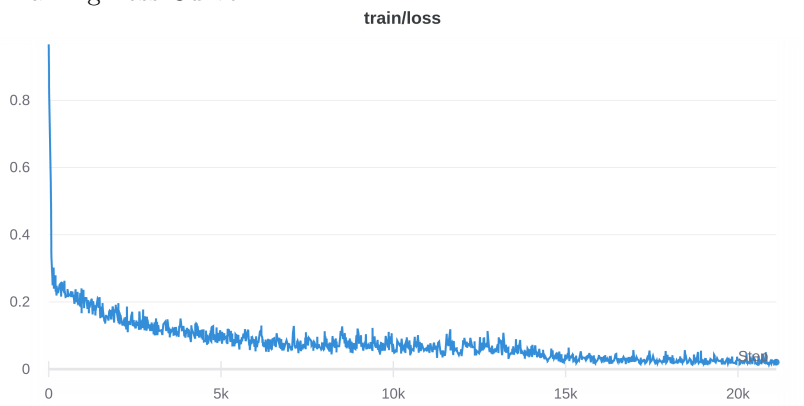



1.7 In the heatmap visualizations you observe that there are usually peaks on salient features of the objects but not on the entire objects. How can you fix this in the architecture of the model?

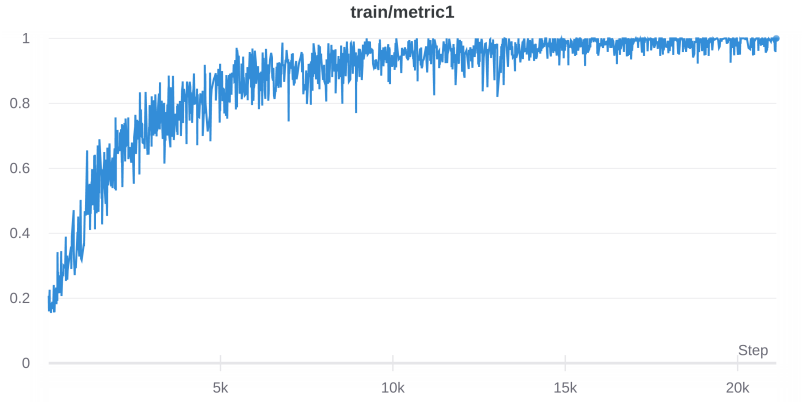
I added a average pooling layer at the end of the LocalizerAlexNetRobust model.

I also adjust the weight decay value in order to deal with the overfitting problem that we have before.

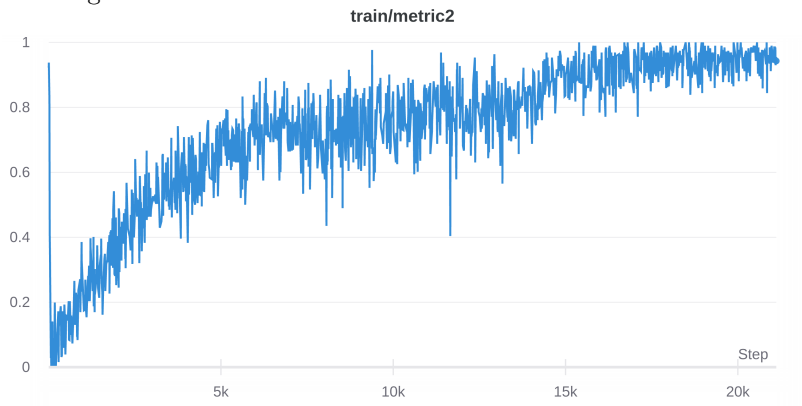
Training Loss Curve:



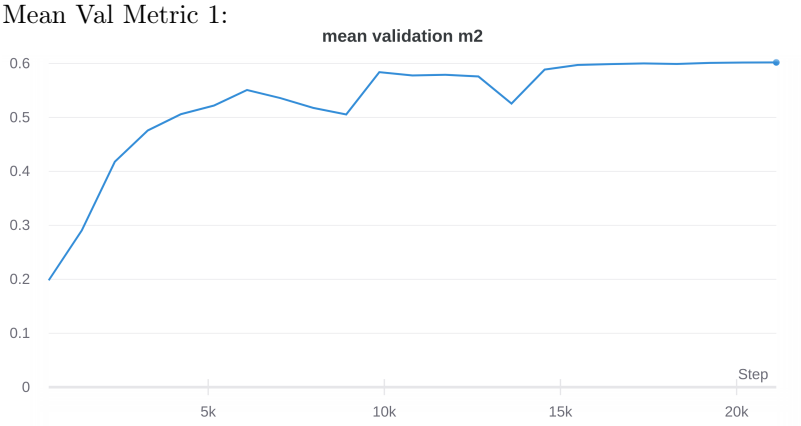
Training Metric 1:



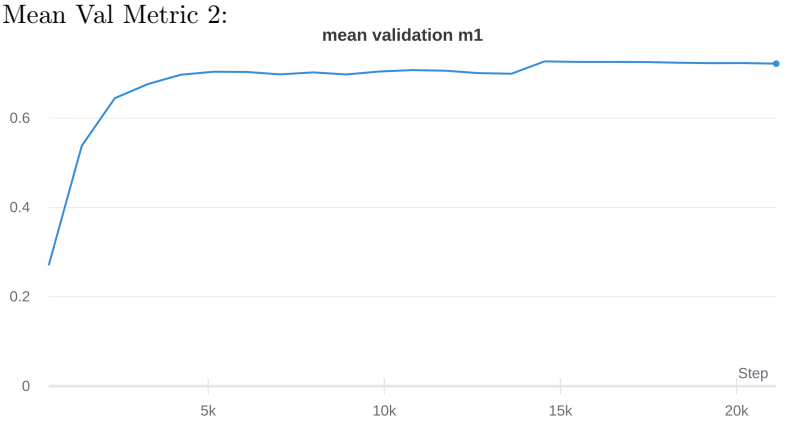
Training Metric 2:



Mean Val Metric 1:



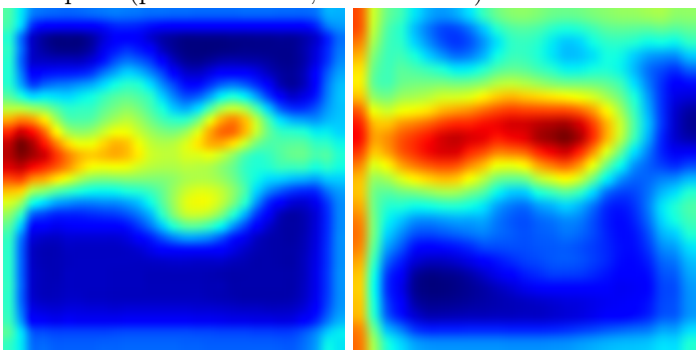
Mean Val Metric 2:



Final train metric1 (average): 0.991
Final train metric2 (average): 0.950
Final train loss (average): 0.0217
Final val metric1 (average): 0.7217
Final val metric2 (average): 0.620

Image:

1st Epoch (previous result, robust result):



Last Epoch (previous result, robust result):

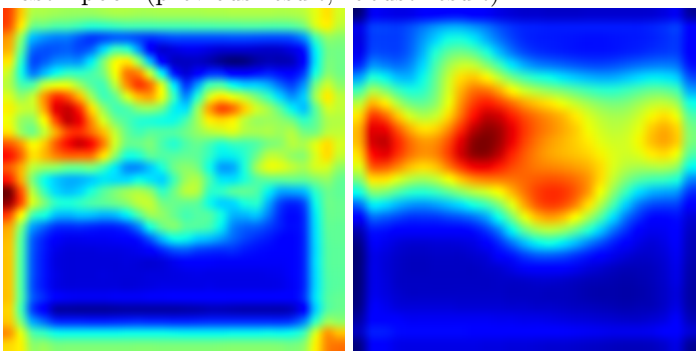
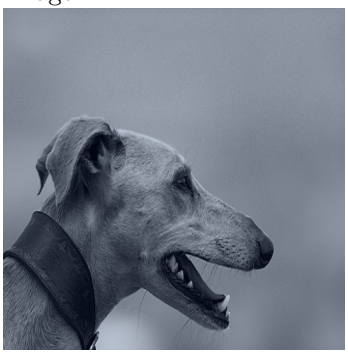
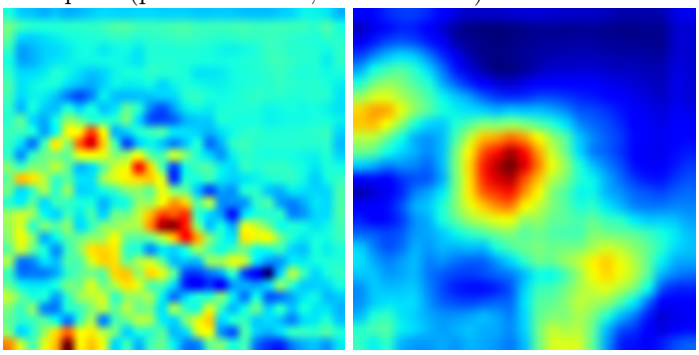


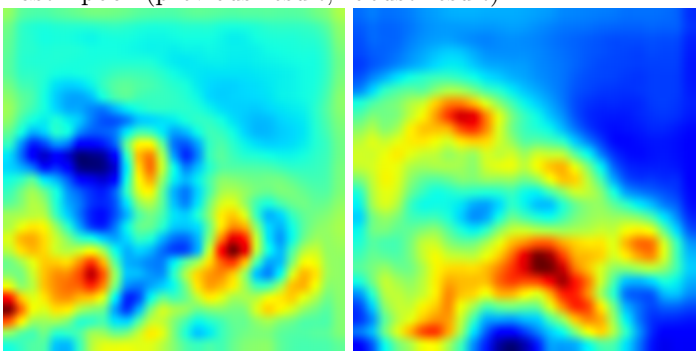
Image:



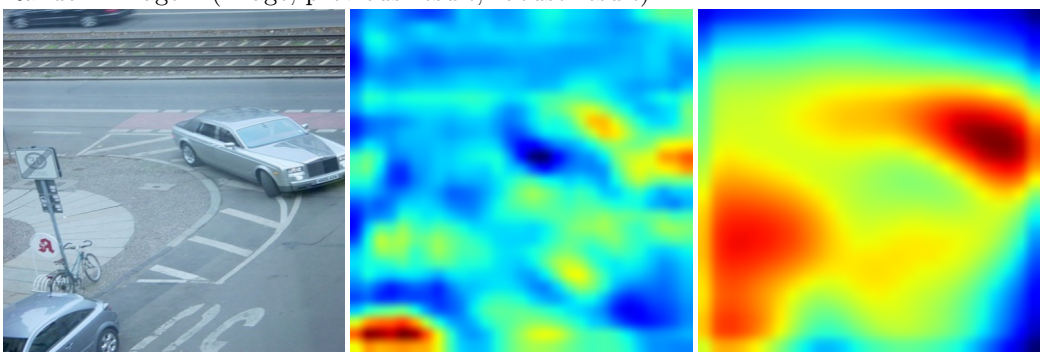
1st Epoch (previous result, robust result):



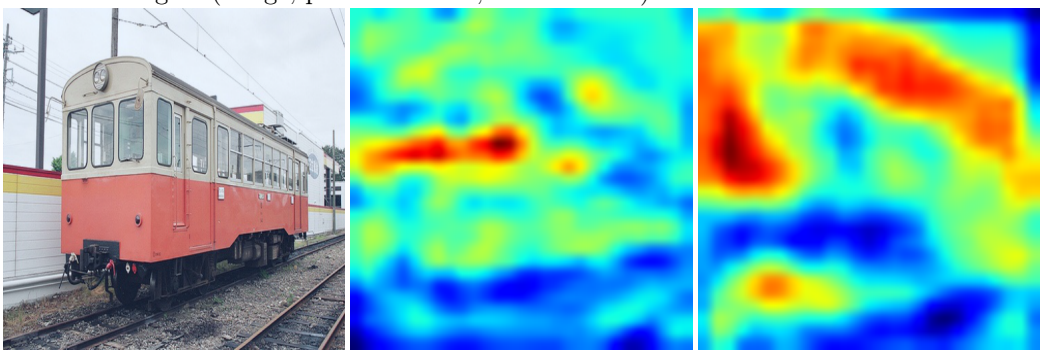
Last Epoch (previous result, robust result):



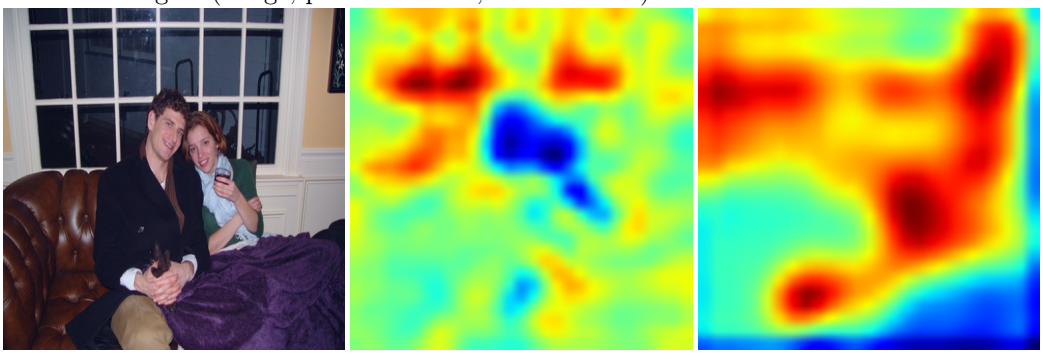
Random Image 1 (image, previous result, robust result):



Random Image 2 (image, previous result, robust result):



Random Image 3 (image, previous result, robust result):



2 Task 2

2.1

See the code for details

2.2

See the code for details

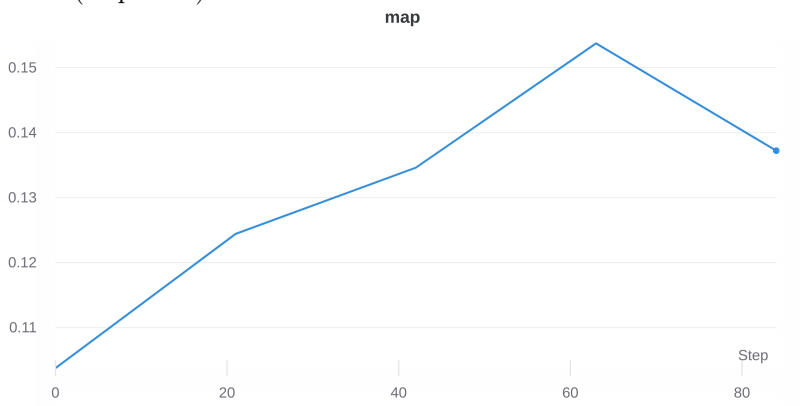
2.3

See the code for details

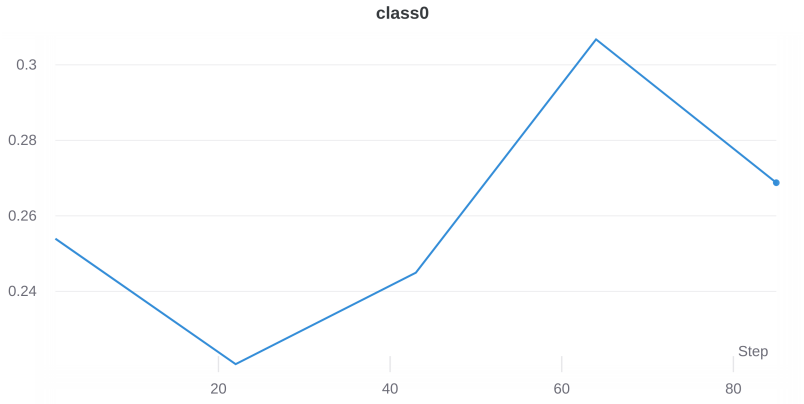
2.4

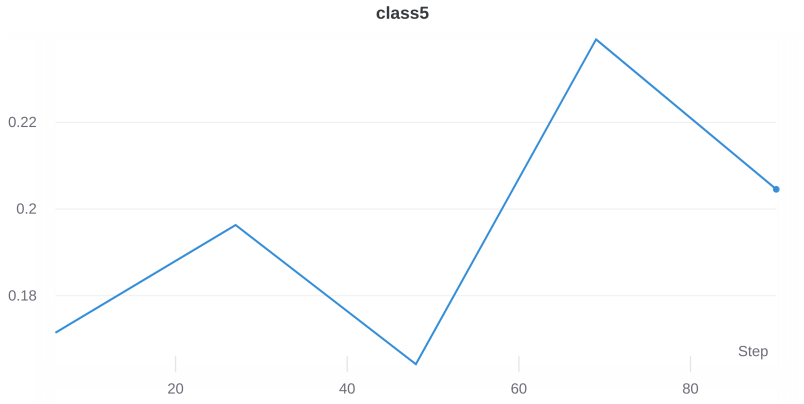
Train Loss (each step stands for 500 iteration/batches):

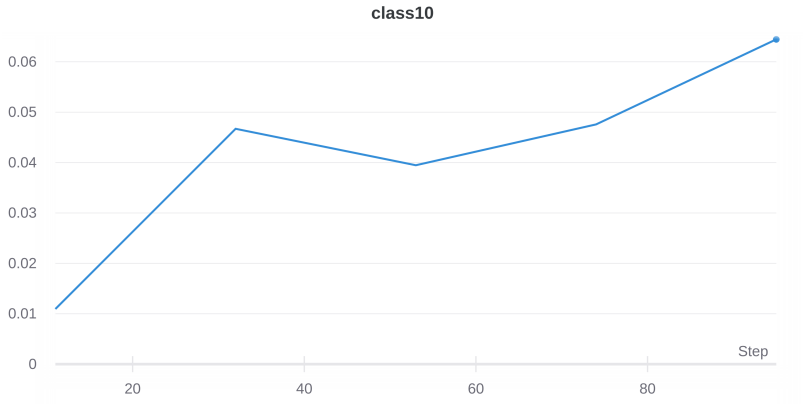
MAP (5 epoches):



I will include AP for three classes here:





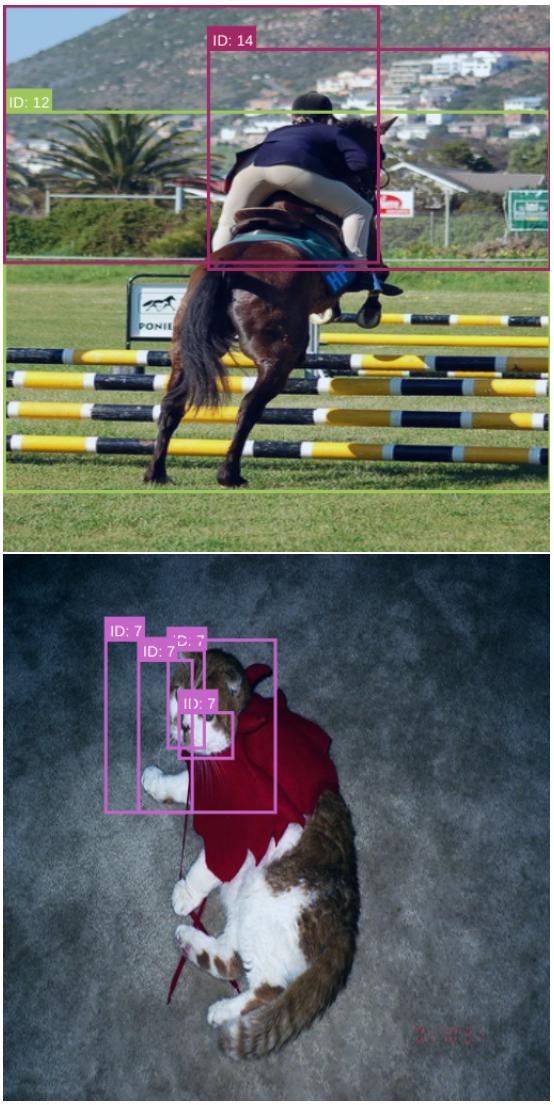


10 images after the first Epoch (Epoch 0):


ID: 18
FESTING ROAD

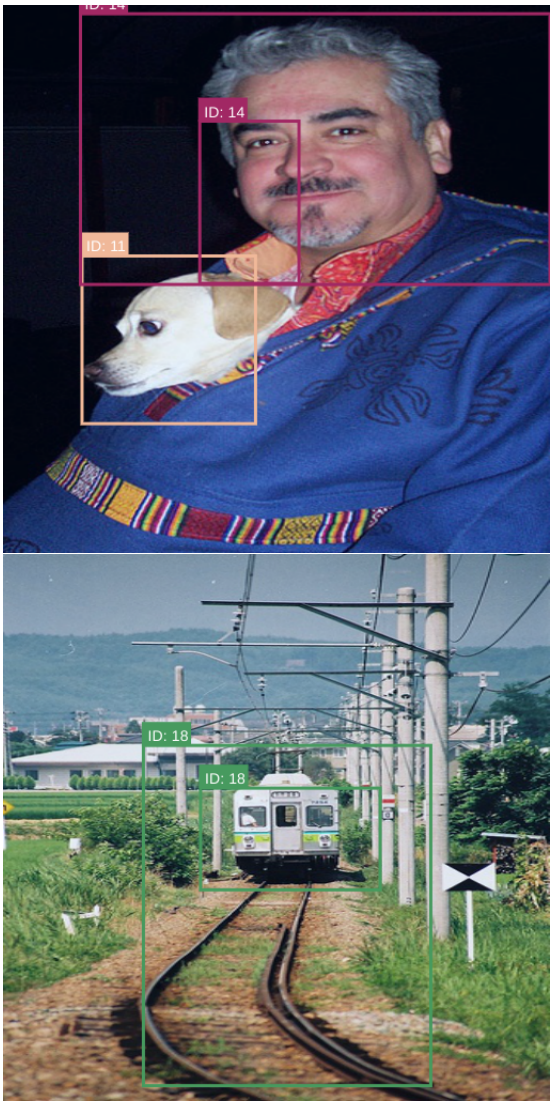

ID: 6
ID: 6


ID: 14
ID: 10
ID: 10
ID: 8

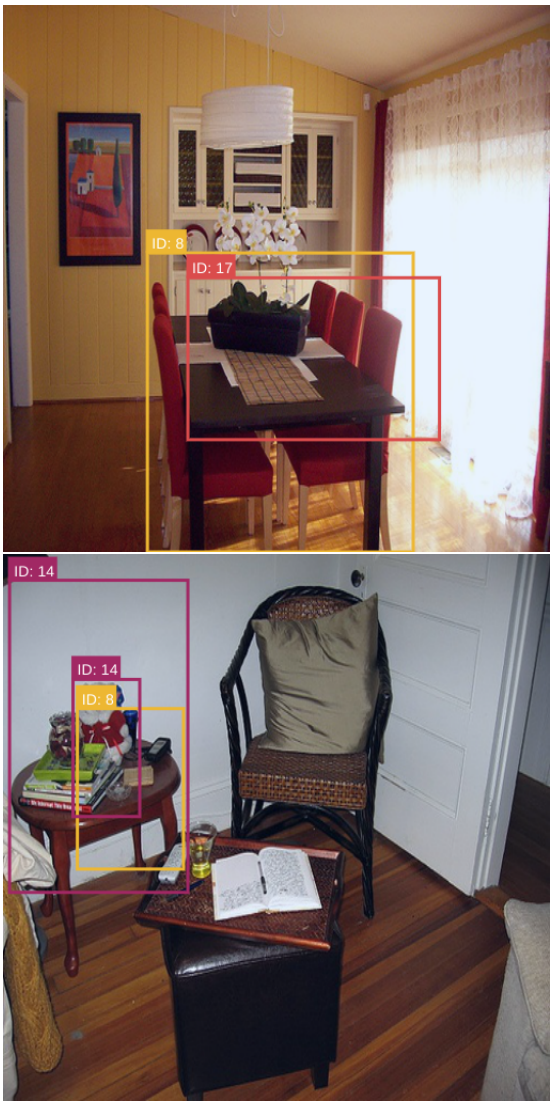

ID: 14
ID: 12
ID: 7
ID: 7
ID: 7


ID: 9
LOEWS
LOEWS
TWO
MEN
ID: 6

10 images after the last Epoch (Epoch 4):




ID: 18
FESTING ROAD
ID: 5
ID: 6
ID: 6


ID: 8
ID: 17
ID: 14
ID: 14
ID: 8


ID: 14
ID: 12
ID: 7

2.5

Final training loss: 0.9386
Final MAP: 0.1372