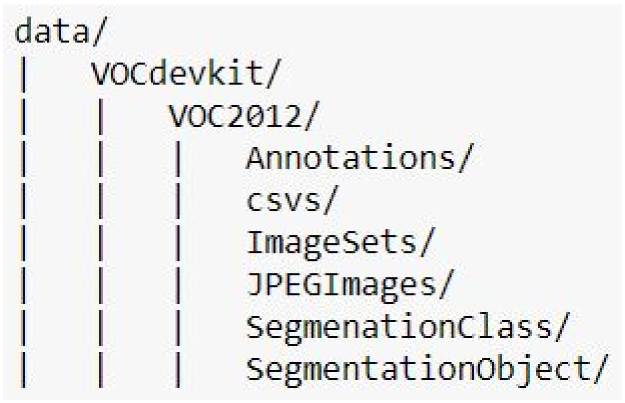
50.039 - Theory and Practice of Deep Learning

The Small Project

Done by - Bhavy Mital (1002945) & Nigel Chan Terng Tseng (1002027)

Steps For Running The Code:

1. Get the dependencies from the ‘requirements.txt’ file.
    a. Access to your preferred virtual environment.
    b. At the root folder of the project, run ‘ pip install -r requirements.txt’.
2. Download the pascal VOC dataset using the following link http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
3. Extract the files straight into the project root folder that we sent. Make sure the directory tree is something like this:



```
data/
|   VOCdevkit/
|   |   VOC2012/
|   |   |   Annotations/
|   |   |   csvs/
|   |   |   ImageSets/
|   |   |   JPEGImages/
|   |   |   SegmenationClass/
|   |   |   SegmentationObject/
```

4. To train the model, please run all the cells in the Ipython notebook file named ‘dl_smoll_project_velocity_9.ipynb’.
5. And grab a cup of coffee while running the code

Model Used:

We used ResNet-18, a convolutional neural network that was trained on the images in the ImageNet database. To speed things up, we used transfer learning (pre-trained parameters from the ResNet) to optimize and help to improve learning in our model.

Loss Function Used:

The output from our model was a vector of 20 values that ranged from 0 to 1, where each of these soft-labels represented a distinct class. As such, we used the

BCEWithLogitsLoss loss function to get the loss for the 20 separate binary classifiers.

This loss function was used as it combines a Sigmoid layer and the BCELoss in one single class, which is numerically more stable as opposed to using a plain Sigmoid followed by the BCELoss. The numerical stability comes from the advantage of combining the mentioned operations into a single layer and using the log-sum-exp trick.

Training Procedures:

Dataset: Using a slightly altered version of the 'vocparseclslabels.py' file provided combined with our PascalVOCDataset class, we generated a dataset where each instance consisted of the image tensor, and it's respective one-hot encoded labels.

Dataloader: Using the dataset generated above, we created a dataloader using PyTorch's in-built dataloader function, where it has a batch_size of 16 and shuffle is set to true. To ensure the reproducibility of our results we used a random seed. (*torch.manual.seed(7)*).

train_epoch() Function: Runs both the train and validation functions for a set number of epochs and returns the train and validation losses and mean average precision scores for each epoch.

train() Function: Our training function does the standard training procedure where it iterates through the training dataloader batch by batch and calculates the average loss per batch. After it finishes training, it returns the average training loss and mean average precision over all classes.

validation() Function: The validation function basically validates and returns validation loss, together with the total average precision score over all classes. It helps us observe the mean average precision and loss after each validation for the current training epoch.

test() Function: We use this function to test our model after a set number of training epochs. This returns the prediction scores and targets for each image instance along with their image paths for the next function.

ap() Function: This function takes the target label tensor, prediction scores tensor and the total number of classes and returns the mean average precision score.

classwise_ap() Function: This function takes the target label tensor, prediction scores tensor and the list of all possible classes and returns a dictionary with classwise average precision scores.

top_bot_scores_50() Function: This function takes in the image_paths, prediction scores, and the class index, and gets the reordered index of the prediction scores from highest to lowest. This function returns the top 50 higest and top 50 lowest scoring images' paths, along with the reordered index.

img_grid_plotter() Function: Helper function for plotting the top 50 images in a grid of 10x5 for a single class.

plot_top_images() Function: Iterates through a list containing pairs consisting of the class index and its respective top/bottom 50 images, and uses the *ImageOnlyDataset* class to create a dataset followed by a dataloader to be passed into img_grid_plotter() function for the image grid plot.

get_tail_acc() Function: This function takes the sorted prediction scores and target labels of a single class and also some other hyperparameters such as t_min (defined by the user), t_max (max of all prediction scores), t_num (defined by the user) and t_vals (linspace from t_min to t_max with t_num steps) and returns the tail accuracy.

Hyperparameters:

For Image Normalization (torchvision.transforms.Normalize)

Mean: [0.4589, 0.4355, 0.4032]

Standard Deviation: [0.2239, 0.2186, 0.2206]

For Dataloader

Batch Size: 16

Shuffle: True

For Optimizer (Stochastic Gradient Descent)

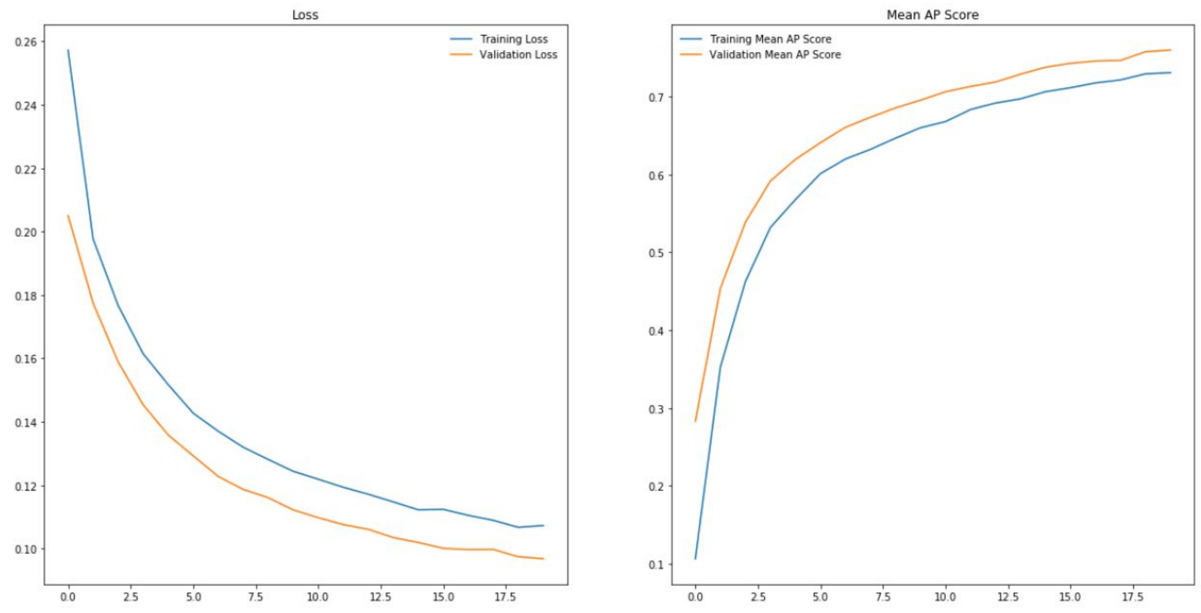
Learning Rate: 0.001

Momentum: 0.9

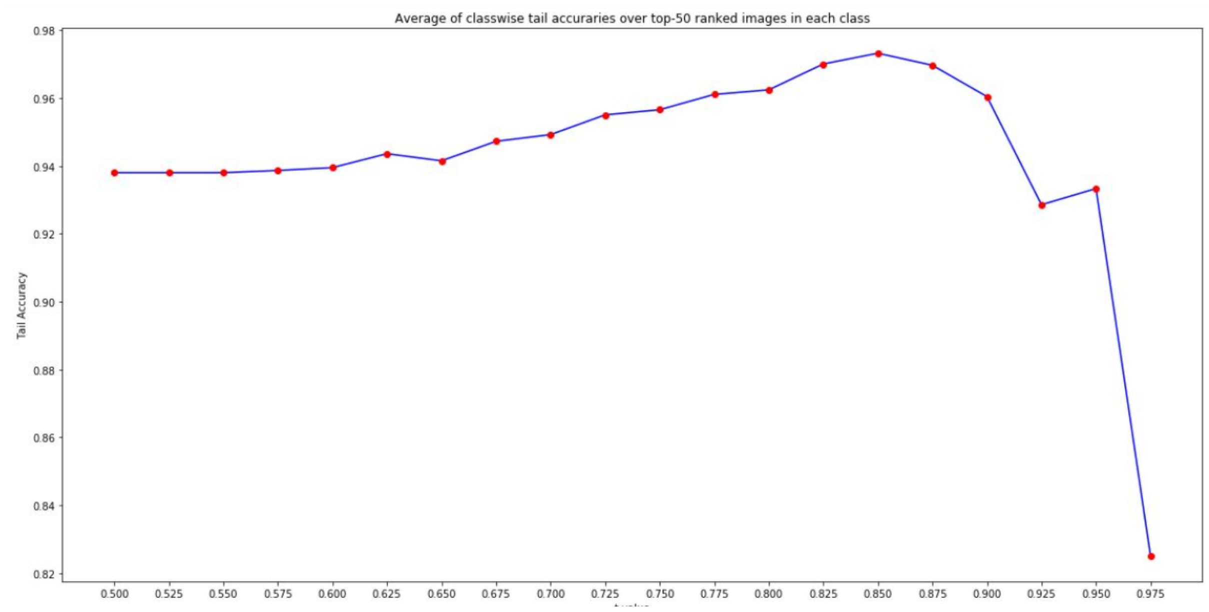
For Training In General

Number of Epochs: 20

Loss and Mean Average Precision Scores over 20 epochs:

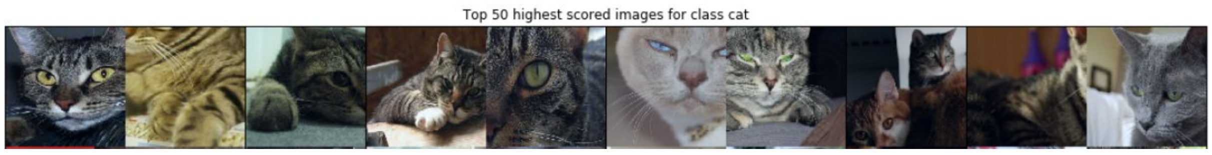


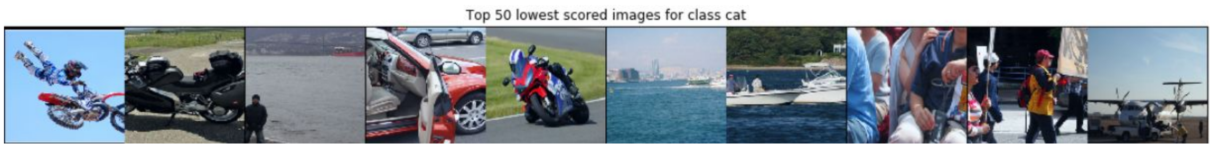
Graph of Tailacc(t) averaged over all 20 classes



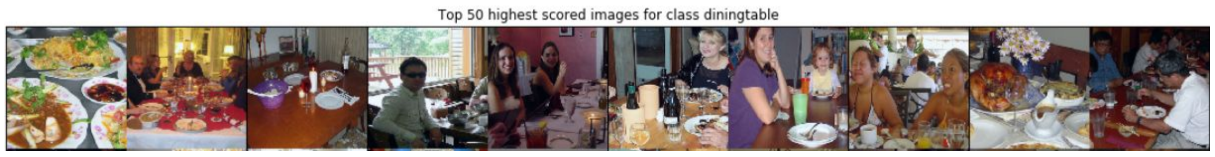
Top-10 Highest vs Top-10 Lowest Scored Images

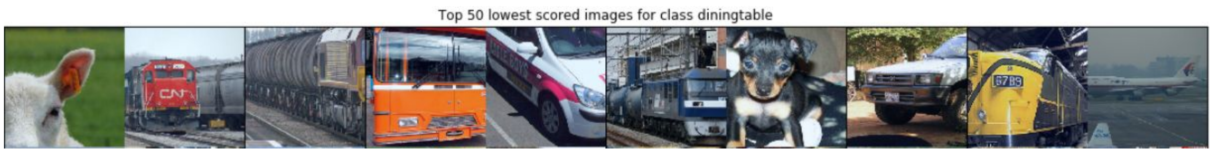
Cat





Dining Table





Bus

Top 50 highest scored images for class bus

Top 50 lowest scored images for class bus

Car

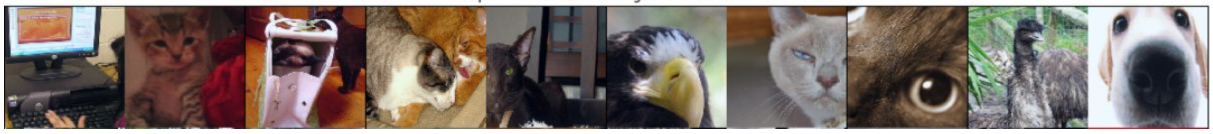
Top 50 highest scored images for class car

Top 50 lowest scored images for class car

Boat

Top 50 highest scored images for class boat

Top 50 lowest scored images for class boat



Plus points for Nezuko please (: