Report

Code
All code is in the file main08.py
Some parts are inspired by the solutions to the weekly exercises.

At line 572 you can make your needed changes in the settings:

```
if __name__ == '__main__':
    # These are the two thing you need to set:
    # 1) path
    data_dir = "/itf-fi-ml/shared/courses/IN3310/"
    data_dir = "/Users/anders/Documents/IN4310/mandatory/"
    # 2) train or evaluate or both
    train = False
    evalu = True
```

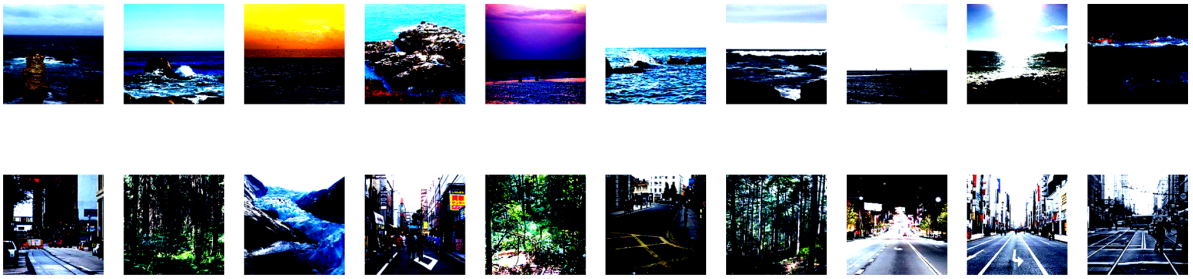
Saved files
Final selected trained model: best_model.pt
Predictions on the test data set: predictions.pt
Softmax scores of the predictions on the test data: saved_softmax_scores.pt
This latter is used in the reproduction routine. So when you run the evaluation part of the code it will produce a new softmax score tensor (softmax_scores.pt), which will be compared with the one I got during running (i.e. saved_softmax_scores.pt).

Reproduction routine
A function (compare_softmax_scores(tolerance=1e-1)) run at the end of the script and it compares the saved_softmax_scores.pt with a file that is made on the fly (softmax_scores.pt).

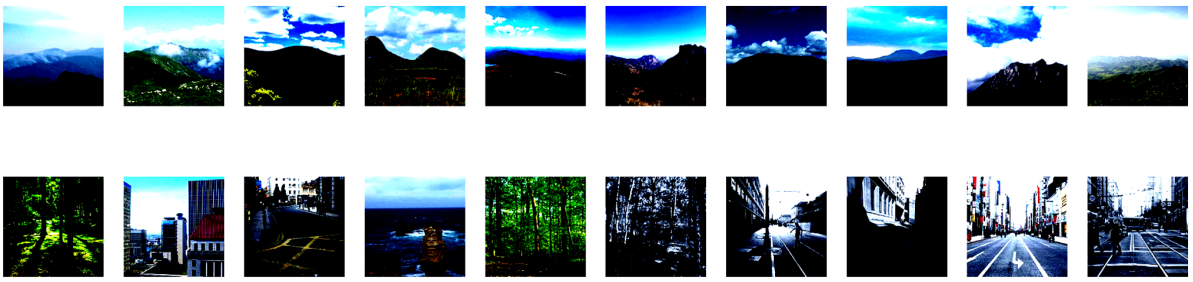
Top-10 ranked images and the bottom-10 ranked images
From the images below it is clear that the model can classify to correct classes. Furthermore the image with lowest softmax score was not put in the respective class. However this dont tell how well the model does on images that are less easy to classify.

Class 1

Class 2

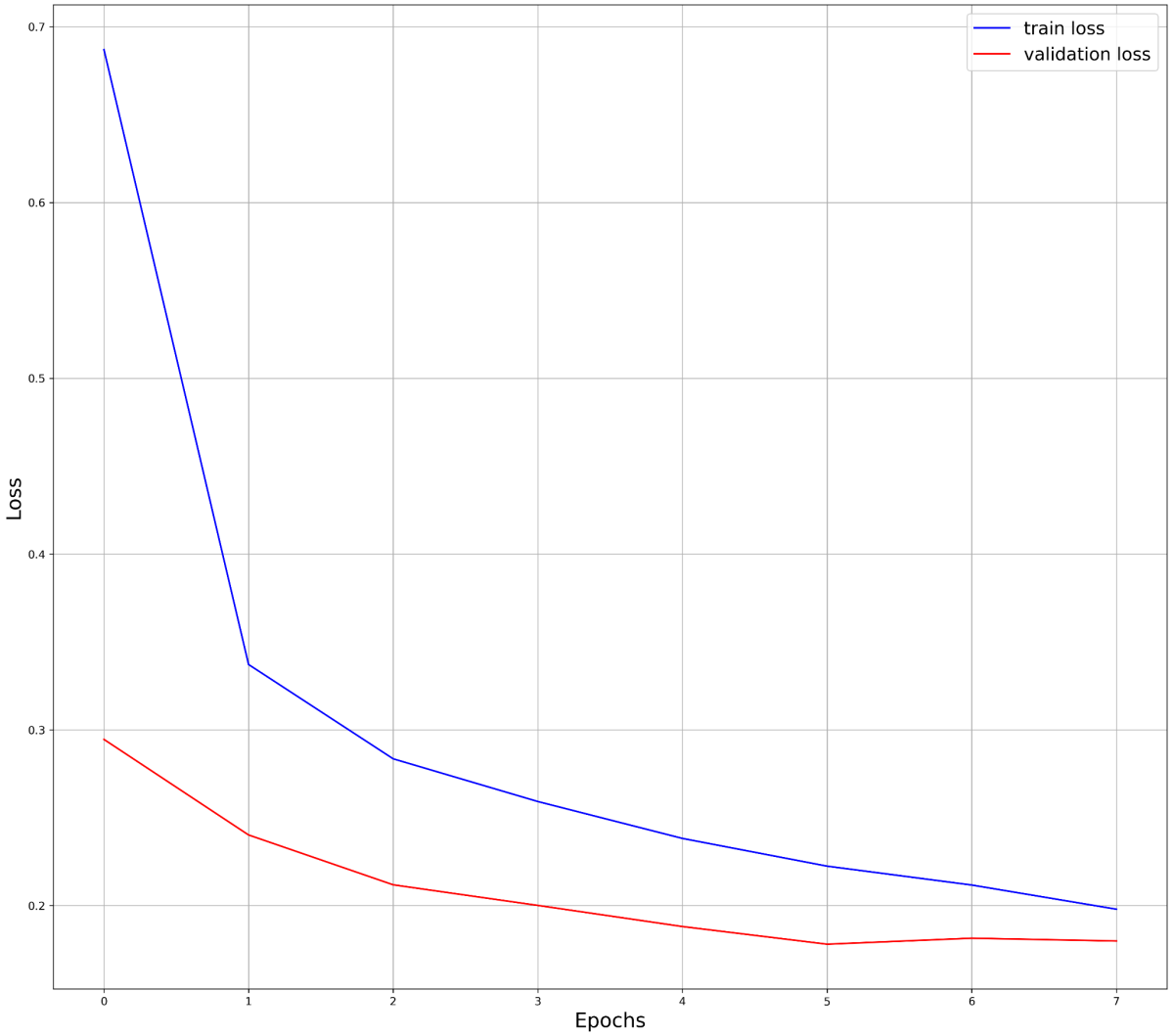


Class 3



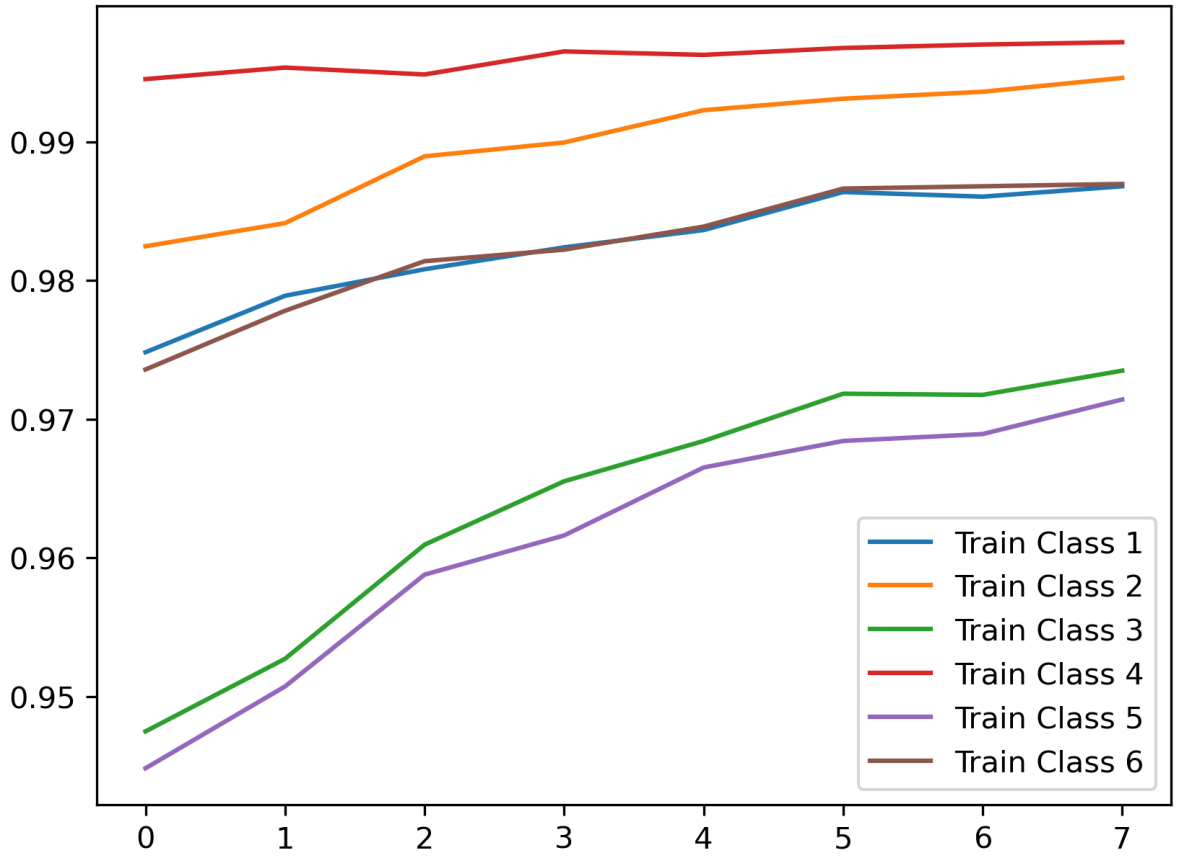
^

The training and validation loss curves for the best model after training

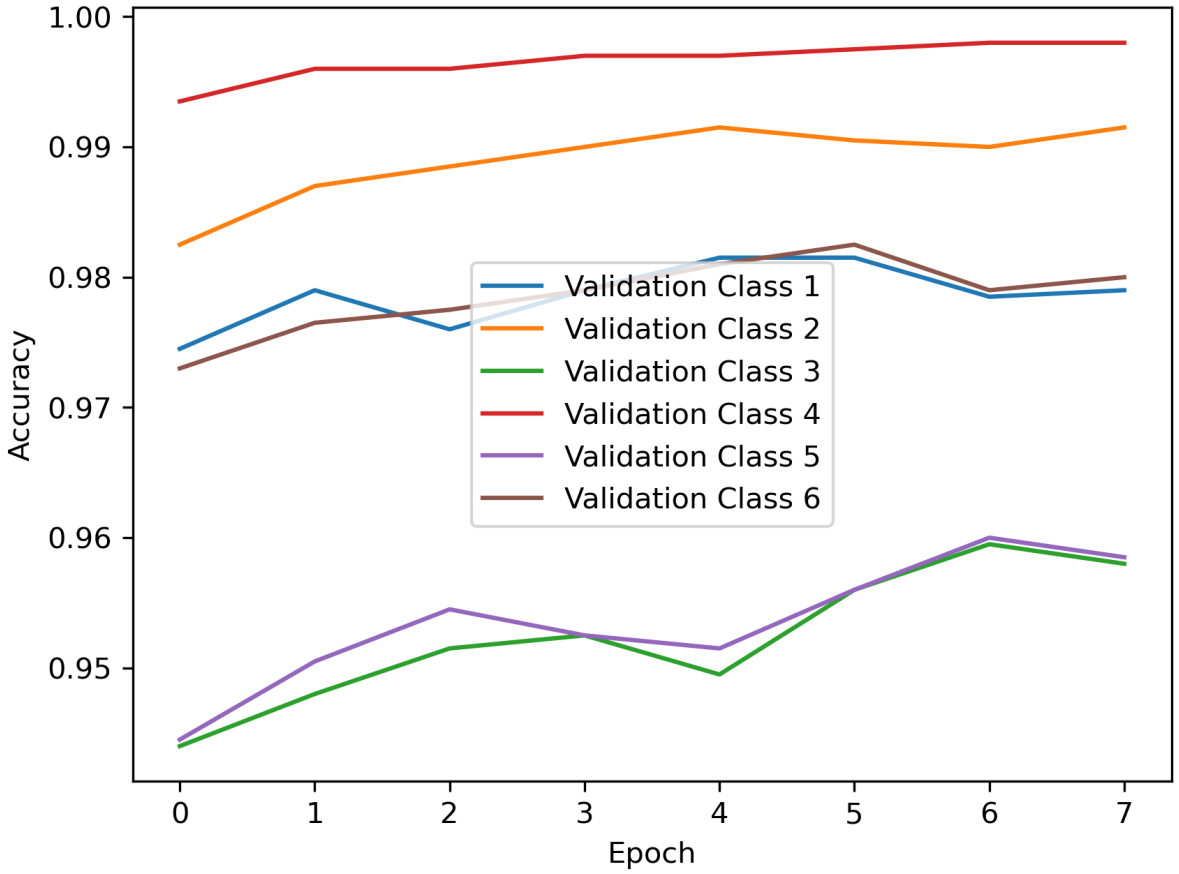


I thought the train loss would be lower than the validation loss, but I think the reason is that I use the mean loss over the batches before updating the weights (backpropagation) as loss for training. I mean the loss used to update the model. Then I have a new model and then I calculated loss on the validation set. So the calculation on the validation set is on a better model. But then again I should be able to just shift the line on epoch; but still validation is lower. Don't get it… ?

Mean class-wise accuracy on training sets for every epoch for the best model



Mean class-wise accuracy on validation sets for every epoch for the best model



Accuracy and average prediction improve as expecgted.

Mean class-wise measurements for the final selected model on test set
Class wise accuracy:
[0.9833333333333333, 0.9133333333333333, 0.93, 0.9333333333333333, 0.8933333333333333, 0.9266666666666666]

Accuracy for all classifications of the model (mean over all classes): 0.93

Class wise average precision score:
[0.9925389074693423, 0.8501109810844649, 0.8706711867993782, 0.8951753203397064, 0.8181644632361181, 0.8596237355085479]

Average precision score (mean over all classes): 0.8810474324062595

Deliverables for Task 2

Module names:

L0 = model.layer1.0.conv1
L1 = model.layer1.1.conv1
L2 = model.layer2.0.conv1

L3 = model.layer2.1.conv1
L4 = model.layer3.0.conv1

Statistics on 5 feature maps from best model on the mandatory dataset:
Percentage of non-positive values:
{'L0': tensor(0.7232), 'L1': tensor(0.6569), 'L2': tensor(0.6281), 'L3': tensor(0.7007), 'L4': tensor(0.7094)}

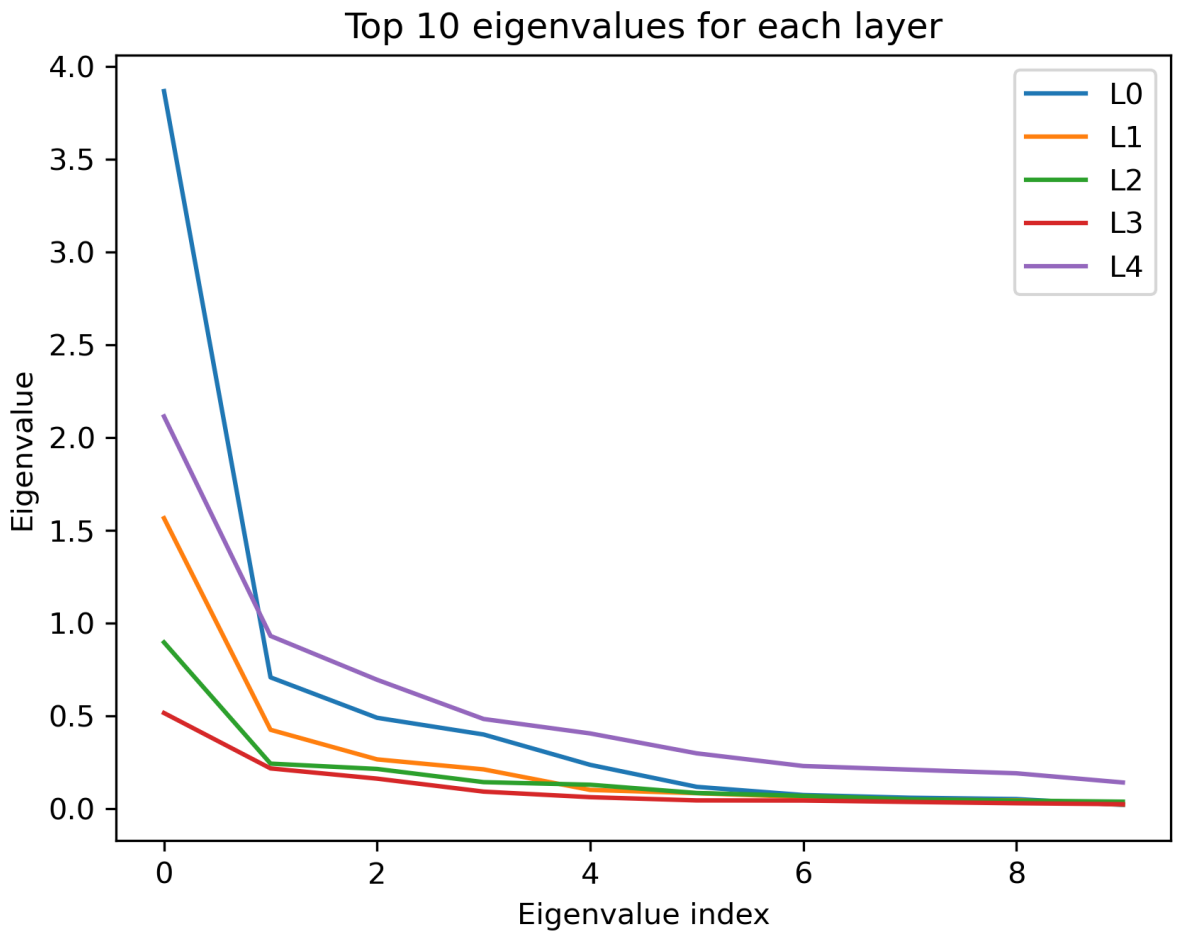
Some comments in the end about the percentage of non-positive.

Deliverables for Task 3

Module names:
L0 = model.layer1.0.conv1
L1 = model.layer1.1.conv1
L2 = model.layer2.0.conv1
L3 = model.layer2.1.conv1
L4 = model.layer3.0.conv1

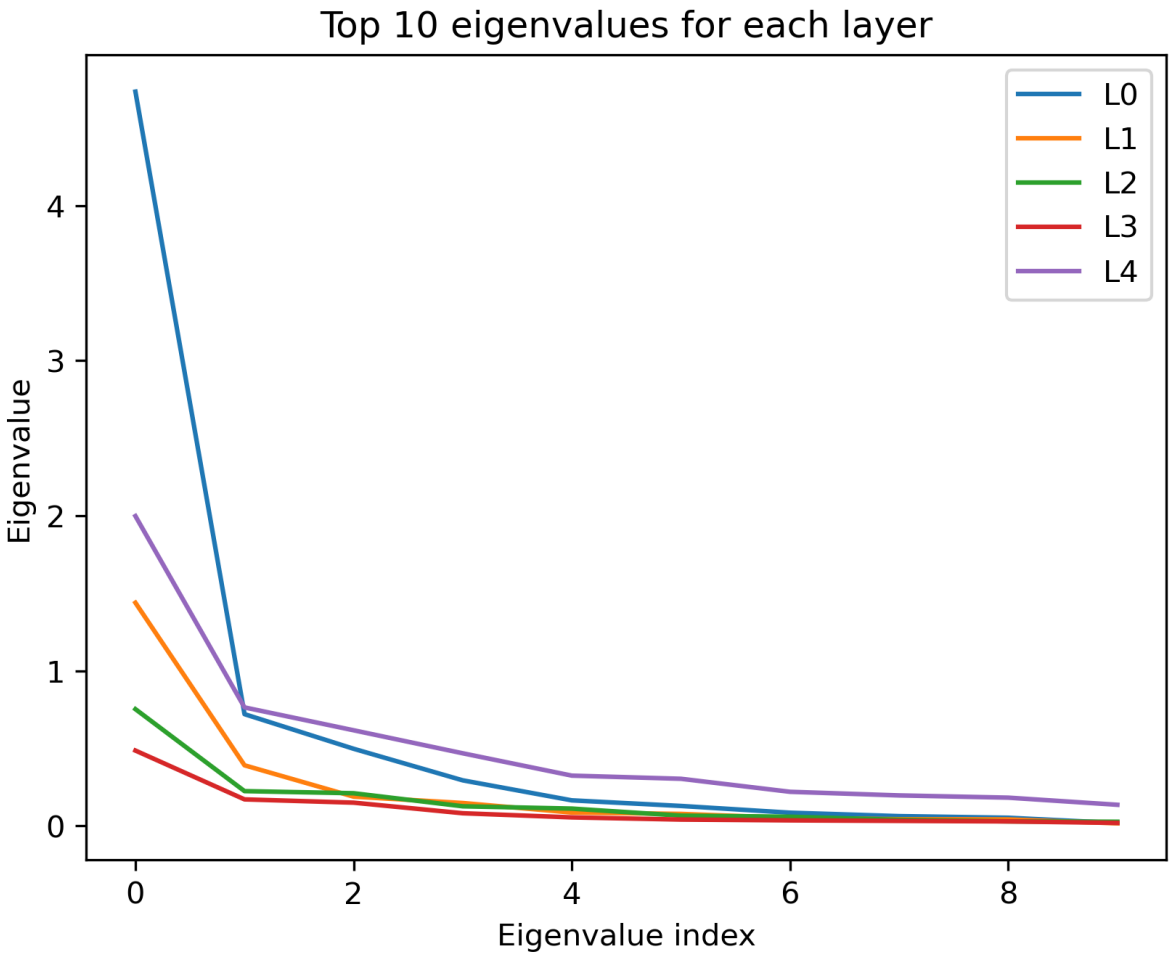
Statistics on 5 feature maps from best model on the mandatory dataset

Percentage of non-positive values:
{'L0': tensor(0.7624), 'L1': tensor(0.7157), 'L2': tensor(0.6373), 'L3': tensor(0.7195), 'L4': tensor(0.7192)}



Statistics on 5 feature maps from NON-fine-tuned model (just pretrained weights)
(the last layer with 6 nodes is just set to random)

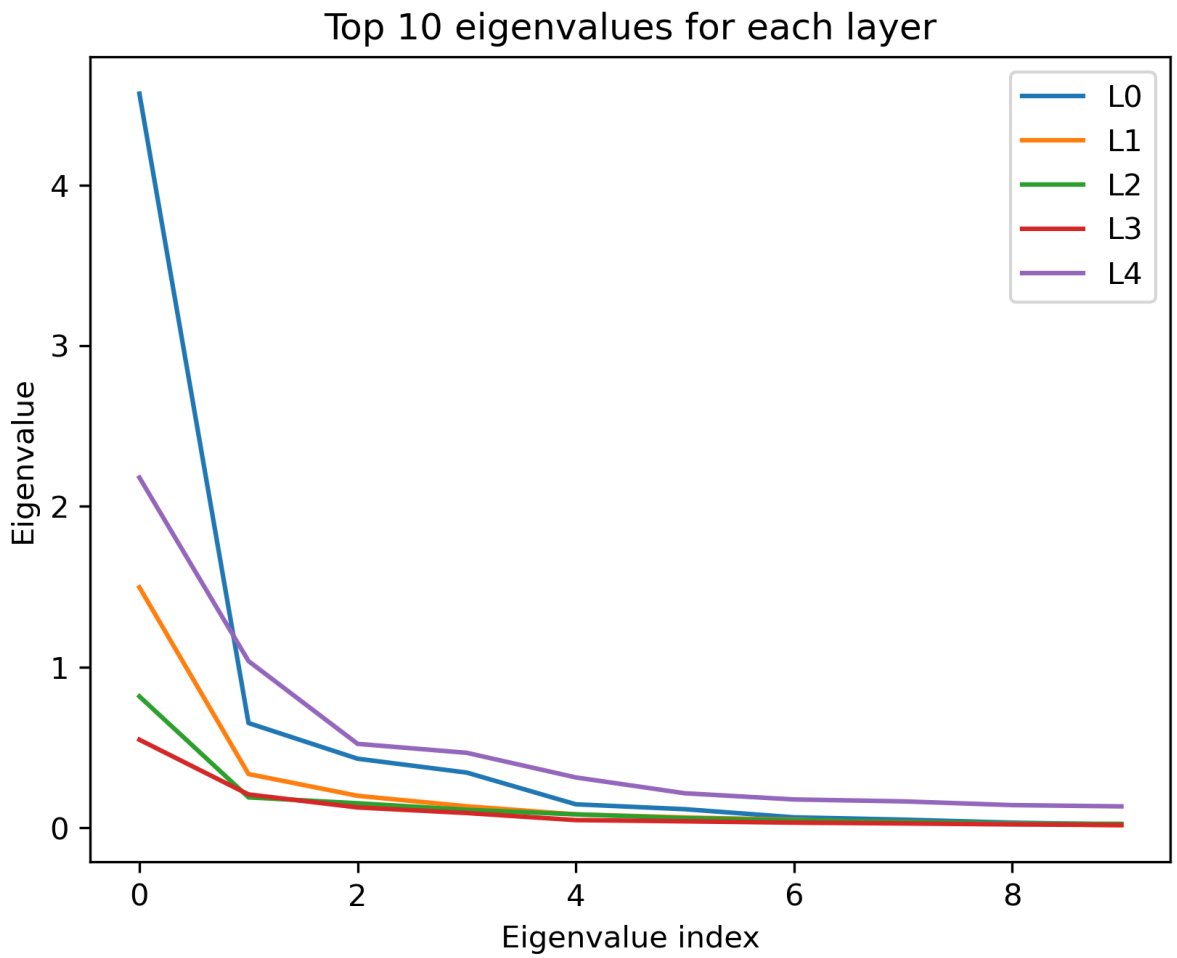
Mandatory dataset:
Percentage of non-positive values:
{'L0': tensor(0.8229), 'L1': tensor(0.7023), 'L2': tensor(0.6370), 'L3': tensor(0.7209), 'L4': tensor(0.7283)}



ImageNet datasdet:

Percentage of non-positive values:

{'L0': tensor(0.8200), 'L1': tensor(0.7039), 'L2': tensor(0.6327), 'L3': tensor(0.7253), 'L4': tensor(0.7183)}



CIFAR-10 dataset:
Dont work. I don't manage to adapt the output of the dataset class so I get the right format out of my dataloader in the feature_map_statistics() function.
I get images out here from the cifar-10 test_batch file (when extracting), but I have written my class dataclass to return paths to the images inside a dictinary. Thus later in the code when I use DataLoader I get a image tensor out for cifar-10, But all the other code was based on getting the dictionary out. I have tried to get around this a lot, but it is too difficult for me…

Comments
For all models and datasets there was a relatively high amount that was not zero in percentages. I guess that tells that something is going on in the layers. There was not so much difference between fine tuned and non-fine tuned suggesting to me that this statistics is kind of crude? But had it all benne zeros I believe the features had been blocked for further analysis by the network.

The eigenvectors for the fine tuned and not fine tuned were to me surprisingly similar. I thought training would lead to layers that learned particular features “more” and that that would be represented by one very dominant eigenvector.