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.

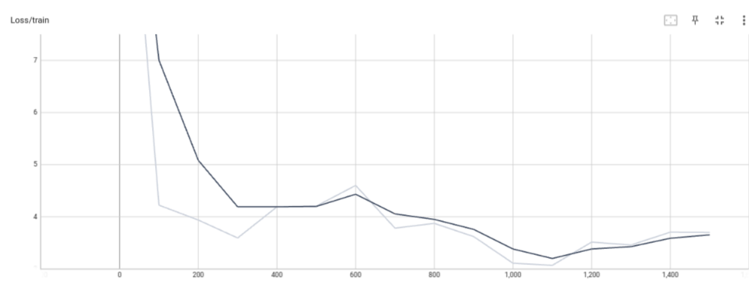


Figure 1.1: `Loss/Train` with and without data augmentations.

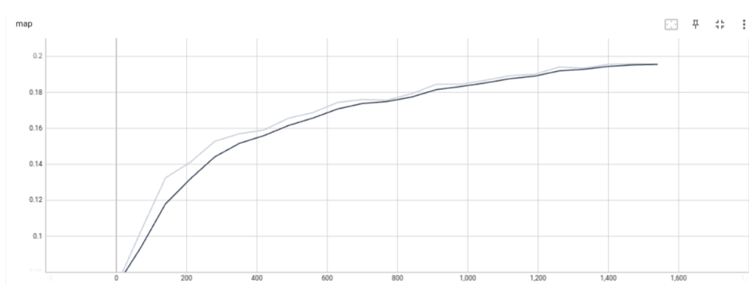


Figure 1.2: `mAP` with and without data augmentations.

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

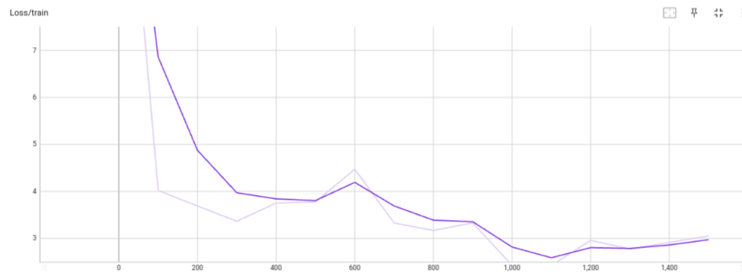


Figure 1.3: `Loss/Train` for simple CNN

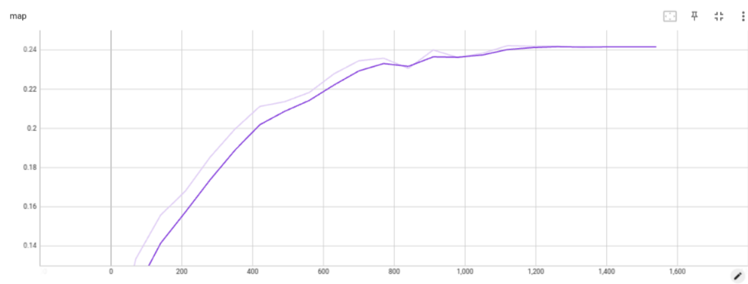


Figure 1.4: `mAP` for simple CNN

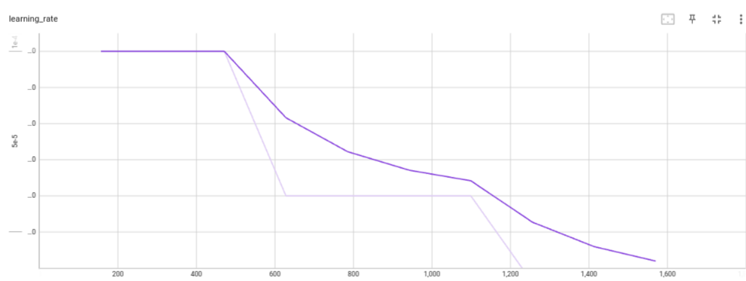


Figure 1.5: `learning_rate` for simple CNN

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

Solution:
I experimented with the learning rate, batch size, step size, and gamma. The learning rate controls the step size that the optimizer takes during the training process, as I increased the learning rate a higher rate did converge faster but it was not very accurate (overshoots past optimal solution) whereas a lower right offered more stability. The batch size represents the number of samples used in each forward/backward prop during training. A larger batch size used a lot more memory and computation (trained slower) but works best with a small learning rate. The step size, though I did not vary this value largely, a small step size seemed to be more stable (how the learning rate is reduced) and for gamma the factor at which the rate is reduced tuning this value mainly impacted the learning rate as it impacted

how ”aggressively” the learning rate decreased at each step.

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`.

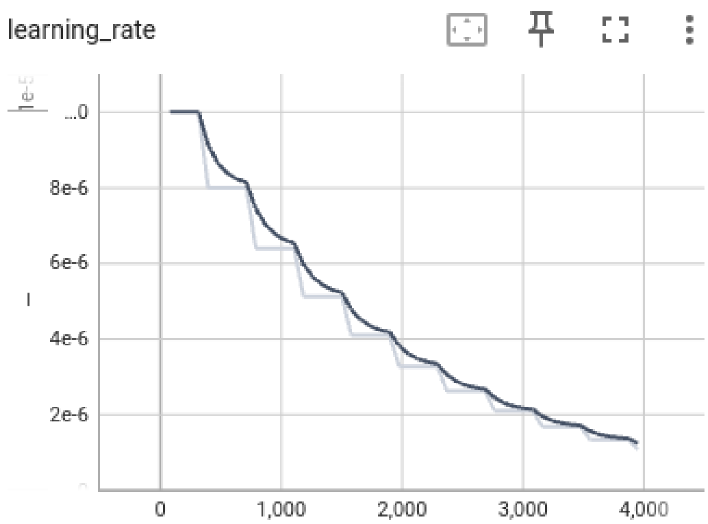


Figure 2.1: `learning_rate` for ResNet

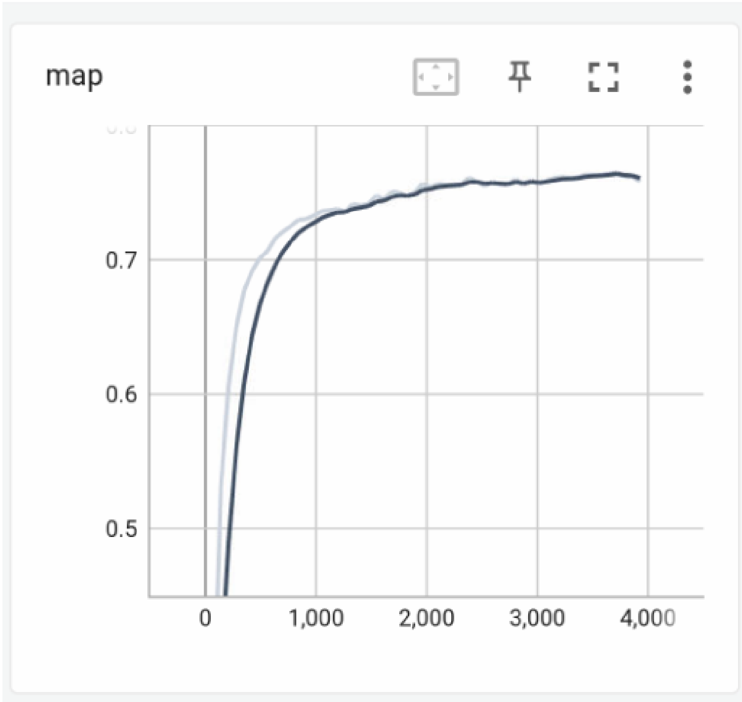


Figure 2.2: `mAP` for ResNet

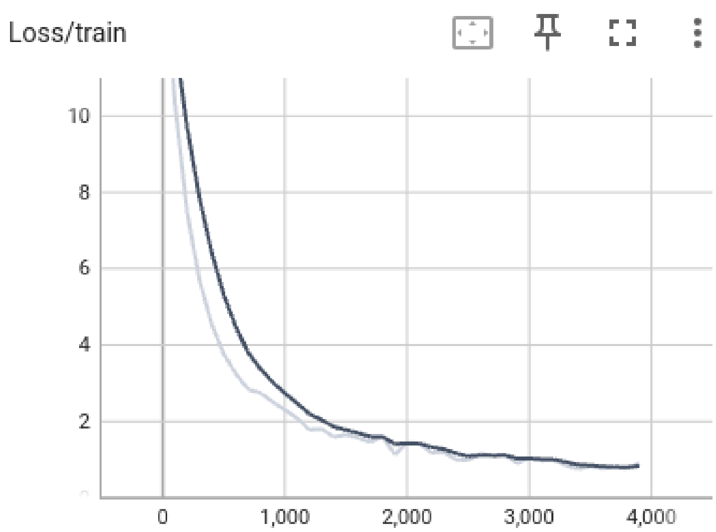


Figure 2.3: `Loss/Train` for ResNet

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

Solution:
The loss/train curve decreases over time throughout the training process, indicating that the model is learning to minimize the training loss that it see. So as the number of epochs increase, the model adjusts fit the training data better and we see a reduction in the loss

value. While the mAP shows an increase over time meaning that the model's performance on the dataset it improving and becoming more accurate in its predictions throughout the training. This can be due to the transfer learning using pre-trained weights as well as tuning of model parameters for the dataset.

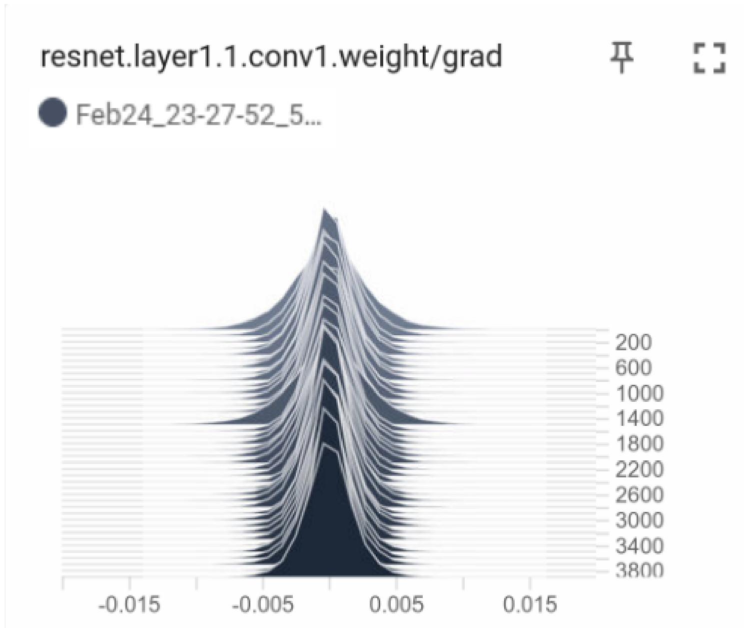


Figure 2.4: Histogram of `Conv1` layer

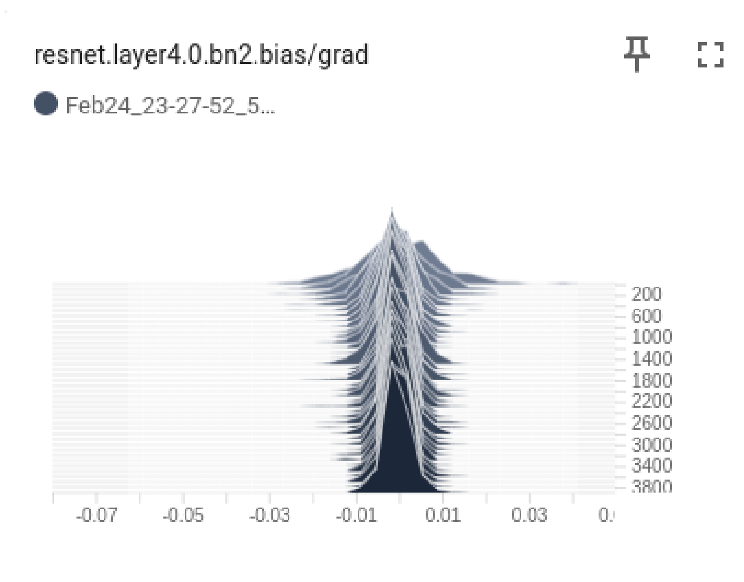


Figure 2.5: Histogram of `BN4` layer

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

> **Solution:**
> Over time as the training progressed, the weights in layer1.1.conv1.weight appear to be more centered around certain values, indicated by the peak frequencies near the center of the weight range. The thinning out of the frequencies towards the bounds of the horizontal (weight) range could mean that as training continues the model is converging to specific weight values for better accuracy. In the layer4.0.bn2.bias, the biases seem to be highly concentrated in the range -0.03 to 0.03, indicating a stableness in the bias values over time as training progressed. The peaks in the frequency in range -0.01 to 0.01 imply that certain biases are necessary for modulating specific feature 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.

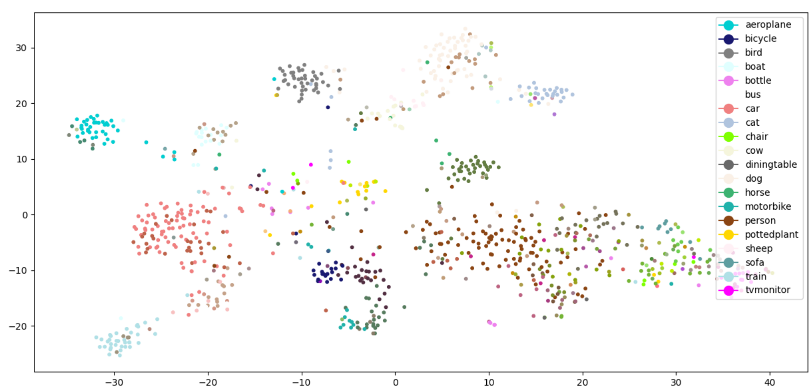


Figure 2.6: t-SNE

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

> **Solution:**
> In the t-SNE plot the clustered dots represents the different classes and what was classified as that specific class. Each color represents a different class, and the clusters show the groups of data points that are similar to each other.

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.

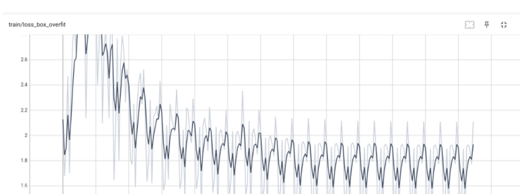
  We have also provided a testing suite in `test_one_stage_detector.py`. First, run the test suite and ensure that all the tests are either skipped or passed. Make sure that the Tensorboard visualization works by running 'python3 train.py –visualize_gt'; this should upload some examples of the training 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.

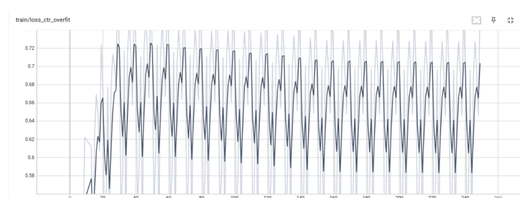
  ```
  cd <path/to/hw/>/detection
  pip install wget
  rm -rf mAP
  git clone https://github.com/Cartucho/mAP.git
  rm -rf mAP/input/*
  ```

  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.

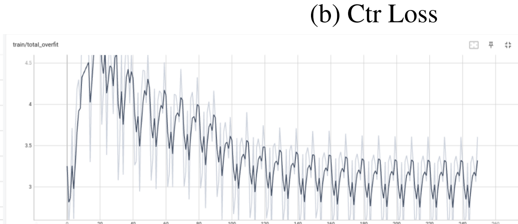


(a) Box Loss



(b) Ctr Loss

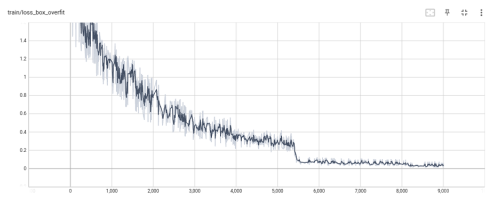
(c) Cls Loss



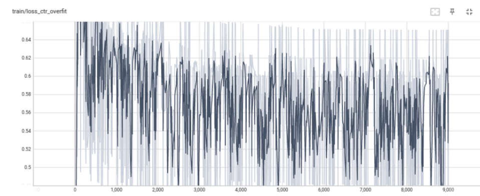
(d) Total Loss

Figure 3.1: Overfit Training/Loss Curve

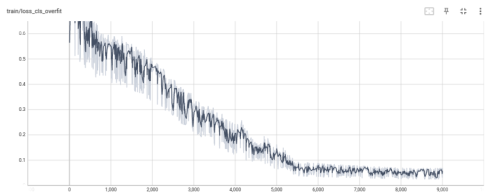
  - Next, train FCOS on the full training set and include the loss curve below.



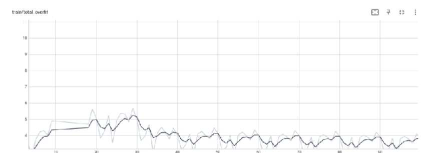
(a) Box Loss



(b) Ctr Loss



(c) Cls Loss



(d) Total Loss

Figure 3.2: Full Dataset Training/Loss Curve

- I was not able to get the mAP to output despite trying on Colab as well as my own machine. The mAP output directory never updated.
- Paste a screenshot of the Tensorboard visualizations of your model inference results from running inference with the `--test_inference` flag on.

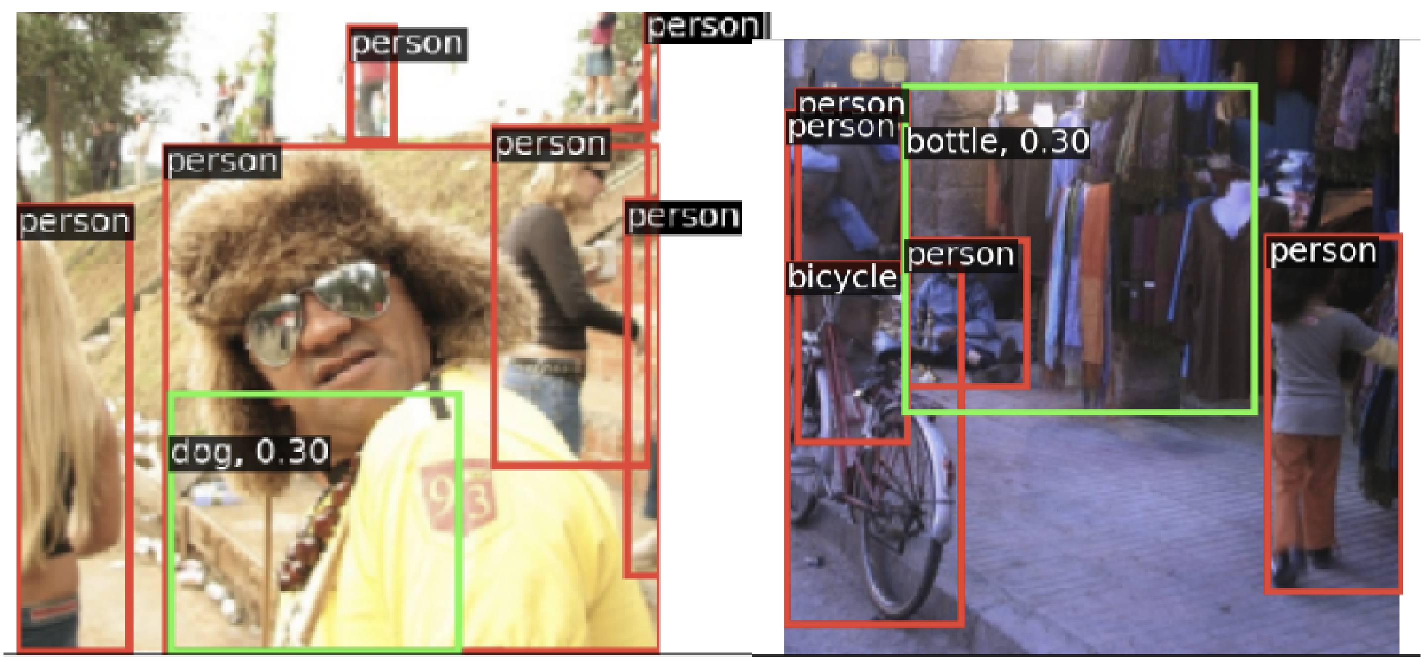


Figure 3.3: Tensorboard Inference Visualization

- 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:
From the above visualisation it seems that the model fails to detect certain cases such as the bottle and the dog, with a confidence score of 0.30 for both. The model is not able to classify the bottle/dog, meaning there is some issue with distinguishing between certain classes. To improve results for these cases, including more diverese examples of the failure classes could help better distinguish differences. And additional tuning of the hyperparameters such as learning rate and batch size could also help improve overall performance.

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.