MANDATORY ASSIGNMENT 1 - IN4310

DF

1. Introduction/code setup

The necessary code can be found in the map under the following:

- Task 1: `data_handling.py.` and `NatureTypesDataset.py` using the `/data/buildings...street` for extracting the data and writing to `train.csv(12034), test.csv(2000) and validation.csv(3000)`
- Task 2: uses all the other `.py-files` with exception of `main.py` and `feature_map.py`.
- Task 3: uses `feature_map.py` and saves and generates files to `non_postive_stas.txt`, `/feature_maps` and `/feature_maps_visualization`. The latter displays the images for each layer structurally.
- `main.py` is used to test and primarily run the files for task 1.

2. ResNet Training Results

The files models that were trained on the dataset are saved wit weights in the following `.pth`-files:

- best_nature_classifier.pth
- best_nature_classifier_only_weights.pth
- best_nature_classifier_only_weights_resnet_sgd.pth
- best_nature_classifier_only_weights_resnet18.pth

These were then loaded accordingly in the `accuracy_eval.py`-file. See comments in code on how to run the code and customize it for the different ResNet-models.

2.1. Chosen ResNet architecture and justification and hyperparameter experiments.

For this task I used primarily the ResNet-18 architecture with different optimizers and learning rates for the provided ResNet-18 class. Here I used both the SGD- and Adam-optimizer along with $\texttt{lr} = [0.01, 0.001]$, where `lr` is the learning rate. The ResNet-18 provided by pyTorch(Torch?) was used as is and trained on the provided dataset and customized `DataLoader` from task1. The training procedure proved to be time-consuming and requiring adjustments along the way, thus the first training was limited to only 5 epochs. The number of epochs were later scaled up to 10 to ensure better convergence at the cost of loss of information through transforming the images from 150x150 to 80x80 pixels, to address the problem of runtime. There are some inconsistencies in the plotted graphs. These were not omitted as these are generated at the end of the training procedure, and for the sake of sanity were not repeated.

2.2. Training loss and validation loss over epochs.

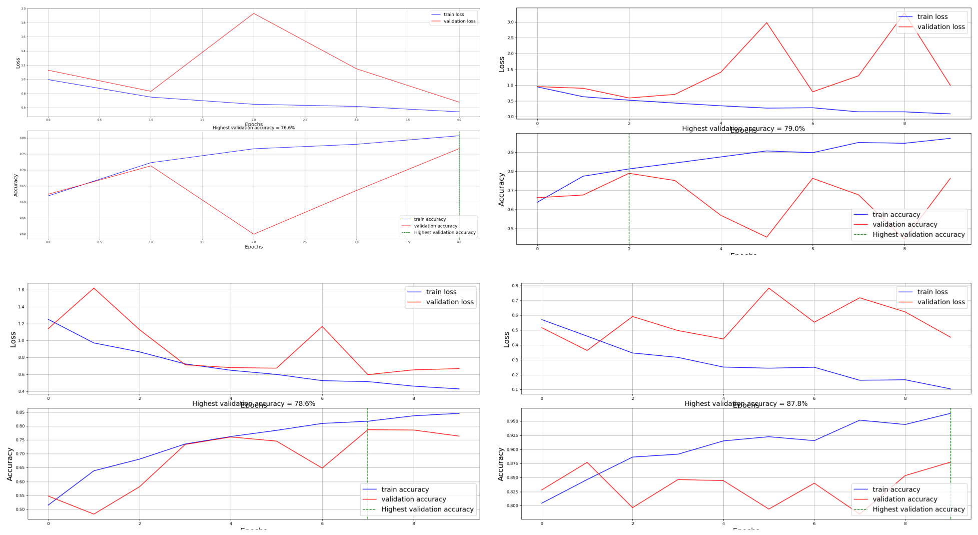

FIGURE 1. All graphs displays the ResNet-18 both loss and accuracy of the architectures' training: *Top-left*: Adam-optimizer with learning rate = 0.001; *Top-right*: SGD-optimizer w/ learning rate = 0.001; *Bottom-left*: SGD-optimizer w/ learning rate = 0.01; *Bottom Right*: Torchvision's ResNet-18.

In this section I point out general features of the validation- and training-loss/ accuracy-graph for the 4 modified models:

- **Validation loss(red):** Unusual behavior with distinctive spikes throughout the training period, suggesting a temporary instability in the model's generalization. Overall trend is that the loss is less at the end of the training compared to the beginning pointing towards a positive trend which is nice. Unfortunately, the validation loss is higher than the training loss. For the top-right graph the validation loss at the end is equal to the start.

- **Training loss(blue):** Shows steady improvement across all epochs by decreasing. The slope is steeper at early epochs but shows a more stable/ slower rate as it progresses, which can imply that the model learns effectively. The overall trend is that the training loss is lower than validation loss, unfortunately.

- **Training accuracy(blue):** Displays the most satisfactory progress as it steadily increases throughout the training schedule, and this without plateauing. Most consistent across the different models.

- **Validation accuracy(red):** Similarly to the validation loss the graphs are varies a lot with distinct spikes and dips, which corresponds to the validation loss. The validation is lower than the training which is a negative trend. In addition, there is a significant gap between the validation and training sets.

2.3. Improvements & Reflection.

10 epochs for the training schedule is relatively short. The optimal choice would be to let the network train for at least 25 epochs, but since I am constrained with hardware and time, I deemed it necessary to move on. Also to begin with when transforming the images to 150x150 resolution, I trained with 5 epochs. This is unfortunate, and afterwards I transformed the images to 80x80 resolution and bumping up the number of epochs. This lack of focus to details is visible in the top-left graph and maybe is reflected in this task. I found it difficult to maintain an overview of hyperparameters and which images belongs to which and their respective resolution. This assignment was difficult because it gave no complete overview of how to structure the code, lack of simple instructions and what felt like extra curriculum made this assignment even harder to complete. Below, you can read some further potential improvements to this assignment:

- More structured approach to code and especially file management.
- Possible dataset anomalies, challenging or unusual images in the dataset led to the model struggle.
- Transforming inconsistencies: this may be the main culprit as to why there were big differences between training and validation since the model trained on images with 80x80 resolution and ran on validation sets containing 150x150, or even 224x224 resolution images.
- Potential overfitting followed by recovery may reflect the optimizer taking bad steps before correcting towards suboptimal/optimal direction.

2.4. **Loss graphs for Testset.**

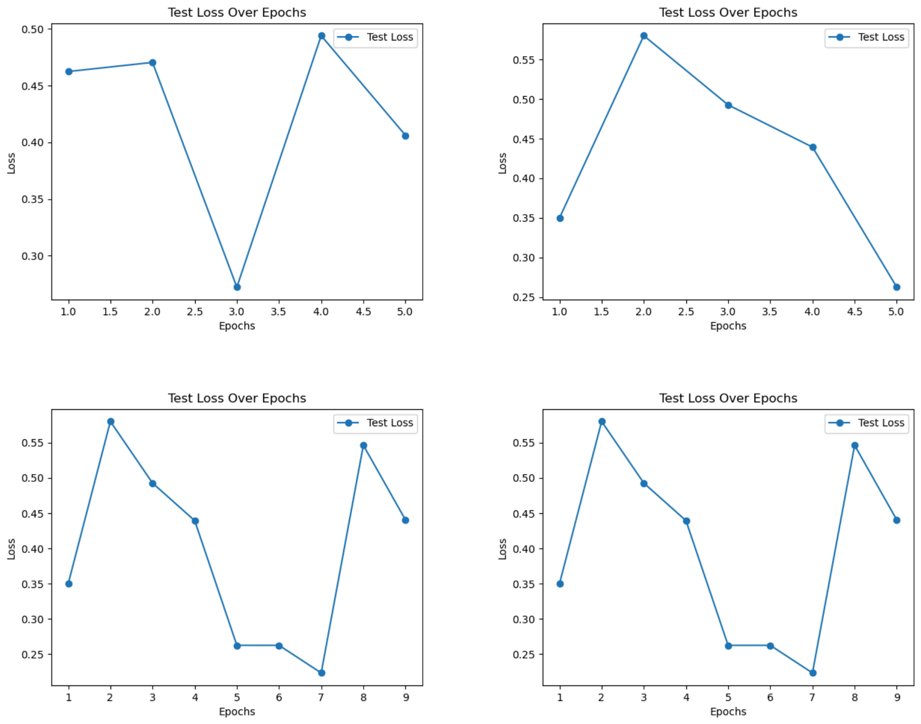

Figure 2. Graphs displaying loss on testset over epochs: *Top-left*: Adam-optimizer, learning rate = 0.001; *Top-right*: SGD-optimizer, learning rate = 0.001; *Bottom-left*: SGD-optimizer, learning rate = 0.01; *Bottom Right*: Torchvision's ResNet-18.

Final model evaluation on testset, see Figure 2. Note the lower number of epochs(5) for top row. These corresponds to the Adam- and SGD-optimizer using learning rate equal to 0.01. Pay attention to top-right graph having similar form as the 2 graphs on the bottom row, when number of epochs are less or equal to 5. It may still indicate potential better generalization over time. The graph displays how sensitive deep models can be to different hyperparameters. For example bottom row may suggest that hyperparameters are closely related or have similar learning dynamics. Lowest test loss is achieved for the bottom row at epoch 7 with a loss equal to ~ 0.25. Similar result are achieved for top-right model at final epoch(5) with a slightly higher test loss.

3. Ap and mAP

ResNet-18	**Buildings**	**Forest**	**Glacier**	**Mountain**	**Sea**	**Street**	**mAP**
SGD, lr= 0.01	1	0.6	0.5	1	0.8333	1	0.8222
SGD, lr= 0.001	1	0.5	0.75	0.3333	1	0.5	0.6805
Adam, lr= 0.001	1	0.6	0.5	1	1	0.3333	0.7388
torch	1	1	0.8333	1	0.6667	1	0.9167

Table 1. Accuracies and mean accuracy for all classes (maP)

The pre-trained model provided from torchvision outperforms the other ResNet with highest mAP of 0.91667. These pre-trained models are typically trained on the ImageNet dataset, which is shown in its superior performance across most classes, scoring 100% accuracy for Building, Forest, Mountain and Street. The SGD optimizer with learning rate 0.01 scores second with mAP=0.8222. Its strength lies in the classes belonging to Bulding, Mountain and Street scoring with a 100% accuracy across these. On the contrary, it scores weaker on Forest(0.6) and Glacier(0.5). The same optimizer with a learning rate 0.001 scores worse. This result might come from a human error as the score with Adam optimizer scores closer to the first SGD-optimizer variant, where the results may have swapped during the test-/validation-process. This kind of mistakes are mentioned earlier in this report(see Section 2.3). The model with the Adam-optimizer and learning rate 0.001 shows mixed results (mAP=0.7388) with middle scores for the Glacier- and Forest-classes. A particular weak performance can be found on the Street-class (0.3333).

3.1. **Feature Map Analysis.**

3.1.1. *Feature Map Visualization.*

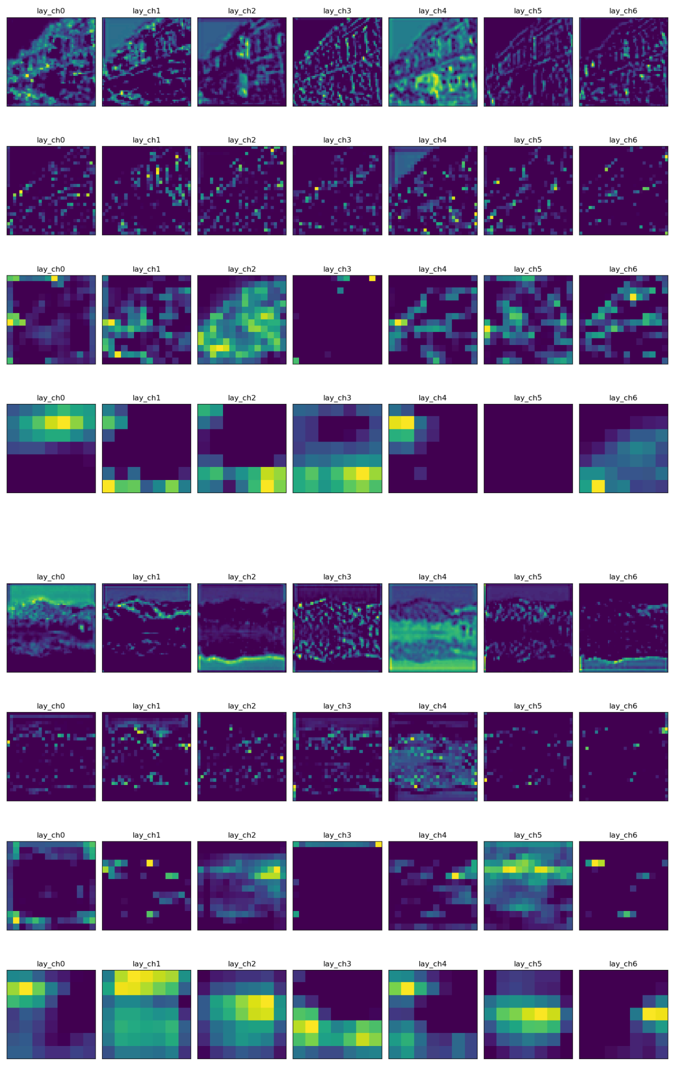

FIGURE 3. Top rows show higher resolution and distinguishable features and represents the first layers. Towards the bottom row, the feature maps become diffuse and pixilated with bigger activation patterns or spots. Top image-bulk shows a building, while the bottom depicts a mountain.

The series of feature maps are organized into 7 different channels and 4 rows, each row depicting a layer in the ResNet-18 network as seen from Figure 3. For each image series, we can distinct the 3 changes of an image throughout the layers:

- Resolution and detail change: top rows has a higher resolution, more features and textures. On the contrary, the images on the bottom row are abstract and pixilated ordered in blocks.
- Channel differences: channel 4(lay_ch4) has consequently brighter areas of green and yellow across the layers. Whereas, channel 1,2 and 5 have more recognizable patterns for the 3 first layers
- Progress towards abstraction: The feature maps at top row are more recognizable compared to the bottom row. The feature maps move towards a higher level of abstraction and less "spatial" details. Here, we can distinguish different activation patterns from the different channels.

The above observations describe the inner workings of a deep neural network through properties such as hierarchal feature learning, receptive field growth, feature specialization and dimensionality reduction. The first property segments the early and late layers into capturing low-level features: edges and textures, as deep layers capture abstract and complex, high level features. The receptive field grows as the layers become deeper resulting in a more blocky representation of feature maps. As for feature specialization different channels specializes in different types of features, thus resulting in different activation for particular patterns, yet difficult to interpret. Lastly, dimensionality reduction occurs as utilities such as pooling and strides are applied resulting in a reduced spatial resolution.

3.1.2. *Non-Positive Value Analysis.*

Module Name	**Avg. % of Non-Positive Values**
layer1.0.conv1	73.63%
layer2.0.conv1	66.76%
layer3.0.conv1	76.42%
layer4.0.conv1	79.90%
layer4.1.conv2	49.54%

Table 2. Non-positive values represented in percentages. Taken from corresping `.txt`-file

Table 2 shows a notably high percentage across the network, with the last layer standing out with the lowest value, and the second to last having the highest with 79.90%. Taking into consideration that zero is given negative value, may suggest that an increase of non-positive values combined with the network using ReLU-activation functions leads to the problem of "dying ReLU". This finding indicates that more neurons tend to output zero, thus reducing the network's performance in learning. A common remedy is to use different activation functions, e.g. ELU, GELU or even SELU.