Lab 4: Finetuning

Nikola Sokolová

April 30, 2022

1 Part 1

In this lab, the vgg11 network was chosen to conduct all experiments. First we calculated the means and standard deviations per color channel over all pixels and all images in the training set in two separate runs over the whole training dataset. Where in the first run we calculated mean over the training dataset and in the second run we compute standard deviation.
The results for the butterflies dataset is as follows.

$MEAN = tensor([0.4631, 0.4483, 0.3237])$
$STD = tensor([0.2830, 0.2650, 0.2754])$

Then we normalize with the statistics which we found to our dataset constructor. The same norm we later use for test dataset.

```
transform_norm = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])
train_data = datasets.ImageFolder('butterflies/train', transform_norm)
```

Then we created validation dataset and train dataset from train data. For this purpose we use $Batchsize = 32$. We use this batchsize within all lab.

The calculation for each dataset is implemented in **main.py**.

2 Part2

In next part, we first freeze all parameters and then we identify and delete classifier part that maps features scores of 1000 Imagenet classes. Then we add new module for 10 classes (replaced last linear layer in the last classifier). Then model architecture was as follows.

```
    VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_r
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_r
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_r
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

For purpose of finding a suitable learning rate we use SGD optimizer and hyperparameter $momentum = 0.9$ and we also use Cross Entropy loss as a loss function. Then we rougly determine the other learning rate order by trying $learningRates = [0.1, 0.01, 0.001, 0.0001]$. After comparing their training loss in 5 epochs, chosen $lerningRate = 0.1$. Results below are for the last fifth epoch.

1. $lr = 0.1, Loss : 0.3599 Accuracy : 0.8930$
2. $lr = 0.01, Loss : 0.2255 Acc : 0.8915$
3. $lr = 0.001, Loss : 0.1338 Acc : 0.8915$
4. $lr = 0.0001, Loss : 0.2255 Acc : 0.8915$

We select a grid of 5 learning rate values around it with which to perform full cross-validation as follows.
$lrs = [0.06, 0.08, 0.1, 0.12, 0.14]$.

```
The best model was the one with lr = 0.14 in 9 epoch.
Epoch 9/19
Train Loss: 0.0850 Acc: 0.9963
Val Epoch: 9 mean loss: 0.23652610193376483, mean acc: 0.9926470588235294
```

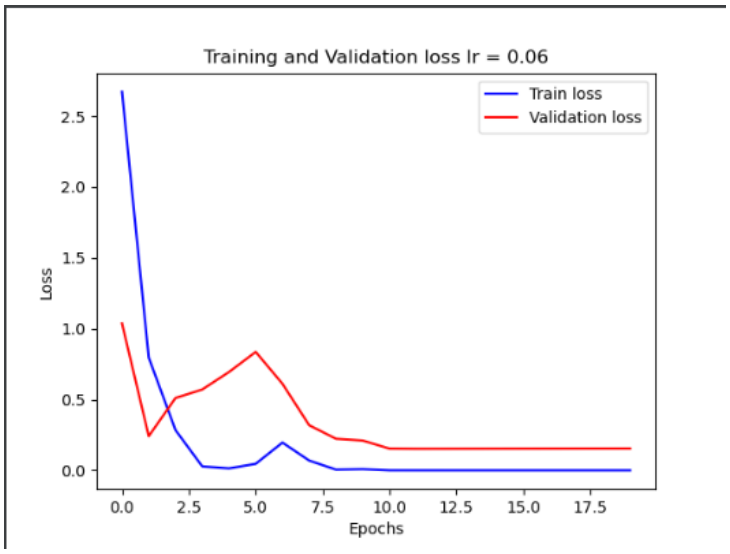

Figure 1: Training and validation loss lr = 0.06

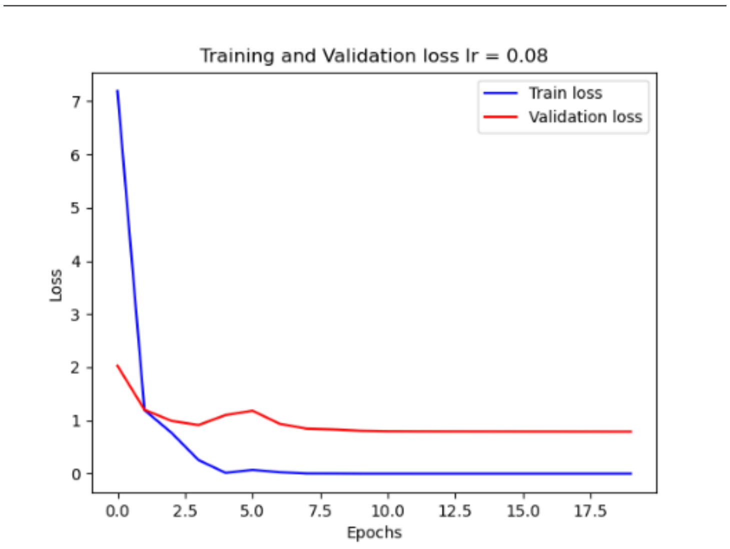

Figure 2: Training and validation loss lr = 0.08

Final test classification accuracy was as follows.

$ModelTestLoss : 0.4949Acc : 0.9956$

```
Network makes some errors on the test data.
In the first case as we can see predicted class was 4 and true class was
Preds
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5,
        5, 5, 2, 5, 5, 5, 5, 5], device='cuda:0')
Target
tensor([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
        5, 5, 5, 5, 5, 5, 5, 5], device='cuda:0')

In the second case we got error for true class 2 and we predicted 8.
Preds
```

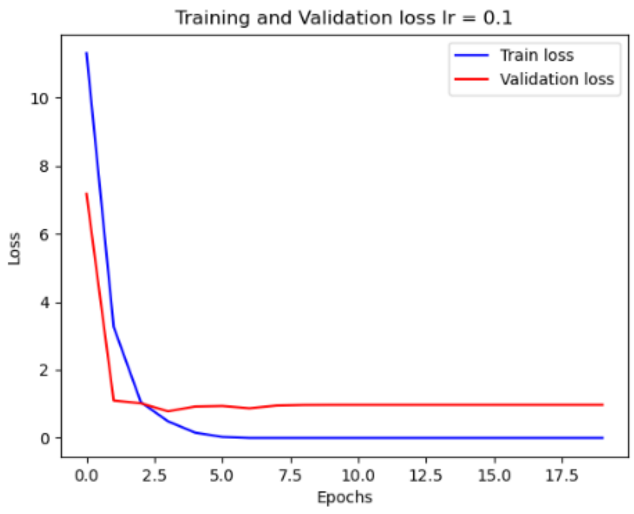

Figure 3: Training and validation loss lr = 0.1

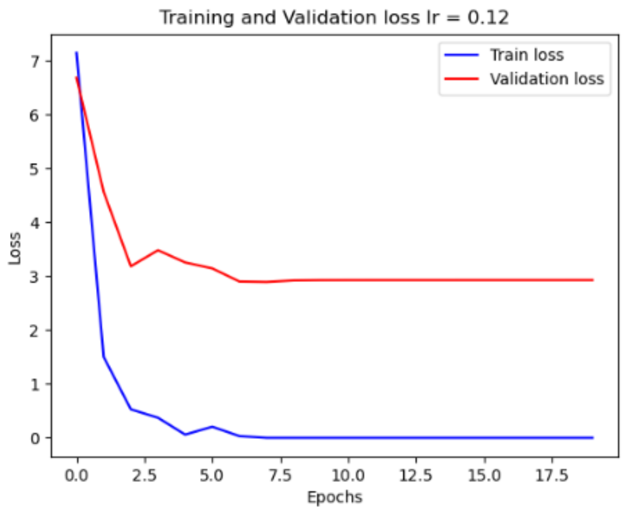

Figure 4: Training and validation loss lr = 0.12

```
tensor([2, 2, 2, 2, 2, 2, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 3], device='cuda:0')
Target
tensor([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 3], device='cuda:0')
```

3 Part3

For model finutune we load pretrained model and do not freeze any parameters. We replace output layer as follows.

```
model = torchvision.models.vgg11(pretrained=True)
model.eval()
#Unfreeze all parameters
```

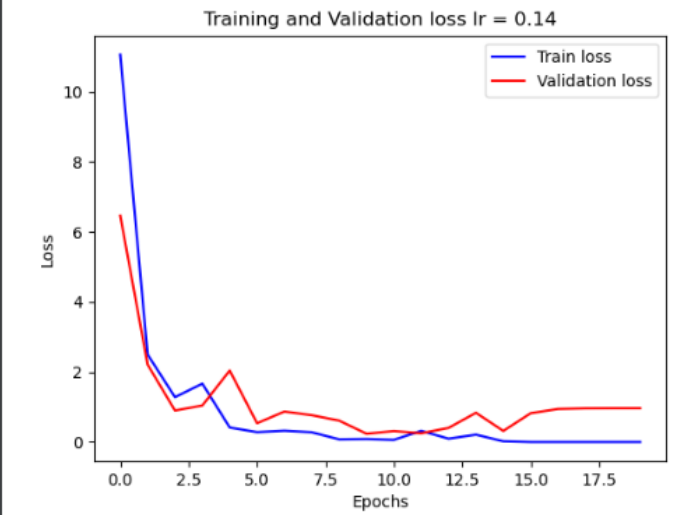

Figure 5: Training and validation loss lr = 0.14

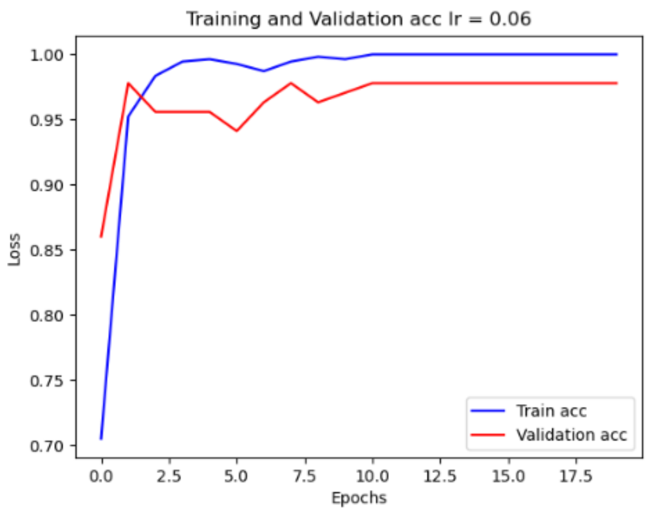

Figure 6: Training and validation acc lr = 0.06

```
for param in model.parameters():
    param.requires_grad = True

def init_weights(m):
    if type(m) == nn.Linear:
        torch.nn.init.xavier_uniform_(m.weight)
       m.bias.data.fill_(0.01)

class_names = train_data.classes
n_features = model.classifier[-1].in_features
n_classes = len(class_names)
fc = nn.Linear(n_features, n_classes)
init_weights(fc)
model.classifier[-1] = fc
```

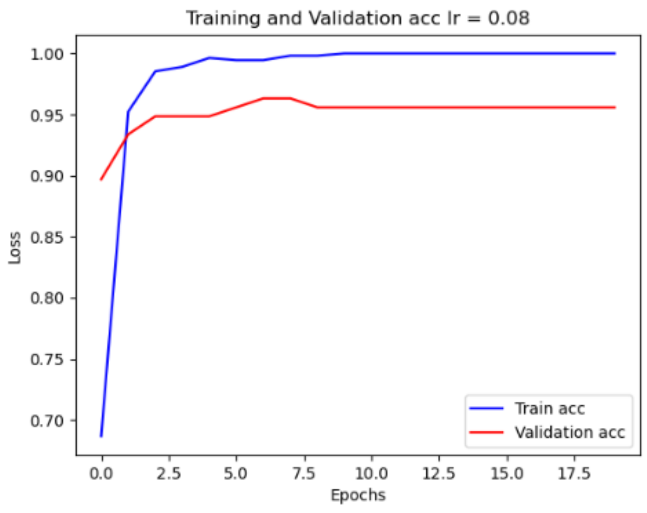

Figure 7: Training and validation acc lr = 0.08

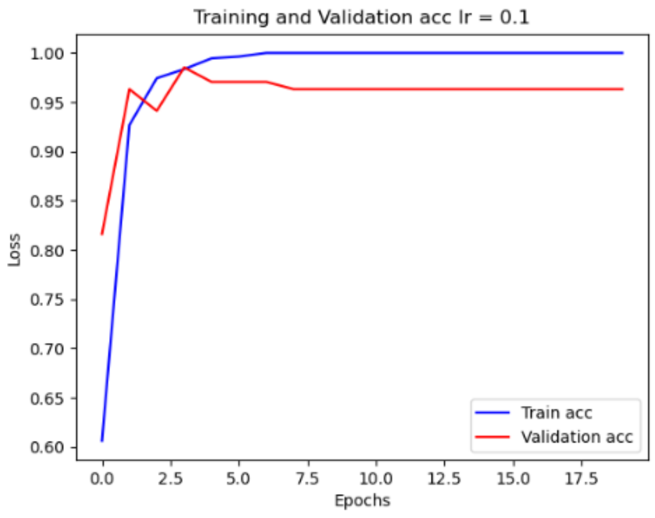

Figure 8: Training and validation acc lr = 0.1

Then we choose small learning rate lr = 0.001 and tried following approche. Using lr scheduler when we decay learning rate with step_size=10 and gamma = 0.1 so lr will decay by by factor 0.1 every 10 epochs. Unfortunately the trainining was too long for 'cpu' and we did not have time to finish this experiment.

Results from training was following.
$TrainLoss : 0.4920 Acc : 0.7965 ValEpoch : 31 meanloss : 0.5082177955370683, meanacc : 0.8125$

We did not receive high val accuracy for this experiment. Number of training epochs was 100 and results above are from epoch 31.

In this experiment, we keep the validation set to be 1/4 as in previous experiments observed the 1/5 to be too small and 1/3 too large a fraction of the training data.

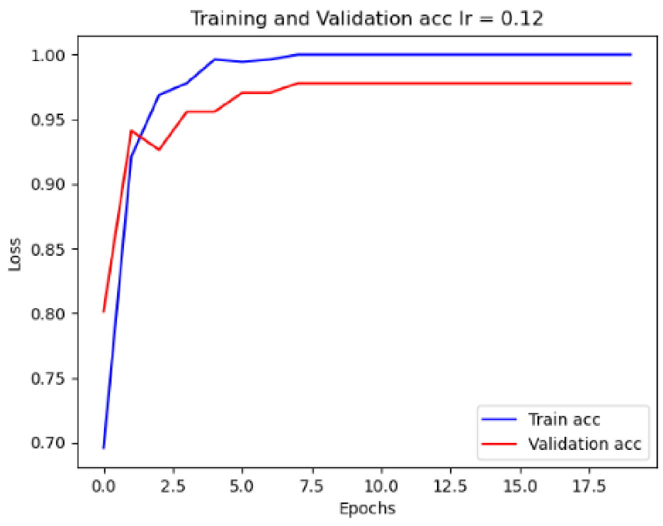

Figure 9: Training and validation acc lr = 0.12

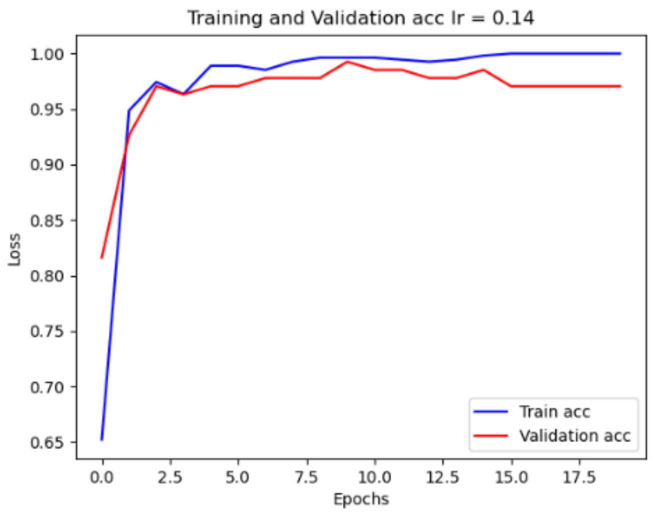

Figure 10: Training and validation acc lr = 0.14