HOMEWORK 1

16824 VISUAL LEARNING AND RECOGNITION (FALL 2025)
https://piazza.com/cmu/fall2025/16824/home

RELEASED: Wed, 17 Sep 2025
DUE: Wed, 1 Oct, 2025
Instructor: Jun-Yan Zhu
TAs: Ananya Bal, Zhixuan Liu, Eungyeup Kim, Jay Karhade

START HERE: Instructions

- **Collaboration policy:** All are encouraged to work together BUT you must do your own work (code and write up). If you work with someone, please include their name in your write-up and cite any code that has been discussed. If we find highly identical write-ups or code or lack of proper accreditation of collaborators, we will take action according to strict university policies. See the Academic Integrity Section detailed in the initial lecture for more information.
- **Late Submission Policy:** There are a **total of 5** late days across all homework submissions. Submissions that use additional late days will incur a 10% penalty per late day.
- **Submitting your work:**
  - We will be using Gradescope (https://gradescope.com/) to submit the Problem Sets. Please use the provided template only. You do **not** need any additional packages and using them is **strongly discouraged**. Submissions must be written in LaTeX. All submissions not adhering to the template will not be graded and receive a zero.
  - **Deliverables:** Please submit all the `.py` files. Add all relevant plots and text answers in the boxes provided in this file. To include plots you can simply modify the already provided latex code. Submit the compiled `.pdf` report as well.

NOTE: Partial points will be given for implementing parts of the homework even if you don't get the mentioned accuracy as long as you include partial results in this pdf.

1 PASCAL multi-label classification (20 points)

In this question, we will try to recognize objects in natural images from the PASCAL VOC dataset using a simple CNN.

- **Setup:** Run the command `bash q1_q2_classification/download_data.sh` to download the train and test splits. The images will be downloaded in `data/VOCdevkit/VOC2007/JPEGImages` and the corresponding annotations are in `data/VOCdevkit/VOC2007/Annotations`. `voc_dataset.py` contains code for loading the data. Fill in the method `preload_anno` in to preload annotations from XML files. Inside `__getitem__` add random augmentations to the image before returning it using [TORCHVISION.TRANSFORMS]. There are lots of options and experimentation is encouraged. Implement a suitable loss function inside `trainer.py` (you can pick one from here). Also, define the correct dimension in `simple_cnn.py`.
- **Question:** The file `train_q1.py` launches the training. Please choose the correct hyperparameters in lines 19-28. You should get a mAP of around 0.22 within 5 epochs.
- **Deliverables:**
  - The code should log values to a Tensorboard. Compare the `Loss/Train` and `mAP` curves of the model with and without data augmentations in the boxes below. You should include the two curves in a single plot for each metric.

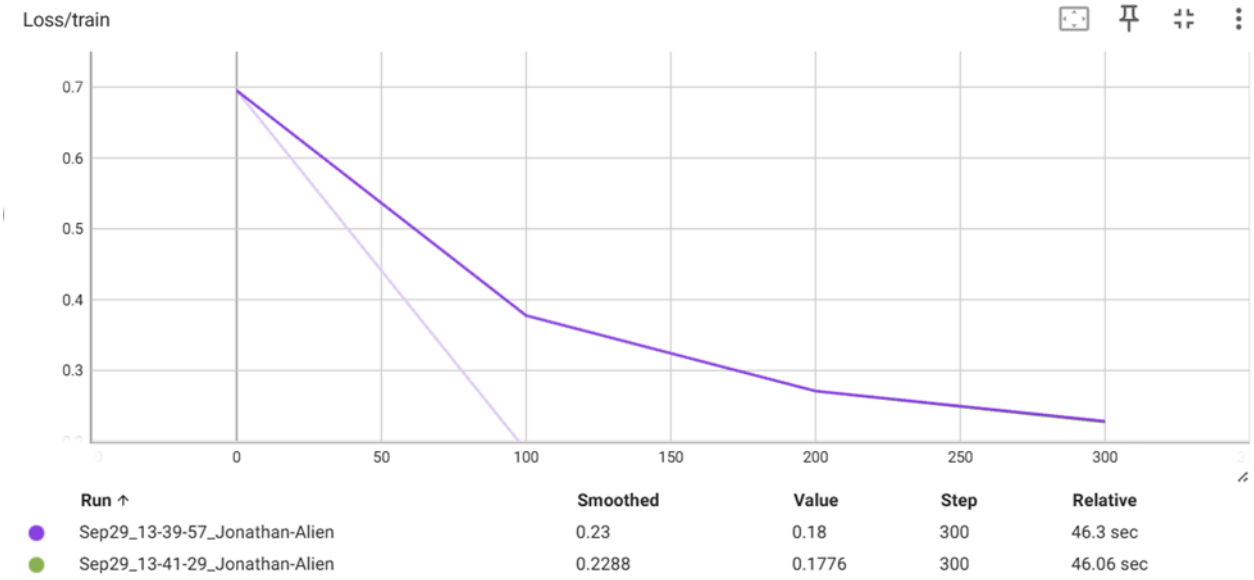

Figure 1.1: `Loss/Train` with and without data augmentations.

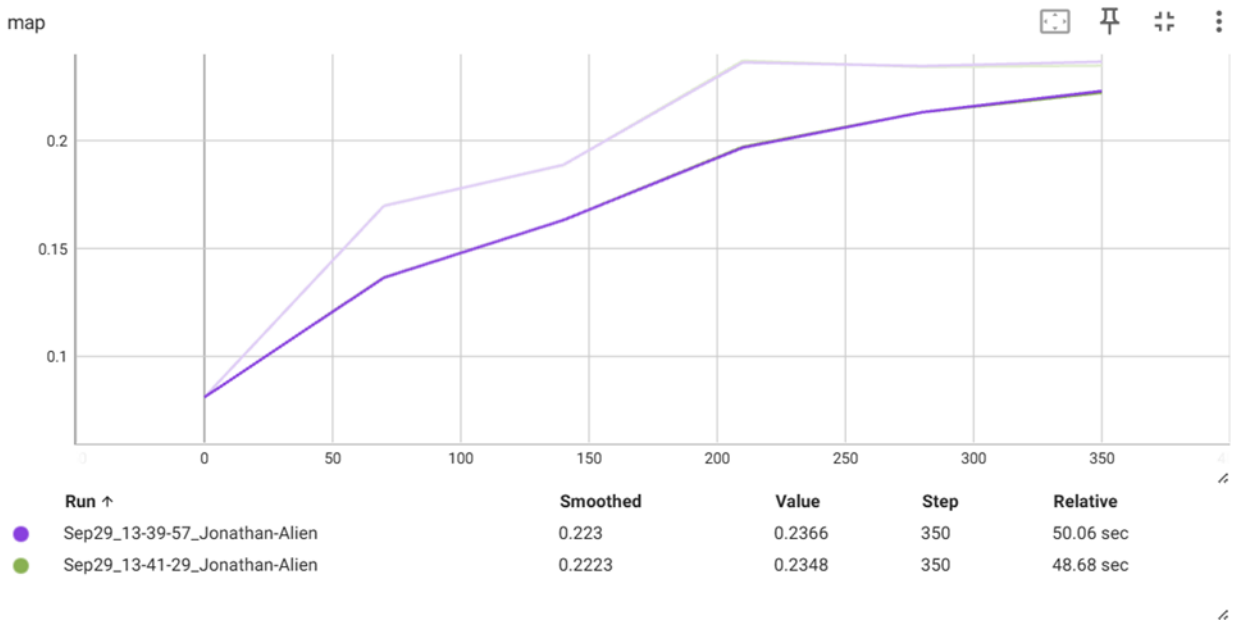

Figure 1.2: `mAP` with and without data augmentations.

– Report the `Loss/Train`, `mAP` and `learning_rate` curves of your best model logged to Tensorboard in the boxes below.

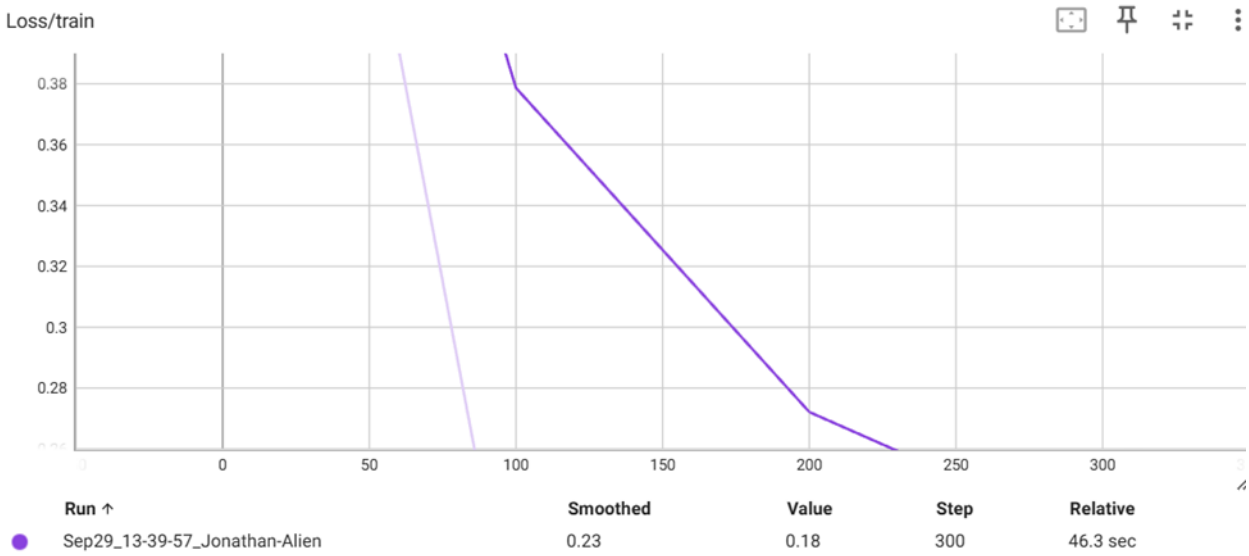

Figure 1.3: `Loss/Train` for simple CNN

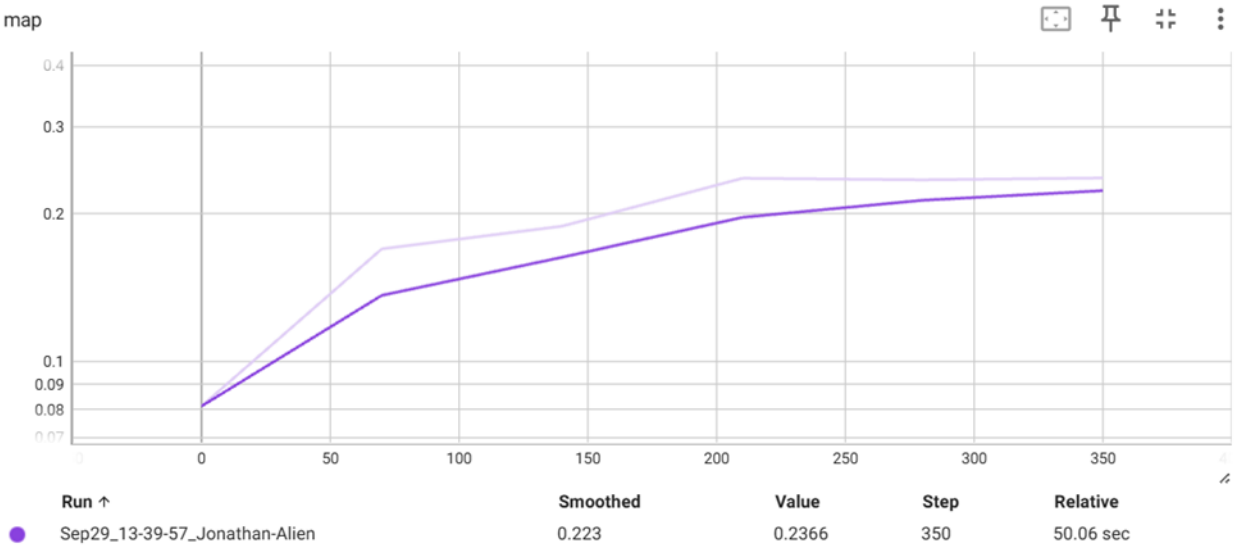

Figure 1.4: `mAP` for simple CNN

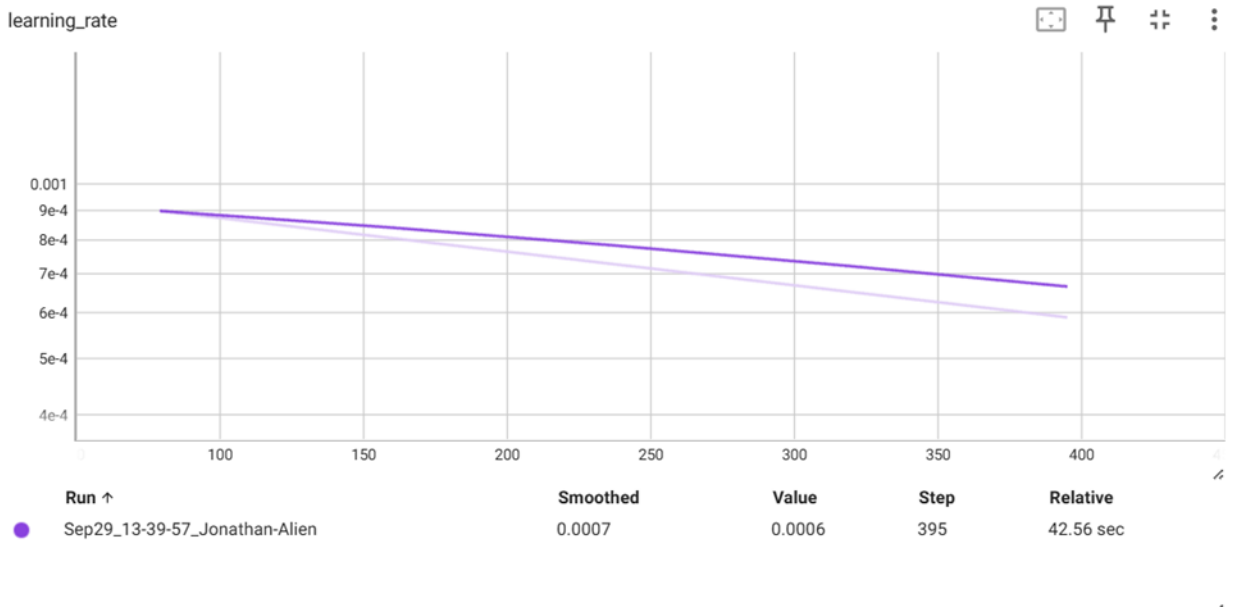

Figure 1.5: `learning_rate` for simple CNN

– Describe the hyperparameters you experimented with and the effects they had on the train and test metrics.

Solution:
Here are my final hyperparameters **lr: 1e-3, batch size: 64, step size: 1, gamma: 0.9**, I also tried **step size: 2, gamma: 0.9** and **gamma: 0.87**, and an **lr: 1e-2**. None of those iterations really had much effect except on the loss which achieved down to 0.13 on some

iterations with a **batch size:32**. But the largest change to mAP was increasing the **batch size** to 64 which increased mAP by 0.02 or 10. However, the loss did not really change much across the epochs. Likely too quick of a decrease for that batch size, but the mAP was actually 0.242 so I called it a win.

2 Even deeper! Resnet18 for PASCAL classification (20 pts)

Hopefully, we get much better accuracy with the deeper models! Since 2012, much deeper architectures have been proposed. ResNet is one of the popular ones.

- **Setup:** Write a network module for the Resnet-18 architecture (refer to the original paper) inside `train_q2.py`. You can use Resnet-18 available in `torchvision.models` for this section. Use ImageNet pre-trained weights for all layers except the last one.
- **Question:** The file `train_q2.py` launches the training. Tune hyperparameters to get mAP around 0.8 in 50 epochs.
- **Deliverables:** Paste plots for the following in the box below.
  - Include curves of training loss, test MAP, learning rate, and histogram of gradients from Tensorboard for `layer1.1.conv1.weight` and `layer4.0.bn2.bias`.

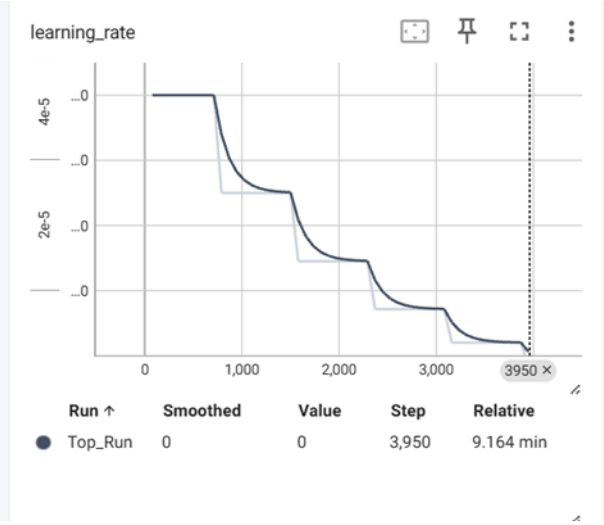

Figure 2.1: `learning_rate` for ResNet

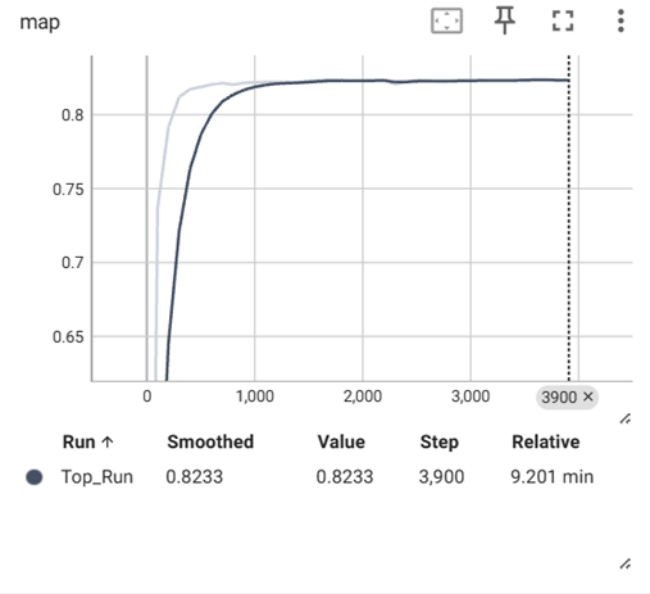

Figure 2.2: `mAP` for ResNet

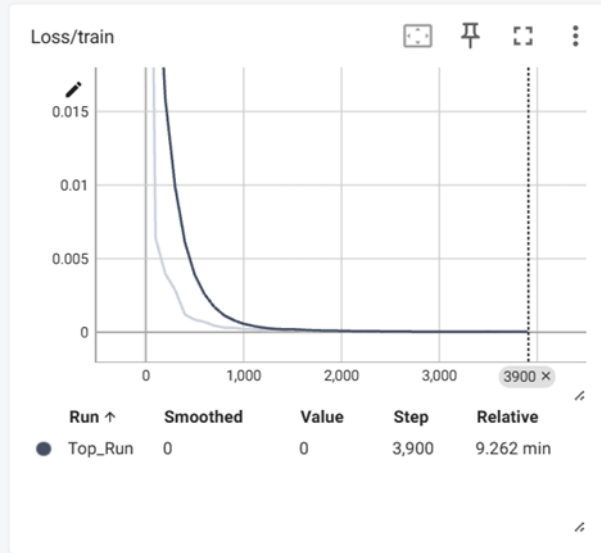

Figure 2.3: `Loss/Train` for ResNet

– How does the test mAP and training loss change over time? Why do you think this is happening?

Solution:
So this was less of a problem when I switched to focal loss because BCE with label smoothing couldn't get me there no matter what I did. And during that time, the mAP would improve rapidly and remain high for the first 10-20 epochs, and degrade or even collapse as the model improved according to the loss. This still happened ever so slightly with focal loss, but not nearly as bad as you can see as the two stayed relatively converged throughout

training. So I determined that the regularization of the training images was the problem after extensive hyperparameter and data augmentation tuning, and changed my loss.

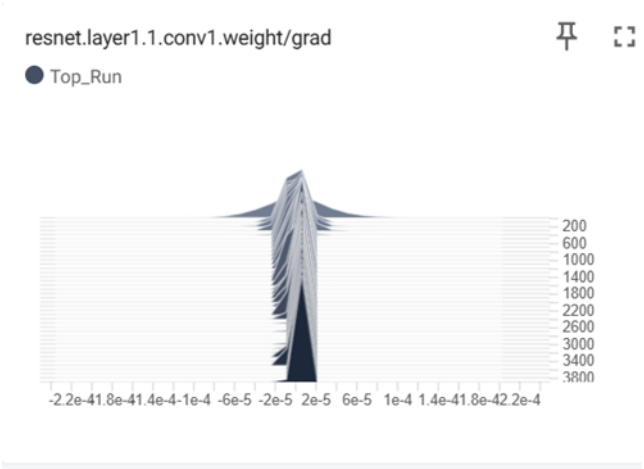

Figure 2.4: Histogram of `Conv1` layer

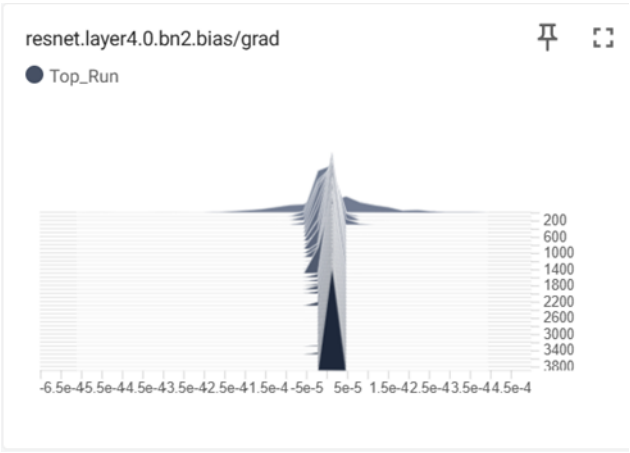

Figure 2.5: Histogram of `BN4` layer

– Compare the two histogram plots. How do they change over time? Why do you think this is happening?

Solution:
As the BN4 layer is much closer to the actual classification loss you see that the deeper layers converge quicker while the earlier layers have to receive the backprop signal through many layers which is a strength of ResNet but still not perfect as we can see so the earlier layers take a bit more time to learn.

– We can also visualize how the feature representations specialize for different classes. Take 1000 random images from the test set of PASCAL, and extract ImageNet (finetuned) features from

those images. Compute a 2D t-SNE (use sklearn) projection of the features, and plot them with each feature color-coded by the GT class of the corresponding image. If multiple objects are active in that image, compute the color as the "mean" color of the different classes active in that image. Add a legend explaining the mapping from color to object class.

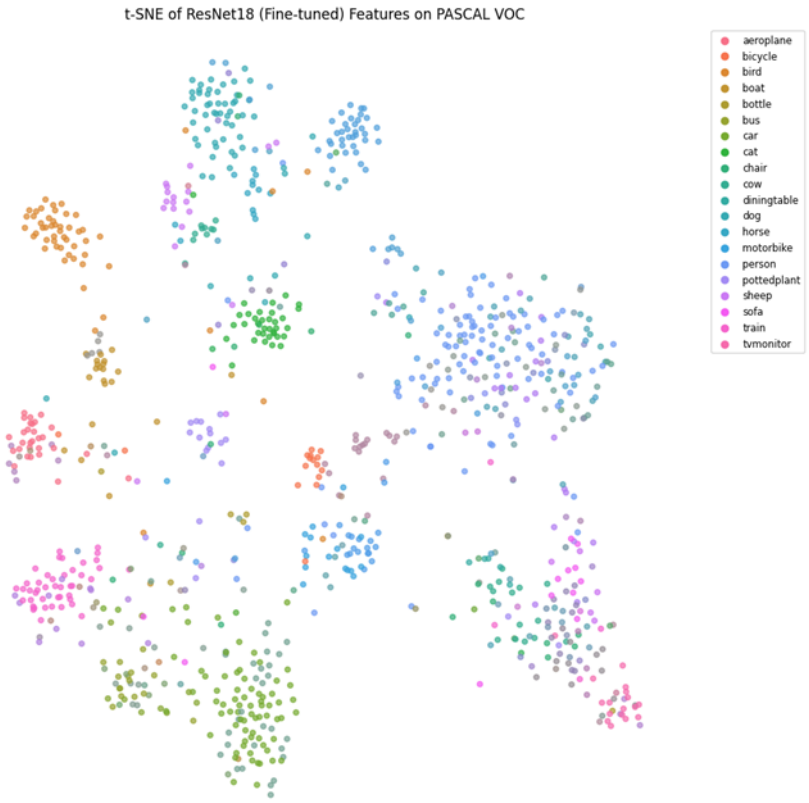

Figure 2.6: t-SNE

– Briefly describe what you observe in the t-SNE plot. Does this match your expectations?

Solution:
This is pretty much what I expected with roughly 82 percent mAP, mostly unicolored clusters with some overlap of classes. The only confusing part would be some relatively substantial overlap in some classes, but I believe that's probably a 2D problem whereas making this 3D would probably eliminate a good bit of the overlap we see in the 2D representation.

3 Supervised Object Detection: FCOS (60 points)

In this problem, we'll be implementing supervised Fully Convolutional One-stage Object Detection (FCOS).

- **Setup**. This question will require you to implement several functions in `detection_utils.py` and `one_stage_detector.py` in the `detection` folder. Instructions for what code you need to write are in the `README` in the `detection` folder of the assignment.

  We have also provided a testing suite in `test_object_detection.py`. First, run the test suite and ensure that all the tests are either skipped or passed. Make sure that the Tensorboard visualization works by running 'python3 train.py –visualize_gt'; this should upload some examples of the training data with bounding boxes to Tensorboard. Make sure everything is set up properly before moving on.

  Then, run the following to install the `mAP` computation software we will be using.

```
cd <path/to/hw/>/detection
pip install wget
rm -rf mAP
git clone https://github.com/Cartucho/mAP.git
rm -rf mAP/input/*
```

  Next, open `detection/one_stage_detector.py` and `detection/detection_utils.py`. At the top of the files are detailed instructions for where and what code you need to write. Follow the `README` and all the instructions for implementation.

- **Deliverables.**
  - It's always a good idea to check if your model can overfit on a small subset of the data, otherwise there may be some major bugs in the code. Train your FCOS model on a small subset of the training data and make sure the model can overfit. Include the loss curve from over-fitting below.
  - **I wasn't sure which loss curve you wanted so I went to the detection helper script and added a total loss function and am adding them all here:**

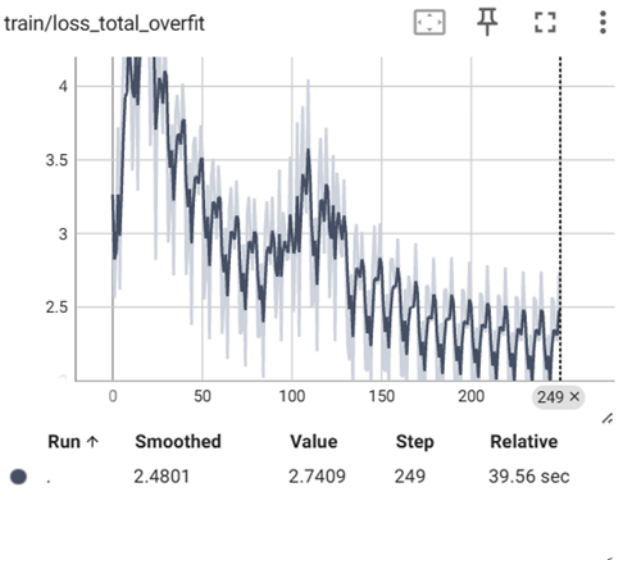

Figure 3.1: Total Loss Overfitting Training Curve

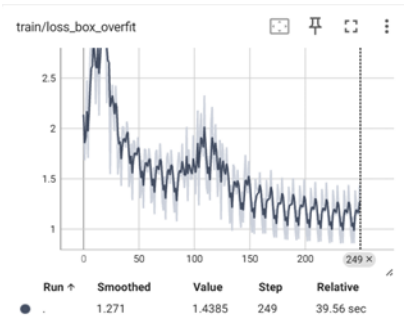

Figure 3.2: Box Loss Overfitting Training Curve

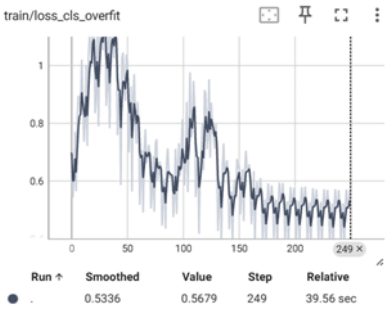

Figure 3.3: CLS Loss Overfitting Training Curve

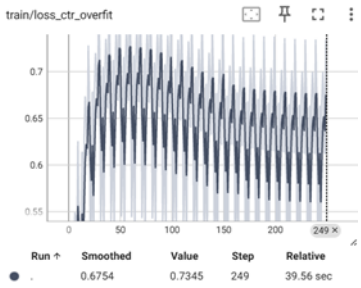

Figure 3.4: Loss Ctr Overfitting Training Curve

– Next, train FCOS on the full training set and include the loss curve below.

– **Same consideration as above**

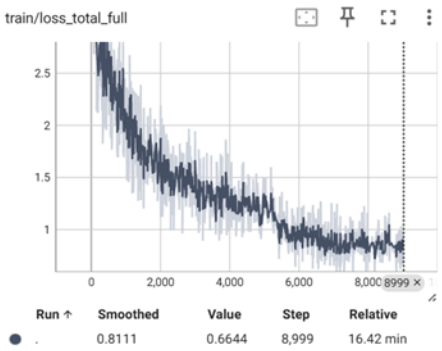

Figure 3.5: Total Loss Full Training Curve

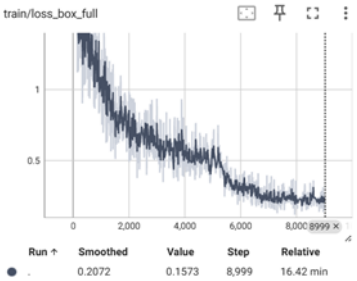

Figure 3.6: Box Loss Full Training Curve

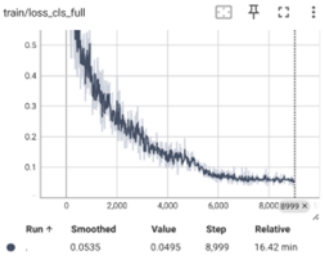

Figure 3.7: CLS Loss Full Training Curve

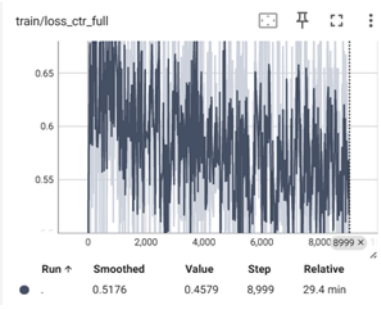

Figure 3.8: Loss Ctr Full Training Curve

– Include the plot of the model's class-wise and average mAP. If everything is correct, your implementation should reach a mAP of at least 20.

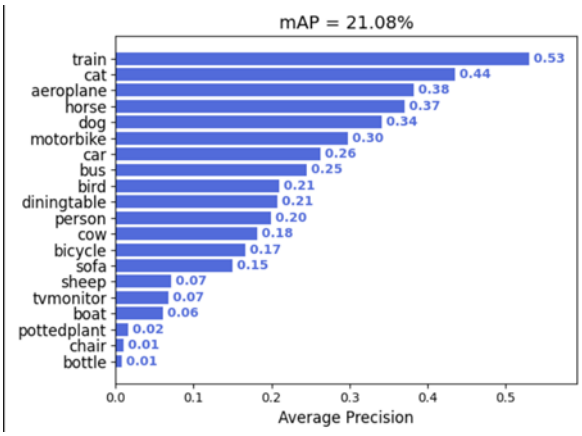

Figure 3.9: mAP plots

– Paste a screenshot of the Tensorboard visualizations of your model inference results from running inference with the `--test_inference` flag on.

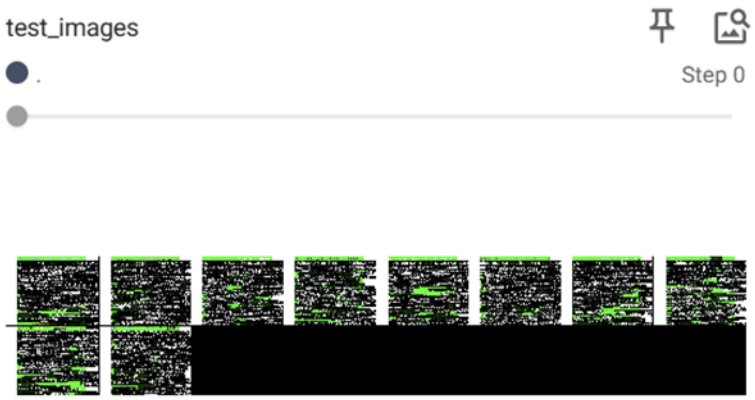

Figure 3.10: Tensorboard Inference Visualization

– What can you conclude from the above visualizations? When does the model succeed or fail? How can you improve the results for the failure cases?

Solution:
We can generally infer that the model does well on large, clearly distinct objects like trains, planes, singular animals, etc., but it does poorly in partially occluded or crowded objects like sheep and boats, (though admittedly I believe there are fewer of these classes in this dataset). So I would probably move to focal loss rather than the centerness which I believe

is a weakness here, adapt to localization/feature alignment error in the model, handle class-balanced sampling or additional data augmentations on those limited classes with stronger data augmentations overall.

Collaboration Survey Please answer the following:

1. Did you receive any help whatsoever from anyone in solving this assignment?

   ◯ Yes

   ◯ No

   - If you answered 'Yes', give full details:
   - (e.g. "Jane Doe explained to me what is asked in Question 3.4")

2. Did you give any help whatsoever to anyone in solving this assignment?

   ◯ Yes

   ◯ No

   - If you answered 'Yes', give full details:
   - (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")

3. Note that copying code or writeup even from a collaborator or anywhere on the internet violates the Academic Integrity Code of Conduct.