TDT4265 Assignment 3

Anders Thallaug Fagerli

Task 1

a)

As the convolved image should be 3×5, the image I is zero-padded along all sides, giving a padded image of size 5×7. Using a stride of 1 and convolving with the Sobel kernel then gives

$$\begin{array}{c|c|c|c|c|c|c} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 2 & 3 & 1 & 0 \\ \hline 0 & 3 & 2 & 0 & 7 & 0 & 0 \\ \hline 0 & 0 & 6 & 1 & 1 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \quad * \quad \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|c|c|c|} \hline 2 & -1 & 11 & -2 & -13 \\ \hline 10 & -4 & 8 & 2 & -18 \\ \hline 14 & -1 & -5 & 6 & 9 \\ \hline \end{array}$$

b)

The convolutional layer (i) reduces sensitivity to translational variations, as a feature map uses the same weights and bias over the entire image.

c)

Using the formulas at the bottom of the assignment, the new width and height are given as

$$W_2 = (W_1 - F_W + 2P_W)/S_W + 1 \tag{1a}$$
$$H_2 = (H_1 - F_H + 2P_H)/S_H + 1 \tag{1b}$$

where F is the receptive field size, S is the stride, P is the padding, and W_1 and H_1 are the original image width and height. Setting $W_1 = W_2$, $H_1 = H_2$, $S_W = S_H = 1$ and $F_W = F_H = 5$, we solve for P_W and P_H, giving

$$P_W = (5-1)/2 = 2$$
$$P_H = (5-1)/2 = 2.$$

d)

Using (1) and solving for F_W and F_H, we get

$$F_W = 512 - 504 + 1 = 9$$
$$F_H = 512 - 504 + 1 = 9.$$

The dimensions of the kernels are thus 9×9.

e)

Using (1) with $W_1 = H_1 = 504$, $F_W = F_H = 2$, $P_W = P_H = 0$ and $S_W = S_H = 2$, the pooled feature maps are of dimension 252×252.

f)

Using (1) with the pooled feature maps, we get that the feature maps in the second layer are of size 250×250.

g)

The number of parameters are given in Table 1, and the total number of parameters in the network is thus 390410.

Layer	Parameters	Total
1	$(5 \cdot 5 \cdot 3 + 1) \cdot 32$	2432
2	$(5 \cdot 5 \cdot 32 + 1) \cdot 64$	51264
3	$(5 \cdot 5 \cdot 64 + 1) \cdot 128$	204928
4	$(128 \cdot 4 \cdot 4 + 1) \cdot 64$	131136
5	$(64 + 1) \cdot 10$	650
		390410

Table 1: *Parameters in the network.*

Task 2

a)

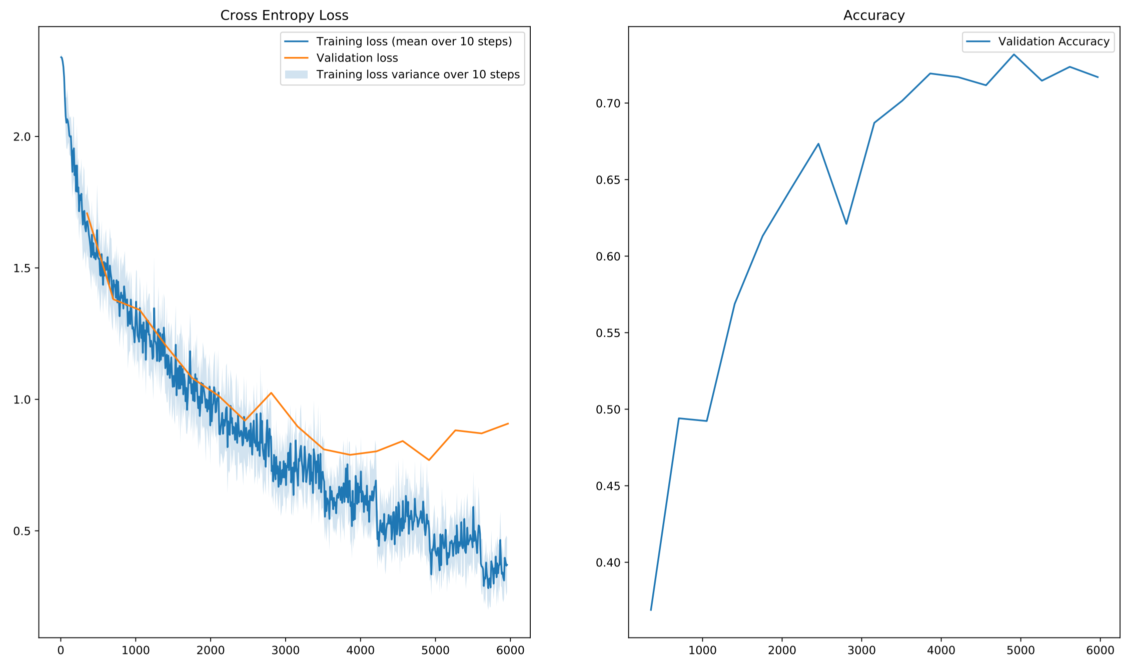

Figure 1: *Training and validation loss (left) and validation accuracy (right) over training.*

b)

Training accuracy	0.8725
Validation accuracy	0.7318
Test accuracy	0.7361

Task 3

a)

Network 1

The first network is almost identical with the one in Task 2, and has the architecture shown in Table 2. The convolutional layers use 5×5 filters, with a stride of 1 and a padding of 2, while the pooling layers use a stride of 2 and kernel size of 2×2. The batch size is 64, and the network uses early stopping. The network uses a SGD optimizer, with no regularization, data augmentation or weight initialization (other than PyTorch's default Kaiming initialization). The learning rate is initially set to 0.05, but the network uses a learning rate scheduler that decreases the learning rate by 0.1 every fourth epoch.

Layer	Layer Type	Hidden Units/Filters	Activation
1	Conv2D	32	ReLU
1	BatchNorm2D	-	-
1	MaxPool2D	-	-
2	Conv2D	64	ReLU
2	BatchNorm2D	-	-
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	BatchNorm2D	-	-
3	MaxPool2D	-	-
	Flatten	-	-
4	Dense	64	ReLU
4	BatchNorm1D	-	-
5	Dense	10	ReLU

Table 2: *Network 1 architecture.*

Network 2

Layer	Layer Type	Hidden Units/Filters	Activation
1	Conv2D	32	ReLU
1	BatchNorm2D	-	-
1	Conv2D	32	ReLU
1	BatchNorm2D	-	-
1	MaxPool2D	-	-
2	Conv2D	64	ReLU
2	BatchNorm2D	-	-
2	Conv2D	64	ReLU
2	BatchNorm2D	-	-
2	MaxPool2D	-	-
3	Conv2D	128	ReLU
3	BatchNorm2D	-	-
3	Conv2D	128	ReLU
3	BatchNorm2D	-	-
3	MaxPool2D	-	-
	Flatten	-	-
4	Dense	64	ReLU
4	BatchNorm1D	-	-
5	Dense	10	ReLU

Table 3: *Network 2 architecture.*

The second network has architecture shown in Table 3. The convolutional layers use identical filters to Network 1, in addition to identical max-pooling. The batch size is 64 here as well. Early stopping is turned off here (as this gave a slight increase in accuracy), and no regularization, data augmentation or weight initialization is used here either, and it uses a SGD optimizer. The learning rate is here set to 0.1, and decreases by 0.25 every fourth epoch.

b)

Metric	Network 1	Network 2
Training loss	0.2460	0.1484
Training accuracy	0.9297	0.9590
Validation accuracy	0.7528	0.8111
Test accuracy	0.7659	0.8193

Table 4: *Performance metrics of the two networks.*

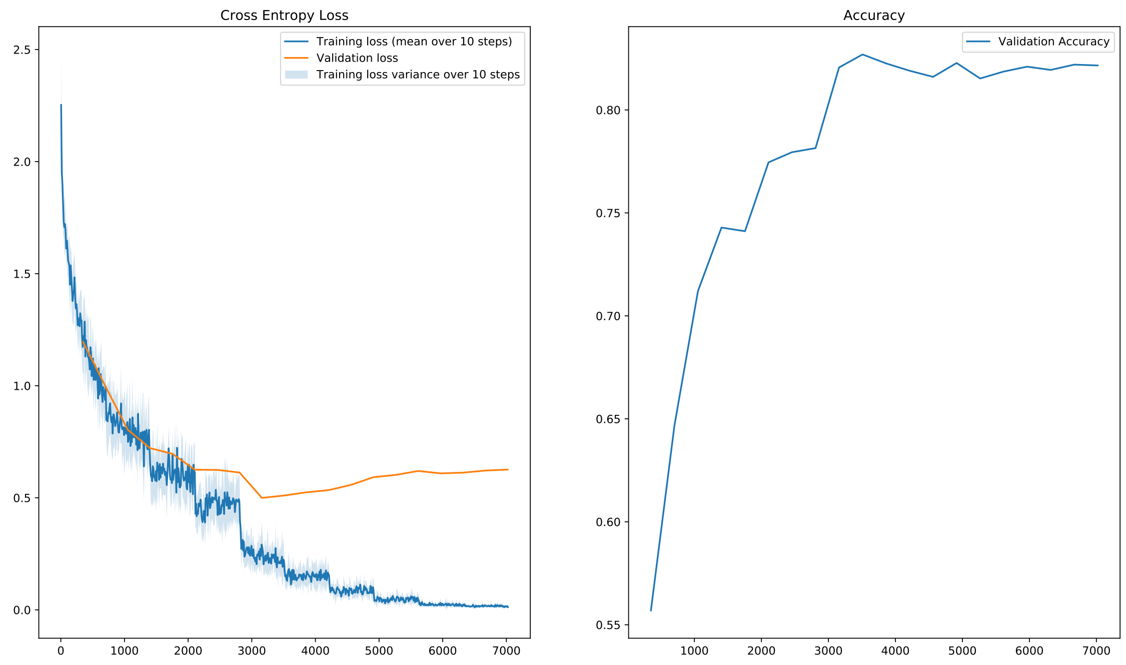

Figure 2: *Training and validation loss (left) and validation accuracy (right) over training for Network 2.*

c)

I used the model from Task 2 as baseline, and iteratively changed parameters and architecture. I found that surprisingly many parameters where good as they were, and that by introducing e.g data augmentation and different weight initialization, optimizer and activation functions, the performance only worsened. However, batch normalization and learning rate scheduling improved the performance by a large margin.

The implementation of data augmentation is probably poor, as it augments both the training and validation set, which may result in poorer performance on the test set. I found that the training set was large enough as it was, and did therefore not feel the need to spend more time on data augmentation.

Different weight initialization methods worked fined, but none gave better performance than the default Kaiming initialization. This apparently works well with ReLU, and keeps the variance around 1, so we avoid exploding or vanishing gradients. This also applies to e.g Xavier and normal initialization, so I am not exactly sure why Kaiming performs better.

I tried different activation functions, where ReLU, ELU and LeakyReLU performed similarly. Using sigmoid or tanh did not work well, as the network converged very slowly with these.

The Adam optimizer was tried out instead of SGD, but gave poor numerical stability. This may only be a matter of tuning the β-values in the optimizer, which would probably fix the issues, but I stuck with SGD as it seemed to work fine. Adam had a slightly faster convergence, so better tuning might give better performance than SGD.

The original filters and pooling layers from Task 2 performed best as well. A smaller filter size on the convolutional layer also worked fine, but larger filters did not perform as well. The reason for this may be that a too large size will not capture the features of the image precisely, making it difficult to learn the specific features of a class. Using strided convolutions instead of pooling did not work either, perhaps because this introduces more parameters in the network.

Batch normalization gave the largest improvement in performance, and was used after each convolution. This increased the convergence speed dramatically, and allowed for a higher learning rate, which in combination with a learning rate scheduler gave best performance. The learning rate was large in the first epochs, giving fast convergence, and was lowered later on as the network converged to a minima.

d)

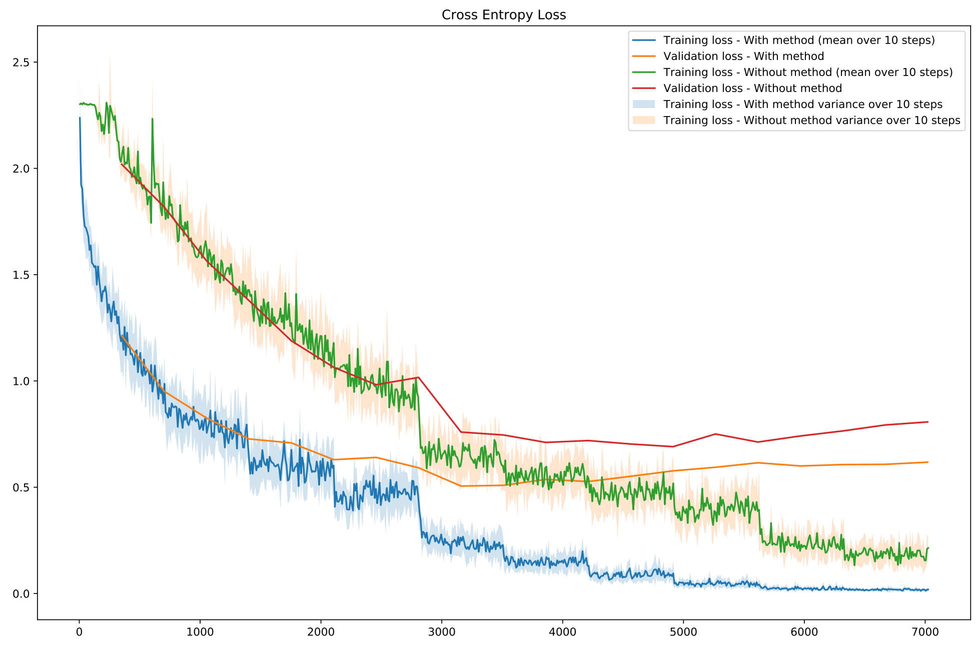

Figure 3: *Comparison of training and validation loss with and without batch normalization.*

e)

Network 2 has an accuracy slightly above 80%, as shown in Table 4, with validation accuracy shown in Figure 2.

f)

In Figure 2 we see clear signs of overfitting, as the validation loss stalls while the training loss continues to decrease. This was remedied by using both dropout and L_2-regularization, but at the cost of a decrease in test accuracy below 80%, so this was not used in the final network. One could perhaps argue that as long as the test accuracy is high, the model is sufficiently generalized. This could however only be that the given test set fits well with the trained model, but other data may not. If given more time, I would therefore try to find a model that achieves above 80% accuracy while also using regularization.

Task 4

a)

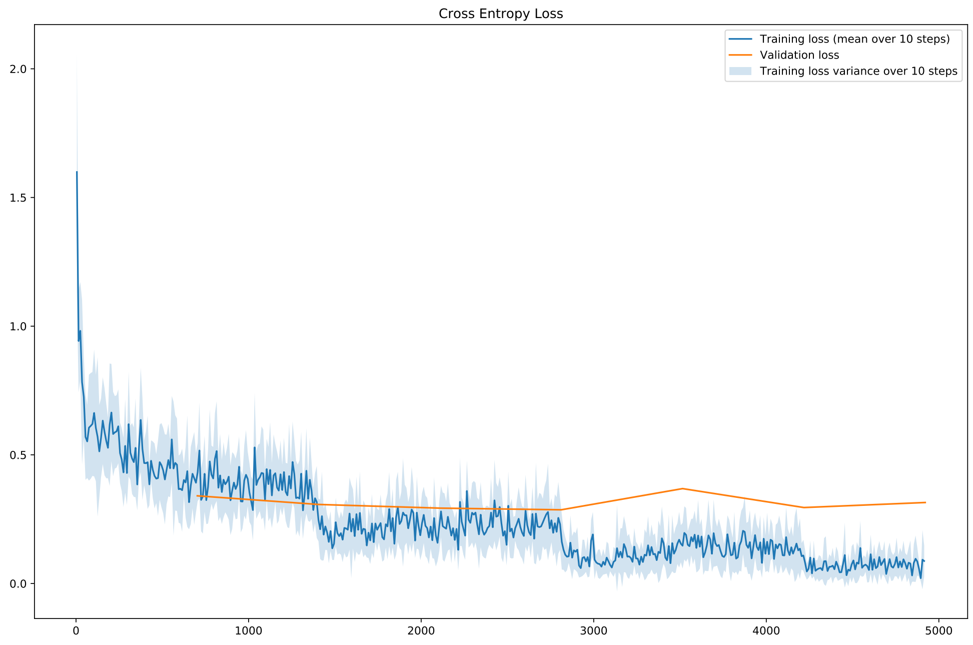

Figure 4: *Training and validation loss over training.*

Results from transfer learning with Resnet18 for the CIFAR-10 dataset is shown in Figure 4, with Adam optimizer, batch size of 32 and learning rate of $5 \cdot 10^{-5}$. Images in the dataset are resized to 224×224, and normalized with `mean:`$(0.485, 0.456, 0.406)$ and `std:`$(0.229, 0.224, 0.225)$. The final test accuracy with these parameters is 0.8727.

b)

From Figure 5, we see that each filter specializes in some feature of the original image. We see that e.g filter 14 activates on vertical lines and filter 26 activates on horizontal lines, and from the corresponding weights we see that both activate at the center of the image.

c)

The activations of the ten first filters from the last convolutional layer is shown in Figure 6, which now are more abstract compared to the output of the first layer in Figure 5. It is difficult to interpret these activations in any intuitive way, but this is what the network ultimately looks for in a zebra before making the classification.

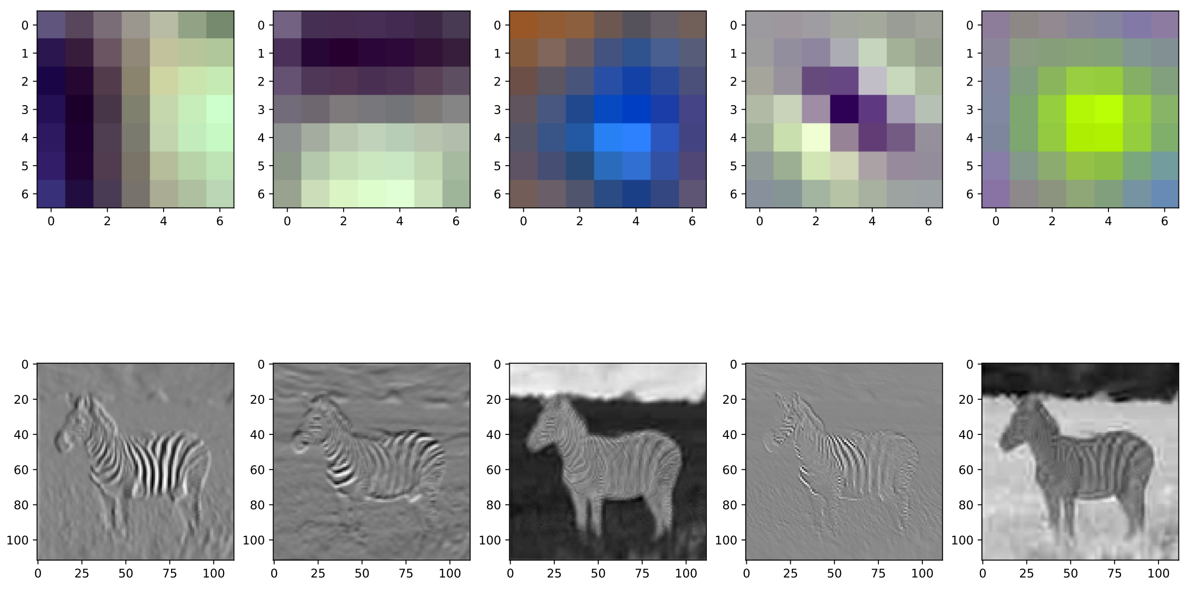

Figure 5: Filter weights (top row) and activation of corresponding filter (bottom row) of the first convolutional layer in ResNet18, with filter indices [14,26,32,49,52].

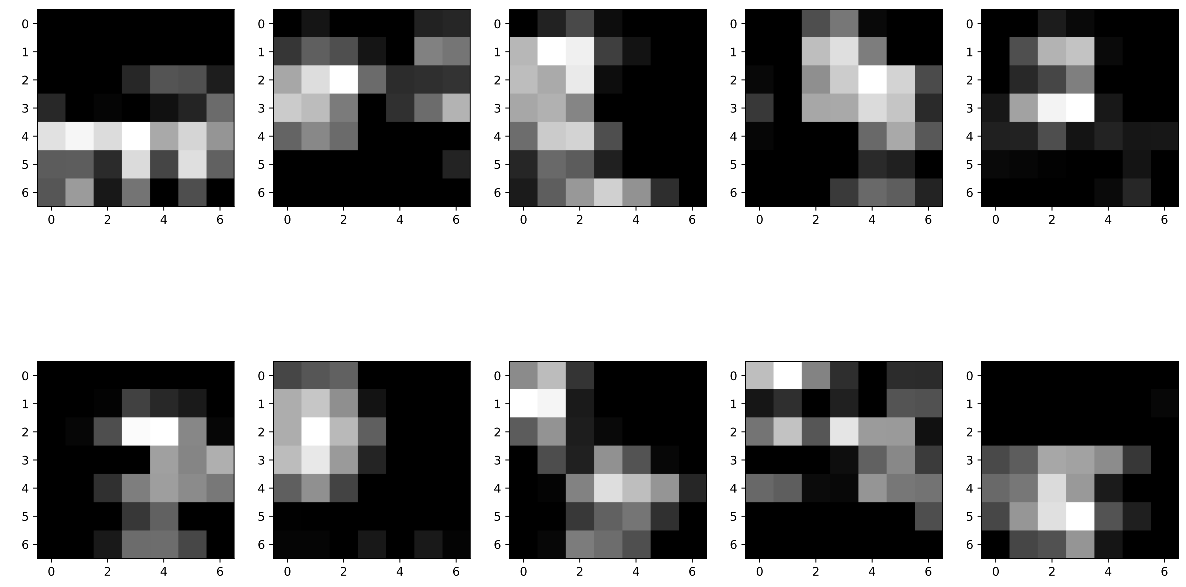

Figure 6: Activation of last convolutional layer in ResNet18, with filter indices 0-9, left to right, top to bottom.