HOMEWORK 1

16824 VISUAL LEARNING AND RECOGNITION (SPRING 2024)
https://piazza.com/class/lr8dzk3rn9n4d8

RELEASED: Mon, 5 Feb 2024
DUE: Wed, 21 Feb, 2024
Instructor: Deepak Pathak
TAs: Mihir Prabhudesai, Kenny Shaw, Shagun Uppal, Himangi Mittal, Sayan Mondal

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 download_dataset.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 13-19. 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.

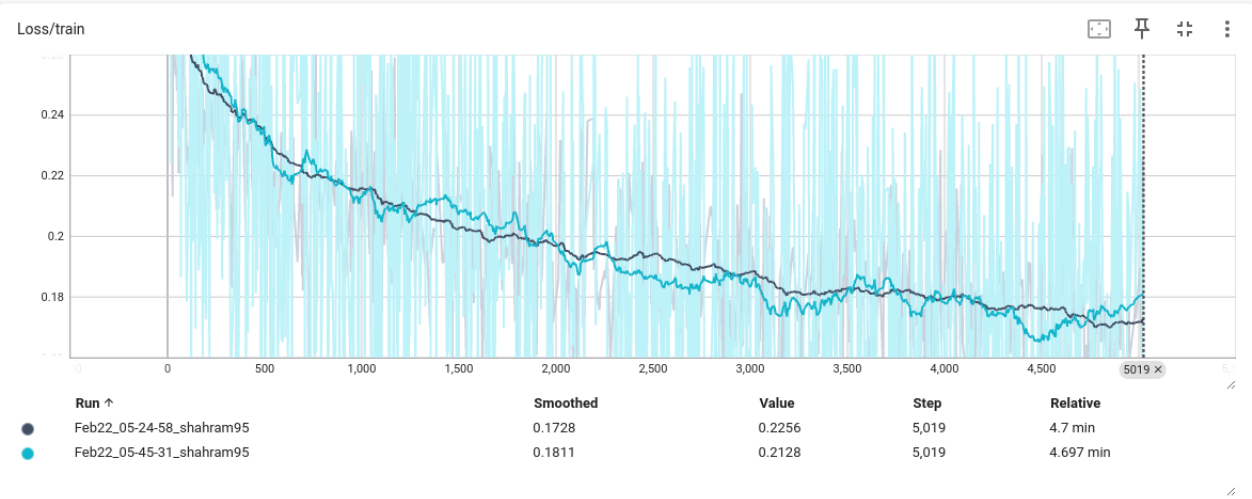

Figure 1.1: `Loss/Train` with (blue) and without (gray) data augmentations.

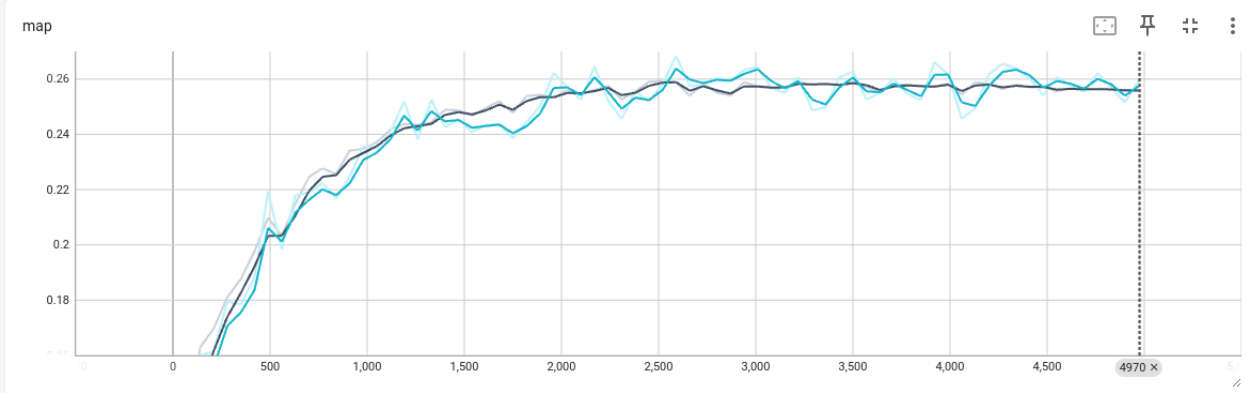

Figure 1.2: `mAP` with (blue) and without (gray) data augmentations.

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

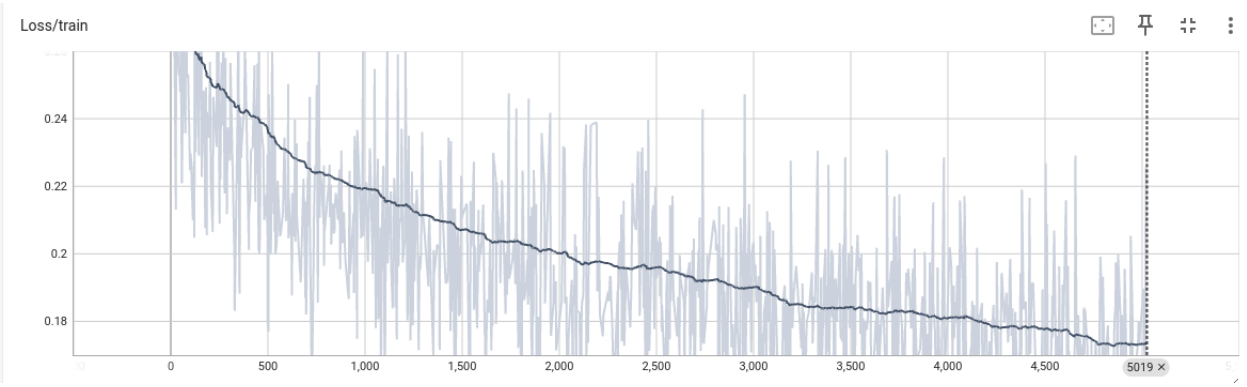

Figure 1.3: `Loss/Train` for simple CNN

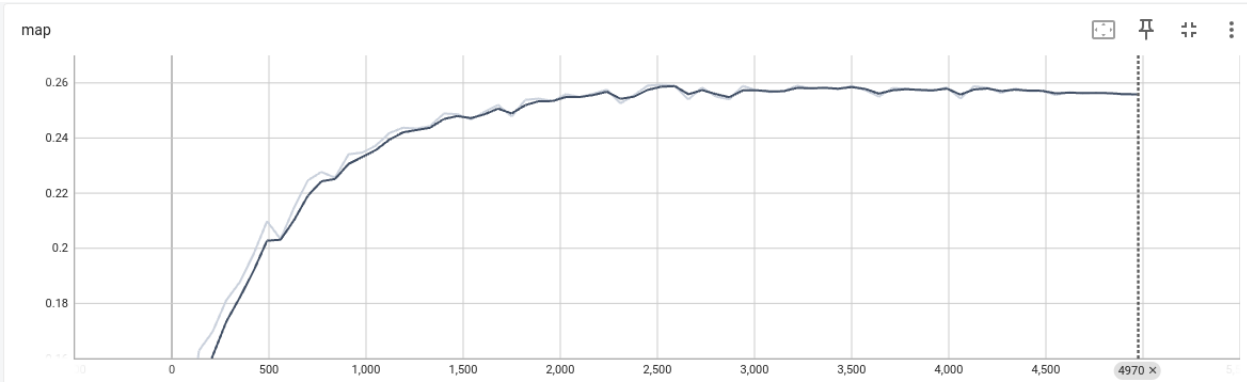

Figure 1.4: `mAP` for simple CNN

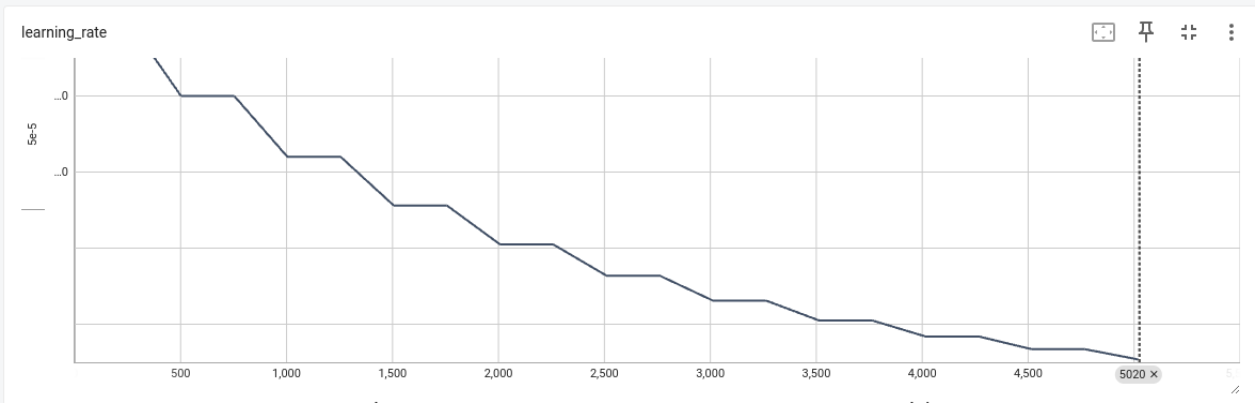

Figure 1.5: `learning_rate` for simple CNN

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

Solution: Table: Hyperparameter tuning and its effects on CNN performance.

Epochs	learning rate	Batch size	Step Size	Gamma	train loss	test map
20	0.0001	32	2	0.8	0.2072	0.255
1	0.0001	32	2	0.8	0.2932	0.164
5	0.0001	32	2	0.8	0.213	0.227
20	0.001	32	2	0.8	0.0179	0.243
20	0.01	32	2	0.8	1.2665	0.077
20	0.0001	16	2	0.8	0.3004	0.257
20	0.0001	64	2	0.8	0.1987	0.239
20	0.0001	32	10	0.8	0.1371	0.247
20	0.0001	32	6	0.8	0.1648	0.249
20	0.0001	32	2	0.1	0.222	0.196
20	0.0001	32	2	0.4	0.2132	0.217

Analysis:

In the ablation study presented above, the impact of varying hyperparameters on the performance of the Convolutional Neural Networks (CNNs) was explored, focusing on learning rate, batch size, step size, and gamma. These parameters were adjusted in a series of experiments to observe their effects on training loss and test mean average precision (mAP).

* *Learning Rate:* Increasing the learning rate from 0.0001 to 0.001 decreased training loss and slightly reduced test mAP, indicating faster convergence but potential for overfitting. Further increase to 0.01 significantly worsened both metrics, suggesting instability and divergence at excessively high rates.
* *Batch Size:* Reducing batch size to 16 improved test mAP, suggesting smaller batches enhance generalization by introducing beneficial noise. Increasing batch size to 64 slightly reduced training loss but also decreased test mAP, indicating larger batches provide more stable gradients but may reduce the model's ability to generalize.

* *Step Size and Gamma:* Increasing step size led to improvements in both training loss and test mAP, implying less frequent learning rate adjustments could be beneficial. Adjusting gamma showed that overly aggressive reduction (to 0.1) worsened performance, whereas a moderate value (0.4) balanced learning efficiency and stability, although not outperforming the baseline setting of 0.8.

Unexpected Observations:

* The significant decrease in test mAP with an increase in learning rate to 0.01 was more drastic than expected. This suggests not just a simple case of overfitting, but potentially the onset of training instability and divergence, underscoring the sensitivity of CNNs to higher learning rates.
* The improvement in test mAP with a batch size of 16 diverged from the common assumption that larger batch sizes always yield better generalization. This outcome indicates a complex interplay between batch size and noise in gradient estimates, where smaller batches may introduce a form of regularization through noise, enhancing the model's ability to generalize.
* The positive impact of increasing step size to 10 and 6 on both training loss and test mAP contradicts the conventional wisdom that more frequent learning rate adjustments lead to better performance. This finding could suggest that less frequent updates allow for more consistent learning phases before the model adapts, possibly improving exploration of the loss landscape.

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`.

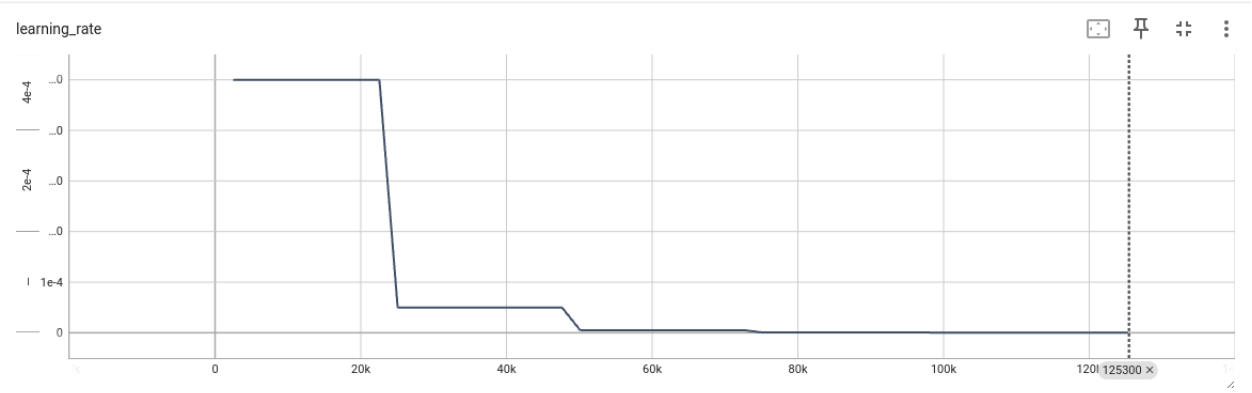

Figure 2.1: `learning_rate` for ResNet

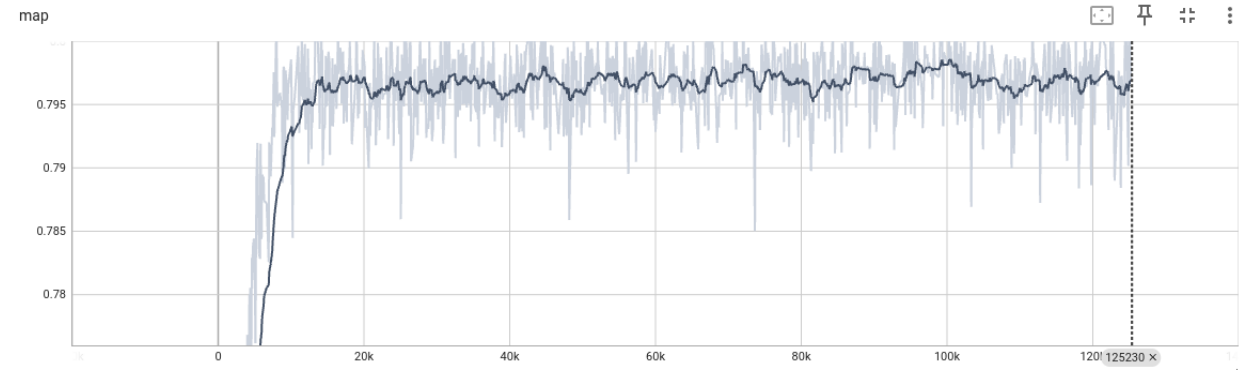

Figure 2.2: `mAP` for ResNet

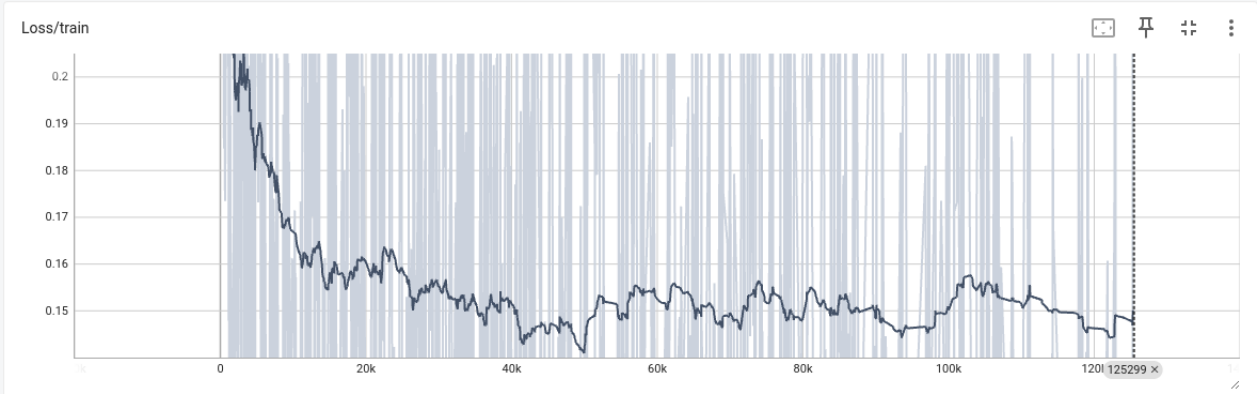

Figure 2.3: `Loss/Train` for ResNet

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

Solution: The following trends are observed during the training process:

Training Loss: The training loss exhibits a clear downward trend, indicating that the model is consistently learning from the training data over time. The loss decreases sharply at the beginning, reflecting rapid learning of the most significant features. As training progresses, the loss reduction slows as the model begins to converge. Notably, there are fluctuations in the loss, likely due to the variability of the batches.

Test mAP: The test mAP, which assesses the model's precision in object detection across various classes, gradually increases. This upward trend suggests an improvement in the model's ability to generalize to unseen data. The mAP did plateau fairly quickly for the set of hyperparameters, implying potential for further enhancement in terms of data quality with continued training. Fluctuations in mAP are present but less pronounced than in training loss.

Reasoning for Observed Trends: The decrease in training loss is attributable to the model's weight adjustments aimed at minimizing prediction errors on the training set. This process is driven by error backpropagation and optimization algorithms. The increase in test mAP indicates an increased capability of the model to detect and classify objects correctly on the test set, a sign of effective learning and generalization.

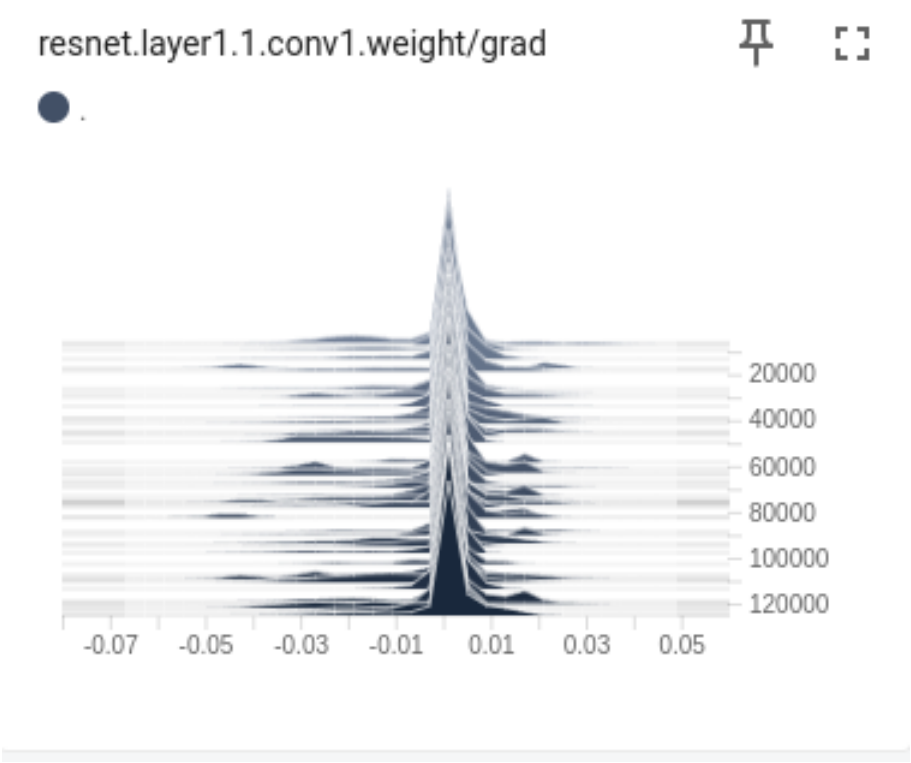

Figure 2.4: Histogram of `Conv1` layer

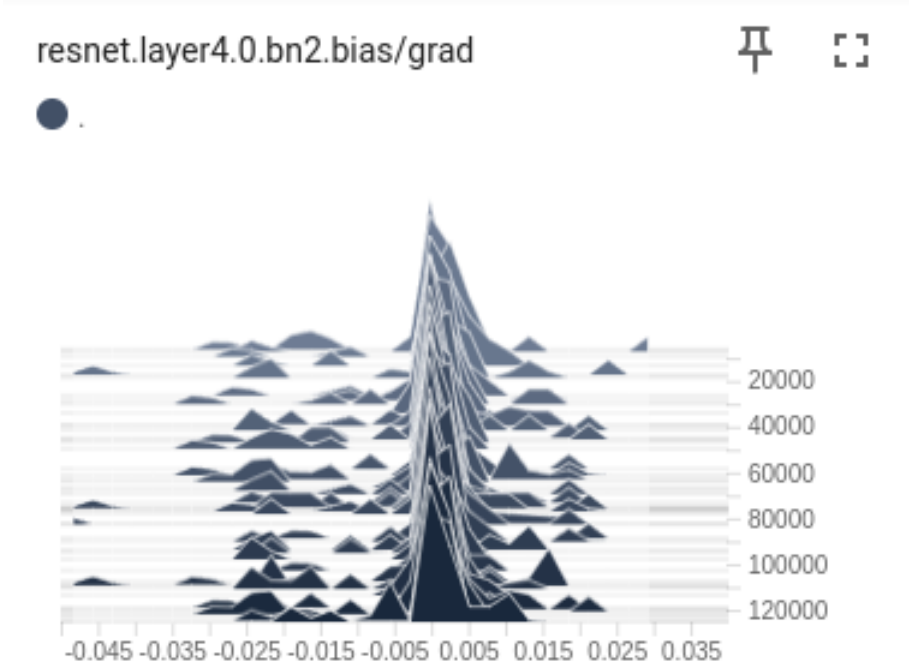

Figure 2.5: Histogram of `BN4` layer

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

Solution: Both histograms represent the distribution of gradient values for different parameters in a ResNet18 model over time.

For `resnet.layer1.1.conv1.weight/grad`, the histogram shows a multi-peaked distribution around zero, suggesting that most gradients for the weights in this convolutional layer are small, with a symmetric distribution on both sides of zero. The multiple peaks may indicate that different subsets of weights are being updated at varying rates, reflecting the layer's learning of diverse features at different scales.

In contrast, the histogram for `resnet.layer4.0.bn2.bias/grad` displays a sharper peak around zero. This indicates a more uniform update across the biases in the batch normalization layer, which typically serves to fine-tune the learned features more evenly across the network's depth.

As the histograms' distributions become narrower, it signifies the model's convergence towards an optimal set of weights. This is expected as initially larger gradients correct the model's weights substantially, and over time, as the model approaches an optimal state, these adjustments become smaller and more refined.

The observed changes in the gradient distributions can be attributed to several factors including the learning rate decay, the model's convergence behavior, and the distinct roles of different layers within the network. Convolutional layers such as `layer1.1.conv1` typically learn basic features and might exhibit more significant gradient variability, while deeper layers like `layer4.0.bn2` refine these features and thus might display a more concentrated gradient distribution, which is consistent with the observation in the above representations.

– 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.

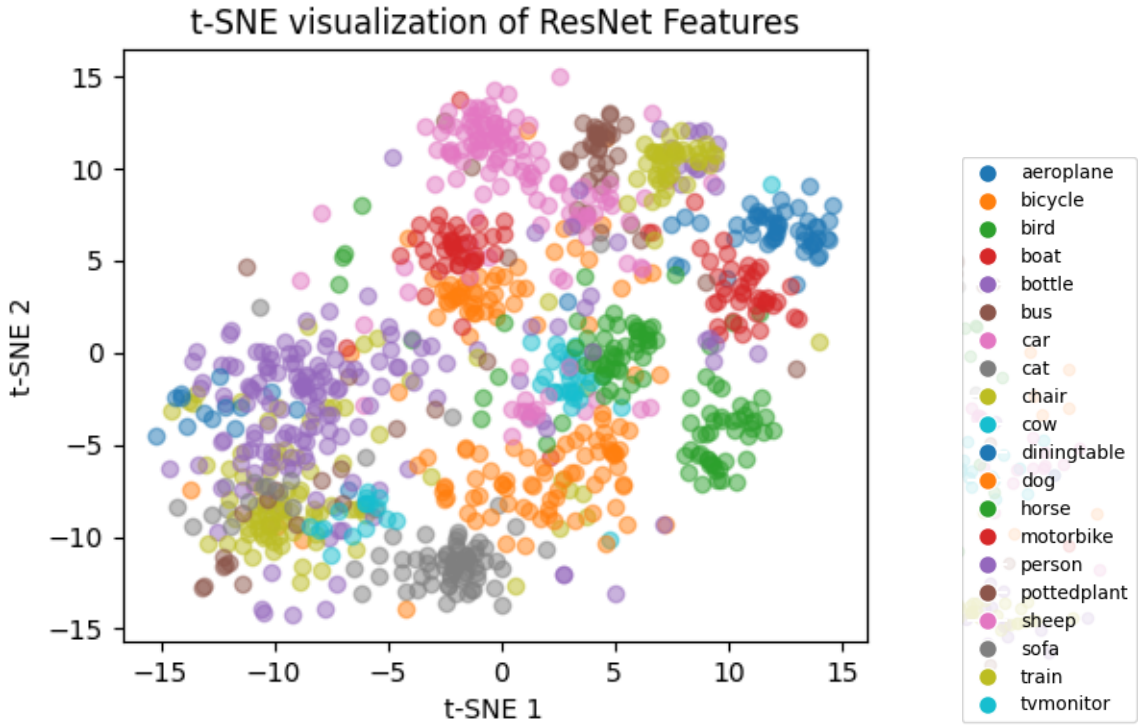

Figure 2.6: t-SNE

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

Solution: The t-SNE visualization presents distinct clusters corresponding to different object categories from the VOC test dataset, as learned by the ResNet model from the training dataset. Each cluster is color-coded, matching the legend provided as required.

The clusters shown above are compact, showing that instances within the same category have similar feature representations. However, some clusters, like those for ”cat” and ”dog,” show overlap, likely due to similarities in the features of these categories, such as shape and size, which is expected. A few outliers are present, which is typical in complex datasets with high intra-class variability.

Overall, the separation between different classes and the tightness of the clusters within the same class indicate that the ResNet model has learned discriminative features. This matches expectations for a model trained on a diverse set of categories, where clear separability is indicative of a well-performing classifier. The presence of overlap and outliers aligns with the anticipated challenges in distinguishing similar categories and dealing with diverse data representations.

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_one_stage_detector.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`. At the top of the file are detailed instructions for where and what code you need to write. Follow 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.

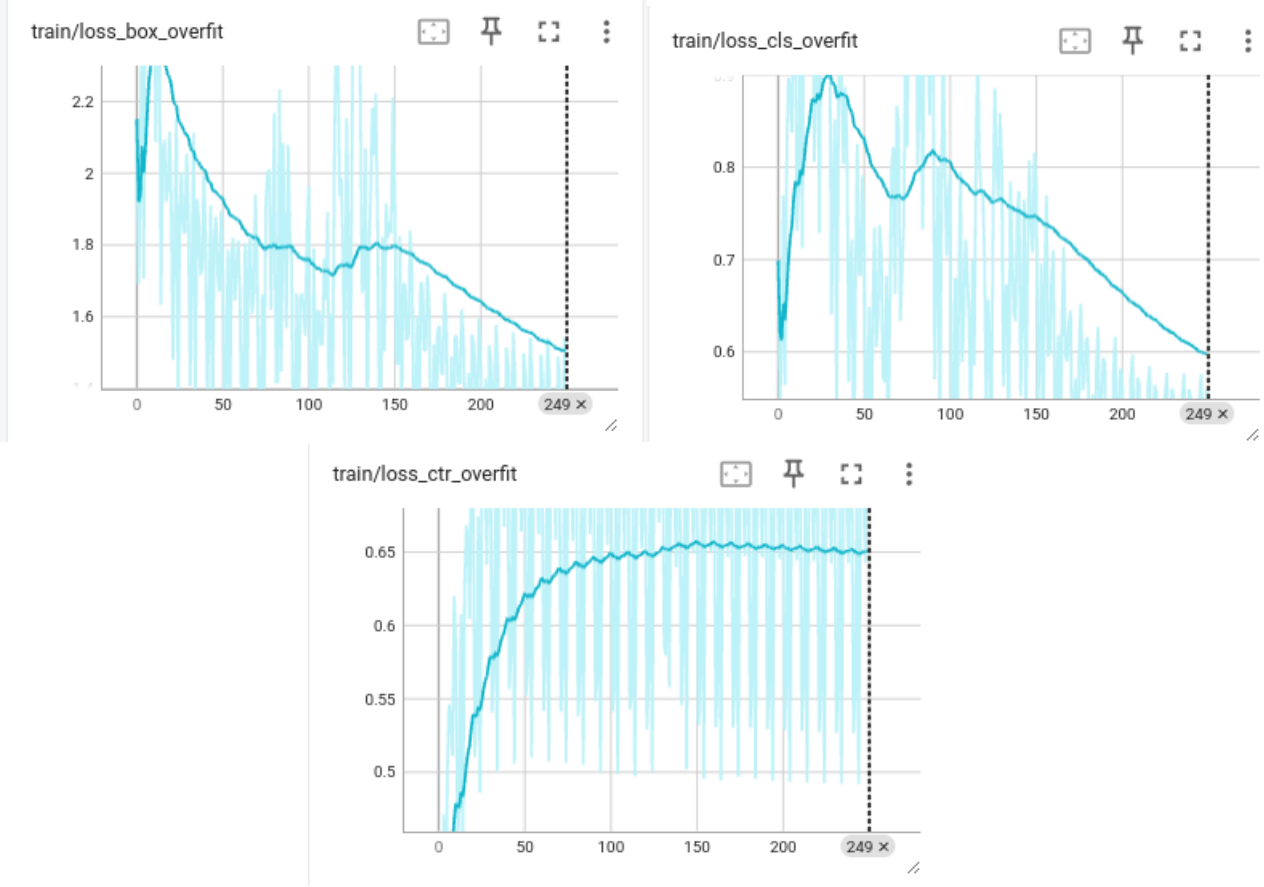

Figure 3.1: Overfitting Training Curve

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

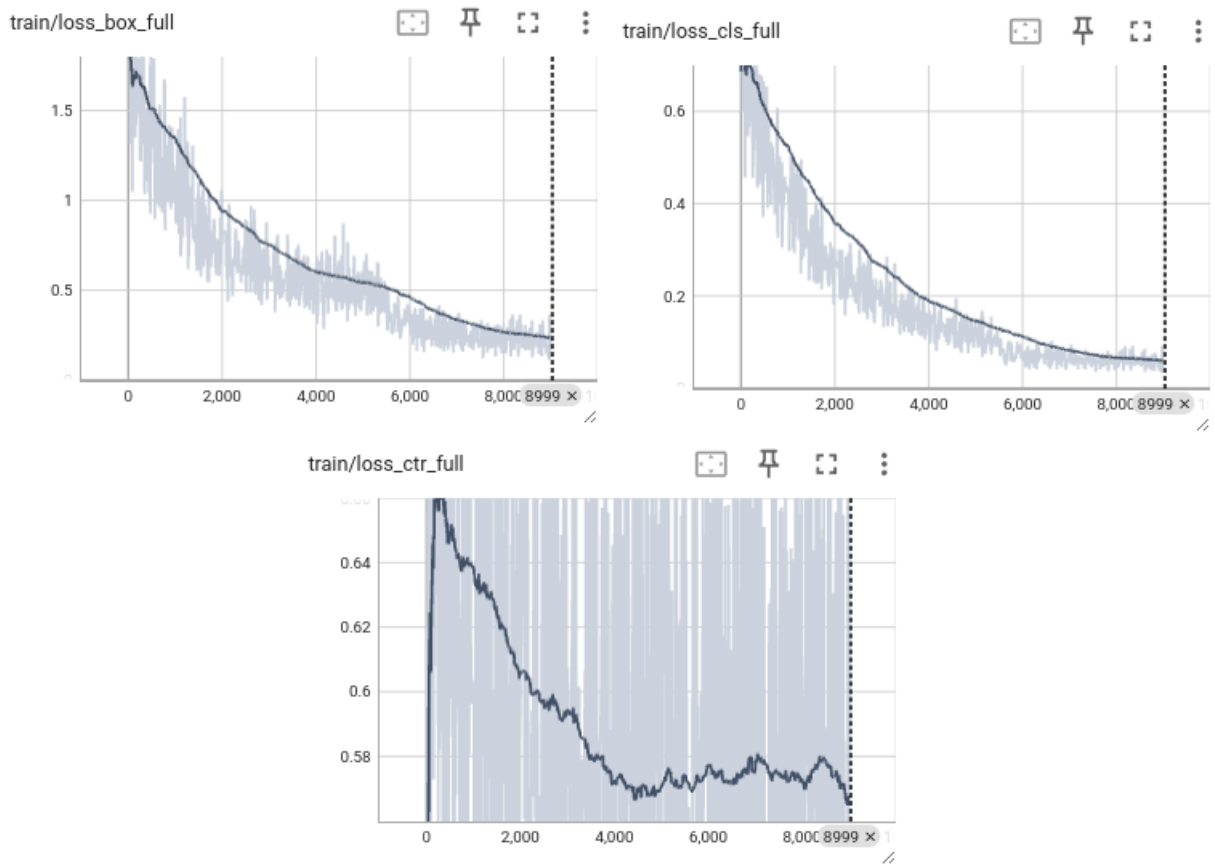

Figure 3.2: Full Training Curve

– 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.

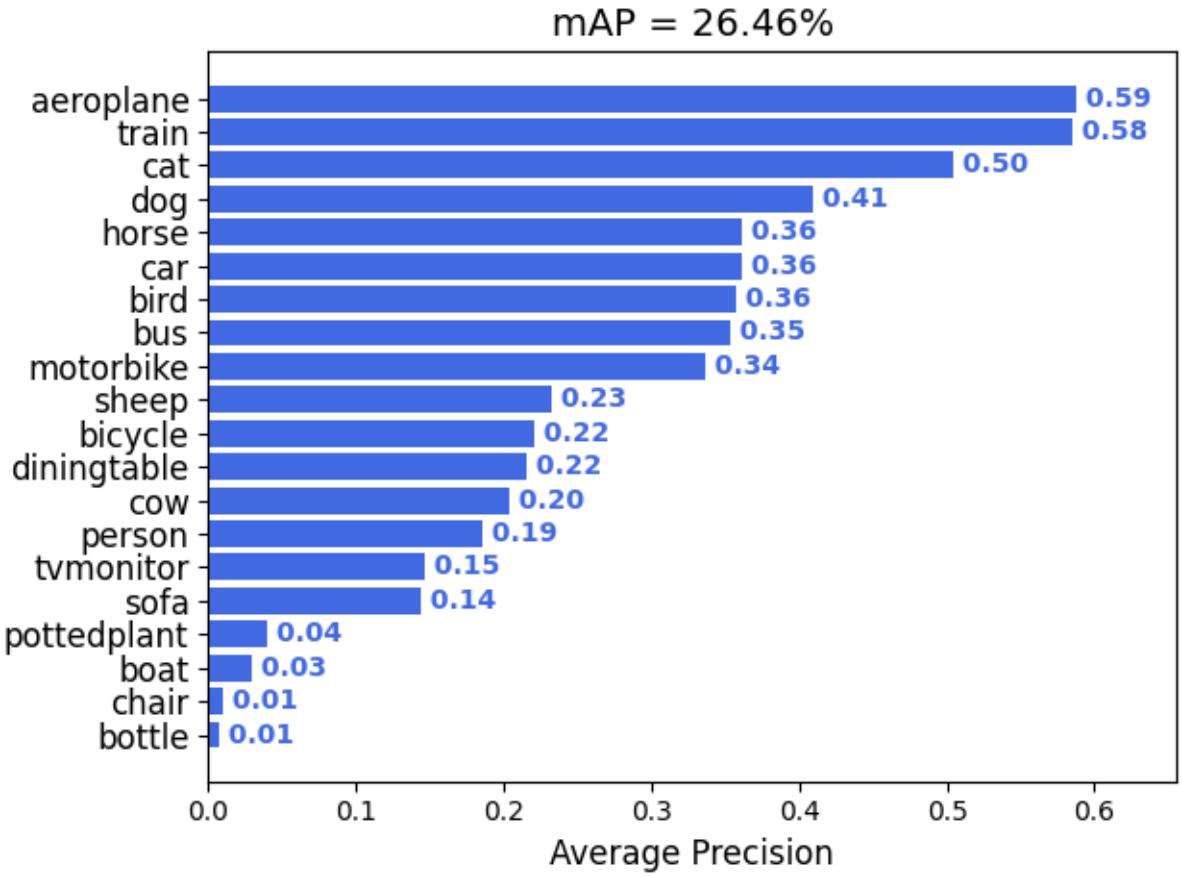

Figure 3.3: mAP plots

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

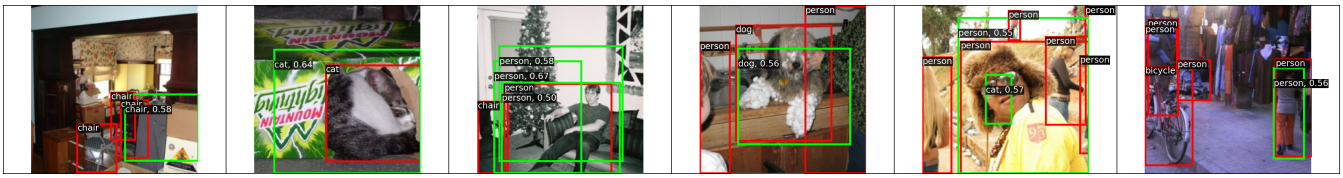

Figure 3.4: Tensorboard Inference Visualization

– 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: Following are the conclusions and insights derived by the visualizations presented above:

1. **Training Losses:**

- The visualizations of `train/loss_box_full`, `train/loss_cls_full`, and `train/loss_ctr_full` depict the box regression, classification, and centerness losses respectively. As an FCOS detector, which operates in a per-pixel prediction fashion, these losses are critical indicators of the model's spatial localization and classification capabilities during training.
- The consistent decrease in all three losses suggests that the detector is progressively learning finer distinctions in both localization and classification tasks.
- The variance in losses, illustrated by the shaded areas, shows the stability of the training process. The FCOS model, known for its robustness to scale variations due to its fully convolutional nature, reflects this attribute in the relatively stable loss curves.

2. **mAP (mean Average Precision):**
   - The mAP, a vital metric for object detection performance, stands at 26.46
   - The model shows proficiency on classes such as "aeroplane", "train", and "cat", which are often well-represented and less occluded in VOC. This is indicative of the FCOS detector's effectiveness in discerning objects with clear, defined boundaries.
   - In contrast, classes like "pottedplant", "boat", "chair", and "bottle" are problematic, likely due to their varied appearance, scale, and frequent occlusion in VOC images. This points to potential areas for dataset augmentation and model refinement.
3. **Detection Samples:**
   - Correct detections with confidence scores are observed in the sample images, showcasing the FCOS model's ability to generalize from the VOC dataset to detect both persons and animals with reasonable accuracy.
   - However, misclassifications and overlapping boxes highlight the challenges in distinguishing between similar object categories and dealing with dense object arrangements, which are common in VOC.

Model Success Cases

- Success is noted where the FCOS detector's inherent strengths align with the data characteristics: objects with distinct boundaries and minimal occlusion are detected reliably.
- Higher AP values for certain classes reflect the model's ability to learn from well-represented data in VOC, leveraging the extensive and diverse examples provided.

Model Failure Cases

- The model's performance dips with occluded objects, those with variable appearance, and those less represented in the dataset, a common issue for object detectors trained on VOC due to its real-world complexity.

Recommendations for Model Improvement

1. **Data Augmentation:**
    * To counteract the issues of scale and occlusion, more sophisticated data augmentation strategies should be employed. These could include random scaling, cropping, and the insertion of occluding objects to mimic challenging scenarios.
2. **Class Imbalance:**
    * Given the FCOS detector's fully convolutional nature, it may benefit from a more balanced dataset. Techniques such as oversampling the underrepresented classes or applying focal loss could help focus the model's learning on difficult or rare examples from the VOC dataset.
3. **Training Data Enrichment:**
    * The addition of more varied examples, especially for underperforming classes, would likely enhance the detector's ability to generalize. This could involve incorporating external datasets or further annotating VOC images to deepen the model's understanding of complex objects.

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")

Yes, I did use a few sources on articles on the internet, Stack Overflow, GitHub, and ChatGPT to help with the assignment (particularly troubleshooting errors). They are referenced as follows:

- https://learnopencv.com/fcos-anchor-free-object-detection-explained/: :text=in
- https://medium.com/swlh/fcos-walkthrough-the-fully-convolutional-approach-to-object-detection-777f614268c
- https://stackoverflow.com/questions/61814762/anchor-based-and-anchor-free-in-object-detection
- https://stackoverflow.com/questions/68271605/how-to-fix-pytorch-runtimeerror-cuda-error-out-of-memory
- https://stackoverflow.com/questions/71498324/pytorch-runtimeerror-cuda-out-of-memory-with-a-huge-amount-of-free-memory?rq=3
- https://github.com/tianzhi0549/FCOS
- https://github.com/best-of-acrv/fcos

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")

I reviewed the code implementations for Srujan Deolasee and Mehal Aggarwal as a sanity check to make sure the results they were getting were cogent. In addition I helped Zihan Wan understand the loss function for FCOS (question 3), and also Cherry Bhatt with her hyperparameters to mitigate CUDA Memory Error for Question 2. No code or visualization was shared between us.

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