Machine Learning (E23) Assignment 3

Tobias Washeim: 202007525, Alexander Myrtoft Skjødt: 202004110

November 8th, 2023

Part I

Short Answers

1 Where are we ensuring that the finetuning does not affect the feature extractor?

In `__init__` of `network.py` we have

```
self.pretrainer = nn.Linear(self.lin_size, self.pre_train_classes)
self.generalizer = nn.Linear(self.lin_size, self.generalization_classes)
```

such that when we do pre-training we can use

```
pre_train_optimizer = optim.SGD(network.parameters(), lr=0.01)
```

and when we finetune we use

```
optimizer = optim.SGD(network.generalizer.parameters(), lr=0.01)
```

with only the generalizer, not the entire network.

2 How does the code work that gets s samples per class for the pre-training dataset?

`main.py` first defines `pre_train_subsampled_datasets` using a function from `get_data.py` which "Returns a copy of the dataset that has k samples for each class". It then has a for-loop which runs over a list of sub-sample sizes, also from `get_data.py`. Then this line

```
pre_train_data_subsample = pre_train_subsampled_datasets[pre_train_samples_per_class]
```

picks out a subset of the required size from the set of subsets. This final subset is what is used for training in this part of the testing which has s samples.

3 What is the 'forward_call' parameter responsible for in the 'train()' method (located in 'network_training.py)?

Forward call is a parameter which is used to determine how a prediction is made using the network. In some cases this parameter is just set to be the whole network, in other cases the `forward` or `generalize` functions from `network.py` are used, dependent on what sort of prediction is needed.

The forward_call parameter in the train() method is used by calling the train_step() method and using it as a parameter.

```
    output = forward_call(data)
    if len(target.shape) == 1:
        target = torch_onehot(target, n_classes)

    error = cross_entropy(output, target)
    error.backward()
    optimizer.step()
```

Using forward_call with the data it stores the output which is used to calculate the cross entropy to return the error.

4 Describe how the ‘augment()‘ method works (located in ‘augmentations.py‘).

For some given batch we apply the augment() method. In the code

```
for i, image in enumerate(batch):

    merge_indices[i] = np.random.choice(np.delete(np.arange(batch_size), i))

    new_batch[i], interpolations[i] = augmentation(batch[i], batch[merge_indices[i]])
```

we create a new batch and interpolation by shuffeling the choosen batch. This means we randomly pair up images from the batch. An image cannot be merged with itself (`merge_indices[i]=i` is not allowed). Next, we create images from the new batch and targets by the augmentation method we wrote early in the same file. Lastly we create torch images from the augmentations

```
images = torch.tensor(new_batch)
interpolations = np.expand_dims(interpolations, 1) # for correct broadcasting
merged_labels = labels[merge_indices]

targets = interpolations * np_onehot(labels, n_classes)
targets += (1 - interpolations) * np_onehot(merged_labels, n_classes)
targets = torch.tensor(targets)
```

and create new targets by mixing the one-hot labels of the two original images (by multiplying interpolations).

5 If we pre-train and finetune on the same dataset, is there any reason to do the finetuning step?

When we pretrain, the feature extractor is also changing, meaning the input to the pretrain classifier is not stable/optimal. Thus when we do the finetuning, the data on which the classifier is trained is better, since the feature extractor is now the best it can be, and is locked.

Part II

Predictions

6 Will the collage and mixup data augmentations help achieve higher finetune accuracies? Which do you expect will be more effective?

If the augmentations help bring out important features using the feature extractor (which we expect it will), this should help the fine-tuner, as it will highlight patterns in the data that are important.

We expect the mixup to be more effective, as this augmentation still allows the full features of both images to be extracted, whereas the collage cuts out information from both images, which could lead to some features being missed.

7 What relationship do you expect between the number of samples in the pre-training dataset and the finetuning accuracy? Does this change with data augmentations?

If we have a larger number of samples in the pre-training dataset we can run more batches and therefor expect a higher finetuning accuracy. If we do not have a lot of samples to begin with we willl improve the finetuning accuracy exponentially. At some point when we have added significant many new samples to the pre-training dataset the finetuning will converge to some accuracy and it requres exponentially more samples to improve it.

Part III

Discussion

We have four plots we below where we have pre-trained our neural network on two training sets. The first trainingset (MNIS) are samples of handwritten digits 0-9 and the second trainingset (EMNIST) are samples of handwritten digits 0-27. For each dataset we have a subdataset where we have 1, 2, 4, 8, 16, 32, and 64 samples of each handwritten digit.

We use our trained neural networks on two datasets. One consisten of MNIST dataset and another of EMNIST

dataset. This gives us in total four different combinations we can plot. We plot the finetuned accuracy against the number of samples per class. These plots can be seen on figure 1.

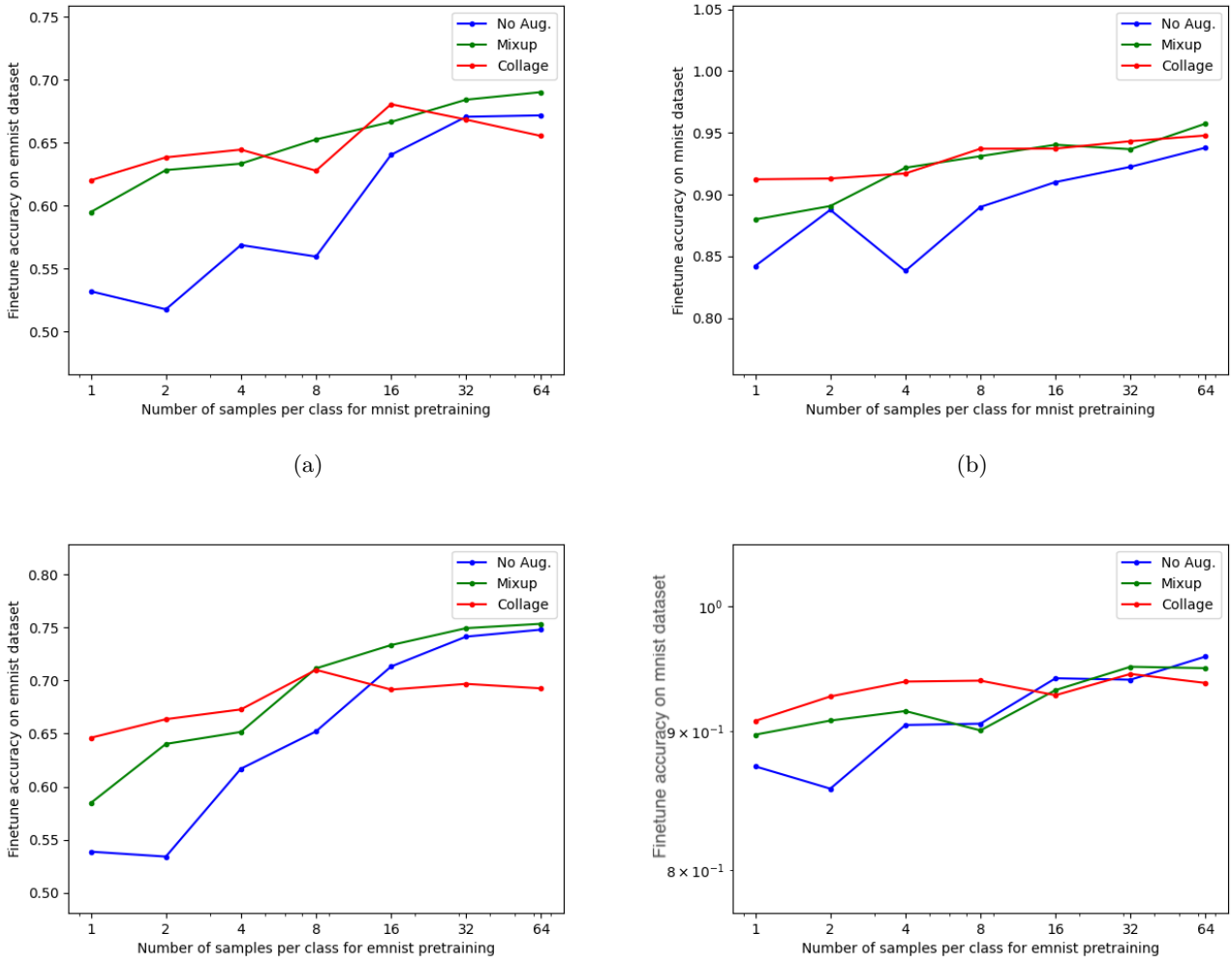

Figure 1

Figure 1.a: Pretrained on MNIST and finetuned on EMNIST

When we look at figure 1.a the finetuned accuracy ranges from 60% to 80% where the accuracy is lowest when the number of samples per class is fewest. It is also noteworthy to see that when augmentation has been applied

it signficantly boost the accuracy for low number of samples per class and when we a relative larger number of samples per class the accuracy for both augementation and without augmentation is roughly the same. The larger our sampleset is the less we will need augmentation.

Figure 1.b: Pretrained on MNIST and finetuned on MNIST

Looking at figure 1.b the accuracy is almost consistently 95% regards from the number of samples per class. It is a bit lower when we only have a few samples per class, but this is to be expected as the neural network will be more accurate the more samples we have. If we have more data to train on we expect to have a higher accuracy as output. If figure 1.a and 1.b we can see that the accuracy is higher for 1.b for all samples per class. This would also have been expected as running our network on the same type of set we have trained it should yield a higher accuracy. We also see the same pattern in figure 1.b as in 1.a when we look at the accuracy for when we use augmentation compared to when we do not.

Figure 1.c: Pretrained on EMNIST and finetuned on EMNIST

On figure 1.c we see a compareable accuracy to 1.a. It makes sense that we more classes have to be distinguished, it is harder to get good accuracy. For the same reason we expect to see the accuracy increase with number of samples, which we indeed do compared to figure 1.b and 1.d. The collage augmentation seems to be especially bad for this setup, whereas the mixup performs the best, but is less effective for larger sample sizes.

Figure 1.d: Pretrained on EMNIST and finetuned on MNIST

On figure 1.d the accuracy is quite high for any sample size, and the difference between augmentations is the lowest of all. This indicated that pretraining on the wider array of letters gives the feature extractor the nessecary capability to very effectively train the network to recognize numbers, even without the help of augmentations.

General remarks

When looking at all the plots we generally see that using augmentations helps the feature extractor do its job, and this effect is most effective for lower sample sizes. The mixup augmentation seems in general to be the most effective for larger sample sizes, whereas it is the other way around for small sample sizes. However the performance difference between the two augmentations varies, is generally small, and so this test is not conclusive.

Comparing the pretraining of MNIST (a/b) and EMNIST (c/d), it does not seem to matter much which is used, as both the small and large sample-size accuracy is the same when finetuning on the same dataset. This indicates

that the feature extractor is effective on either, and that the features it highlights are the same for numbers as for letters.