Deep Learning on Rainforest Dataset

Mandatory assignment 1

Bendik Nyheim, 15500

IN5400 - Machine learning for Image Analysis

University of Oslo
Norway
March 2022

Contents

1 How to run

Simply change the root directory in train_pytorch... and run that file. It will load presaved prediction scored and models, calculate new prediction scores on validation set, and compare. Uncomment the function calls under to see the different models train.

2 Method

2.1 Loss

The loss function I am using for this multi-label classification problem is binary cross entropy (BCE) with a Sigmoid layer. In math language, the BCE is written as

$$l(x,y) = -(y \cdot logx + (1-y) \cdot log(1-x)), \tag{1}$$

where x is the output of the last layer. The only difference here is that the Sigmoid activation is used on x, such that

$$l(x,y) = -(y \cdot \log \sigma(x) + (1-y) \cdot \log(1-\sigma(x))), \tag{2}$$

where the Sigmoid function $\sigma(x)$ is defined as

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3}$$

The effect of this function is mapping the output values of the last layer from the domain $[-\infty, +\infty]$ to $[0,1]$. There will be one loss for each of the 17 classes, so a mean of these values (from eq. (2)) are used for the total loss that is used in the backpropagation.

There is a PyTorch function for this with the name **BCEWithLogitsLoss**, which is in the module *torch.nn*. The code looks as follows, where you see the class with the loss function, and an example on how to use it.

```python
import torch.nn as nn

class yourloss(nn.modules.loss._Loss):

    def __init__(self, reduction: str = 'mean') -> None:
        super(nn.modules.loss._Loss, self).__init__()
        self.criterion = nn.BCEWithLogitsLoss(reduction = reduction)

    def forward(self, input_: Tensor, target: Tensor) -> Tensor:
        loss = self.criterion(input = input_, target = target.float())
        return loss


lossfunction = yourloss()
loss = lossfunction.forward(outputs, labels)
```

2.2 Hyperparameters

The hyperparameters used in this experiment are presented in table 1. These parameters are the same for all layers. A step wise learning rate decay is used, where the learning rate decays with a factor of 3 every 10 steps.

Table 1: Hyperparameters used in experiment

Parameter	Value
Learning rate η	0.005
Training batch size	16
Validation batch size	64
Max number of epochs	35
Scheduler step size	10
Scheduler factor	3
Momentum γ	0.9
Torch seed	321
Numpy seed	321
Train test split seed	321

2.3 Models

Three different CNN's were used in this assignment, all rooted in the pretrained ResNet-18. The first model, which I'll call **Single Network**, is just the regular pretrained ResNet-18, where the only modification done is to the number of output classes in the last layer, changed from 1000 to 17. This modification is naturally made for all the three models. The Single Network has three input channels, corresponding to the channels in an RGB-image.

The next model is **Single Network 4 channels**, which is quite similar to the **Single Network**. The difference is that a new input channel is added to the first layer. Note that the number of output features in the first layer still remains the same. The new input channel corresponds to the infrared (IR) channel in the images. The three first channels has the same pretrained weights from ResNet-18, but in the new IR channel, the so called Kaiming He initialization is used.

The third model is **Two Networks**, which is also takes 4 input channels. The difference between this model and the last, that the RGB-channels are fed into one network, and the IR-channel is fed into another. The output features of the next to last layer are concatenated before put into the last linear layer from which the predictions are made.

3 Results and Discussion

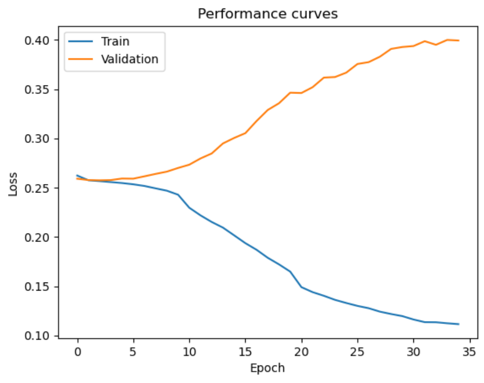
Figure 1 shows the train and validation loss in every epoch for the three different models tested in this exercise. Common for all of these is that they are all overfitting a lot. However, depending on the performance measure, it is not necessarily always better to have the lowest validation loss. In this exercise, we used mean average precision over all the 17 classes to measure the best epoch, and for the Single Network 1a the highest average precision was obtained in epoch 31/34, where the validation loss is clearly much worse than for earlier epochs.

Despite the overfitting it is clear that the models are training as they should, since the train loss keeps decreasing. Figure 2b shows the mean average precision for the Two Networks model. This actually shows that the loss doesn't necessarily reflect the mean average precision. The highest mean average precision is in epoch 7, and from 1c we see that the validation loss is actually slighly higher in epoch 7 in in epoch $0 - 5$.

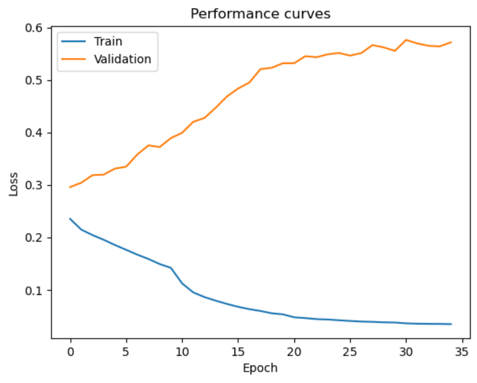
For the Single Network with only RGB-channels i also calculated the tailaccuracy for the images with top 50 prediction scores. For the best performing model by the measure of highest mean average precision over all classes, I looked at the top 50 images for the class with the highest average precision. The tailaccuracy is given by

$$\frac{1}{\sum_{i=1}^{n} I[f(x_i) > t]} \sum_{i=1}^{n} I[f(x_i)) = y_i] I[f(x_i) > t], t > 0.5 \tag{4}$$

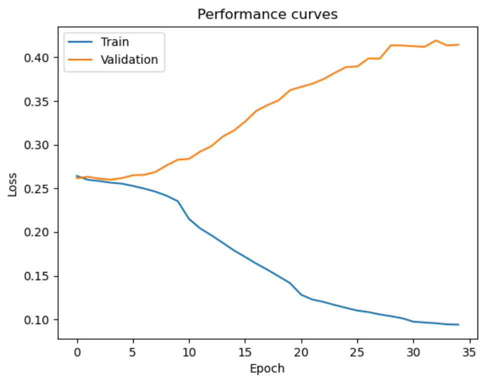
To break it down a little, it just looks at the prediction accuracy when using different values of t as a threshold for predicting a label. Figure 2a shows this. Naturally the prediction accuracy is higher for the top 50 images. This is because the prediction score tells us how certain the model is of a given prediction. We can see that the tailaccuracy for the top images increases with the threshold. This is expected as the model is more certain about the label when t increases. I am not really sure why it drops for higher values of t, but it might have something to do with using the same threshold values regardless of what class I am predicting.
Figure 3 and 4 shows the 1+ images with the highest and lowest prediction score respectively. They do look quite similar, but there seems to be more variation in the bottom predictions. This seems reasonable as more variation in an image makes it harder to predict for the model.



(a) Single network with RGB channels

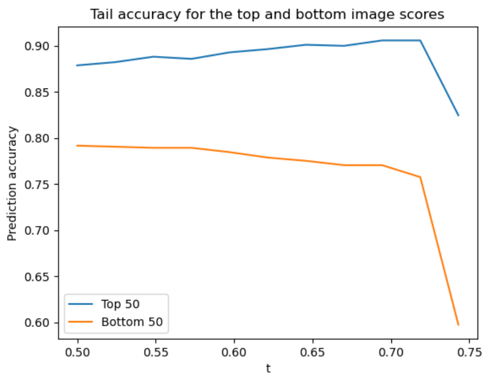


(b) Single network with 4 channels, RGB and IR

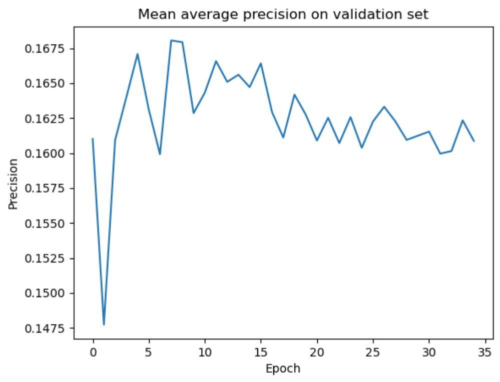


(c) Two networks concatenated at the last layer, one with RGB and one with IR input.

Figure 1: Train and Validation loss for the three different models presented in section 2.3



(a) Tailaccuracies for single network with RGB channels. It is calculated for the 20 images with highest prediction score.



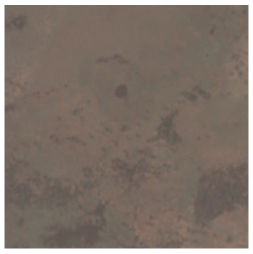
(b) Mean average precision over all classes for the best epoch of the two networks model.

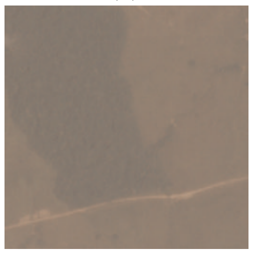
Figure 2

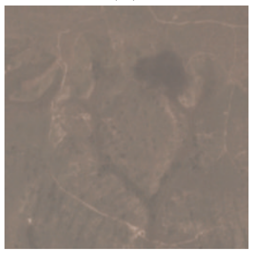
(a)

(b)

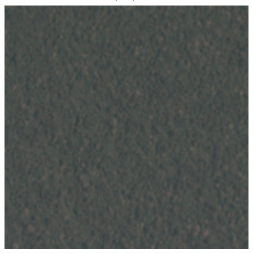
(c)

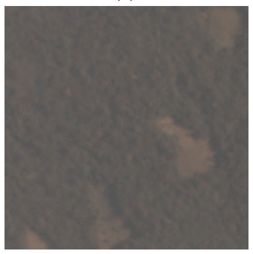

(d)


(e)


(f)

(g)


(h)


(i)

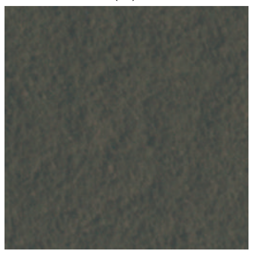

(j)

Figure 3: Images with the top 10 prediction scores.

(a)

(b)

(c)

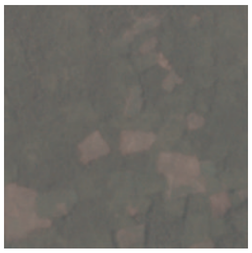
(d)

(e)

(f)

(g)

(h)


(i)

(j)

Figure 4: Images with the bottom 10 prediction scores.