TDT4265: Computer Vision and Deep Learning

Assignment 3

Espen Bragerhaug, Alfred Lieth Årøe

espentb@stud.ntnu.no, alfredla@stud.ntnu.no

Task 1: Theory

Task 1a:

We are given the image in Figure 1a and the kernel in Figure 1b and the task to perform a spatial convolution on the image I using the kernel K by hand.

3	0	1	7	3
6	4	0	4	5
4	6	3	2	4

Figure 1a: Image I

0	1	0
1	-4	1
0	1	0

Figure 1b: Kernel K

To handle the boundary conditions we choose to simply use a zero padding around the image, and this is the result after the convolution

-6	8	3	-20	0
-13	-4	12	-2	-9
-4	-13	-4	3	-9

Figure 1c: Image I after the convolution

Task 1b:

The Max Pooling layer reduces the CNNs sensitivity to transitional variations in the input. This is because the Max Pooling layer returns nearly identical outputs from two inputs that only differ in a small transitional shift.

Task 1c:

If you want to make the output shape to be equal to the input image after a convolutional layer with a stride of 1, kernel size of 5x5 and 6 filters, you would need a padding of 2 on each side.

Task 1d:

The special dimensions of the first layers are 504x504 with 12 feature maps, the stride is one, no padding is used, and the kernels used are square and of an odd size. Since the input are RGB images of size 512x512, and we need to remove 8 spaces in height and width, the kernel size must be 9x9 so that 4 spaces are remove to the left, right, top and bottom.

Task 1e:

This effectively halves the size of the spatial dimensions. Since the size is 504x504 after the first convolutional layer, the spatial dimensions of the pooled feature maps are 252x252.

Task 1f:

A 3x3 filter with stride 1 removes one space around the border, so the spatial dimensions of the feature maps in the second layer are 250x250

Task 1g:

The calculations of the total number of parameters in the network

Input	$32 \cdot 32 \cdot 3$
Layer 1	$5 \cdot 5 \cdot 3 \cdot 32 + 32 = 2432$
Layer 2	$5 \cdot 5 \cdot 32 \cdot 64 + 64 = 51264$
Layer 3	$5 \cdot 5 \cdot 64 \cdot 128 + 128 = 204928$
Layer 4	$4 \cdot 4 \cdot 128 \cdot 64 + 64 = 131136$
Layer 5	$1 \cdot 64 \cdot 10 + 10 = 650$
Total parameters	$2432 + 51264 + 204928 + 131136 + 650 = 390410$

Task 2: Convolutional Neural Networks

Task 2a:

Here is a plot of the train and validation loss for the whole training period.

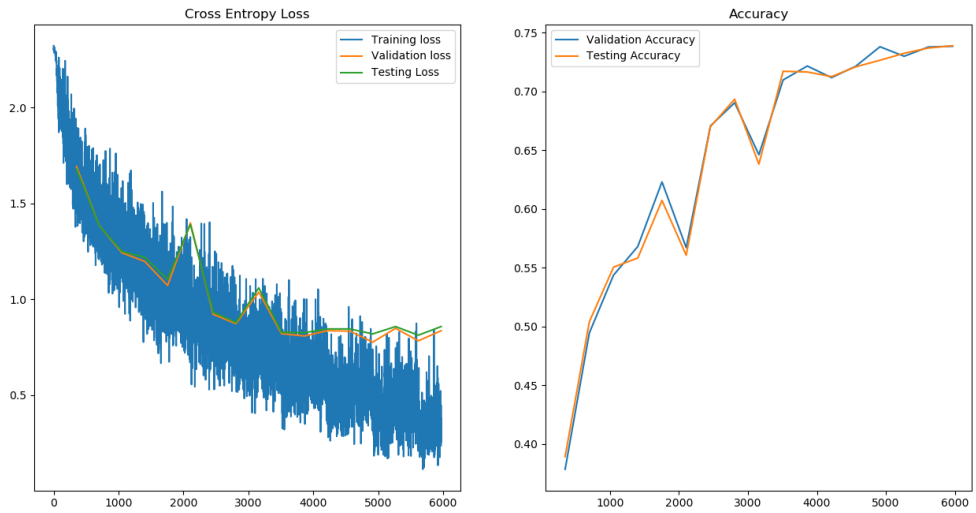

Figure 2: Entropy loss and acuracy graphs in task 2a

Task 2b:

Here is the final training, validation and test accuracy from the training done above and calculated over the whole train/val/test datasets:

Training_loss: 0.257
Training_acc: 0.915

Validation_loss: 0.834
Validation_accuracy: 0.740

Test_loss: 0.858
Test_accuracy: 0.739

Task 3: Deep Convolutional Network for Image Classification

Task 3a:

Model 1:

Model 1 in this task focuses on applying some depth to our model. Each convolutional layer has a filter size of 5×5 with a padding of 2 and a stride of 1. Each MaxPool2D layer has a stride of 2 and a kernel size of 2 × 2. We also applied batch normalization to train faster. The learning rate we used for this model was 0.05 and the batch size was 64.

Layer	Layer Type	Number of hidden units/ filters	Activation Function
1	Conv2D	32	ReLU
1	BatchNorm2D	-	-
2	Conv2D	32	ReLU
2	MaxPool2D	-	-
3	Conv2D	64	ReLU
3	BatchNorm2D	-	-
4	Conv2D	64	ReLU
4	MaxPool2D	-	-
5	Conv2D	128	ReLU
5	BatchNorm2D	-	-
6	Conv2D	128	ReLU
6	MaxPool2D	-	-
	Flatten		
7	Fully-Connected	512	ReLU
8	Fully-Connected	64	ReLU
9	Fully-Connected	10	Softmax

Model 2:

Model 2 in this task focuses on applying more number of filters to our model, as well as having smaller filters. Each convolutional layer has a filter size of 3×3 with a padding of 2 and a stride of 1. Each MaxPool2D layer has a stride of 2 and a kernel size of 2 × 2. The learning rate we used for this model was 0.05 and the batch size was 64.

Layer	Layer Type	Number of hidden units/ filters	Activation Function
1	Conv2D	64	ReLU
1	MaxPool2D	-	-
2	Conv2D	128	ReLU
2	MaxPool2D	-	-
3	Conv2D	256	ReLU
3	MaxPool2D	-	-
	Flatten		
4	Fully-Connected	512	ReLU
5	Fully-Connected	10	Softmax

Task 3b:

	Model 1	Model 2
Training Loss	0.254	0.132
Validation Loss	0.604	0.832
Test Loss	0.596	0.862
Training Accuracy	0.913	0.960
Validation Accuracy	0.805	0.767
Test Accuracy	0.810	0.763

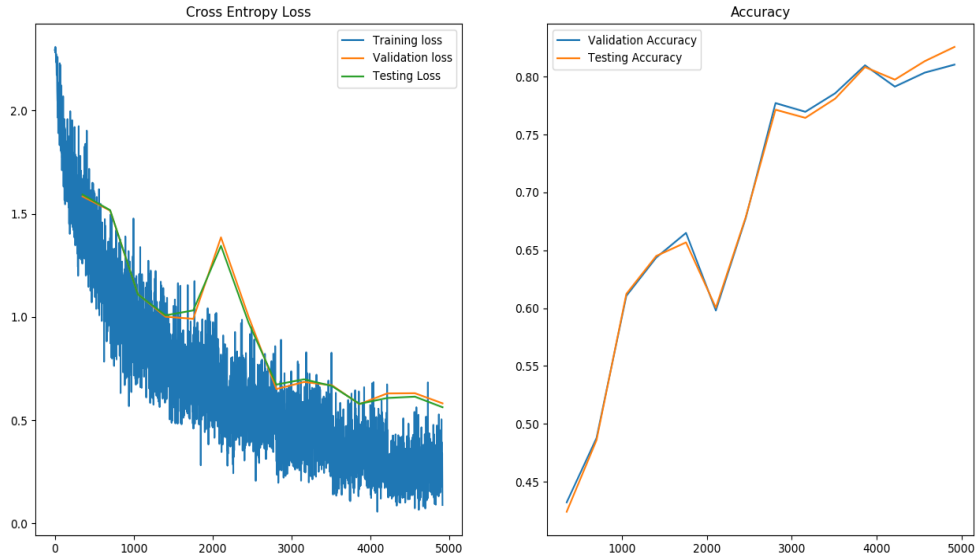

Figure 3: To the left we see the loss, and to the right we see the accuracy. It is clear that applying some depth to our model is very beneficial.

Task 3c:

We tried some other activation functions, but did not find any that were an improvement to ReLU. We also tried the Adam-optimizer and the RMSprop-optimizer. These did not improve our networks. We also applied some L2 regularization, but this did not improve our network either. Maybe because we do not train long enough that complex networks are an issue.

The things that had the greatest effect on our network was to make it deeper. By applying twice as many layers we got a test accuracy of a little over 80%. It seems that the deepness of the network is important to be able to generalize well on a class. We also had some success with making the amount of filters higher, but when we did this the training accuracy became really high, while the test accuracy only had a little improvement. It seems that when we use a high amount of filters we tend to overfit on the data.

After applying a deeper network (doubling the amounts of layers with the same amount of filters at every step) we had a good improvement. The network also stopped much earlier. Maybe it would be even better if we increased the stop criterias. Here are the train and validation loss (and test loss) before and after deepening the model.

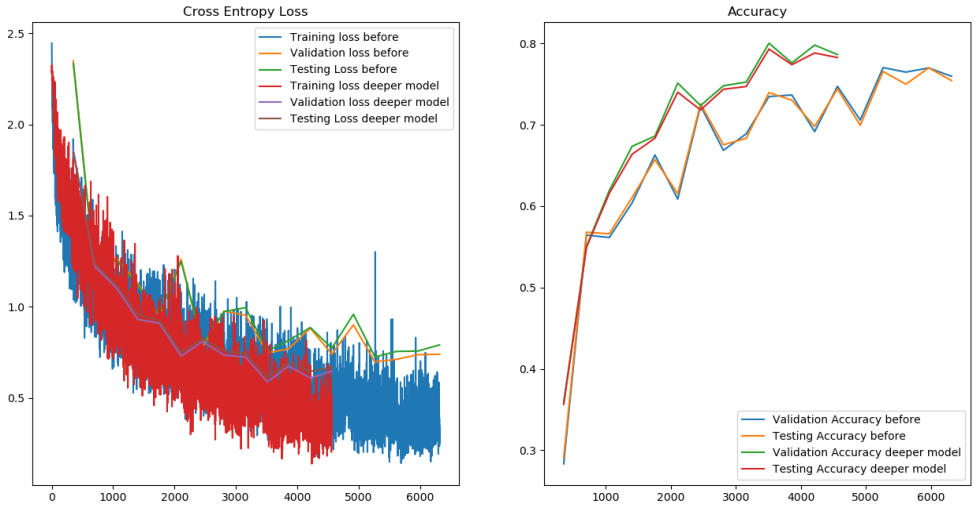

Figure 4: To the left we see the loss, and to the right we see the accuracy

Task 3d:

We applied even more depth to our model, and also increased the amount of filters at the final layers. This had a small improvement on the network, and seemed to be more consistent than before. We also continued to use batch normalization. The resulting plot were as follows:

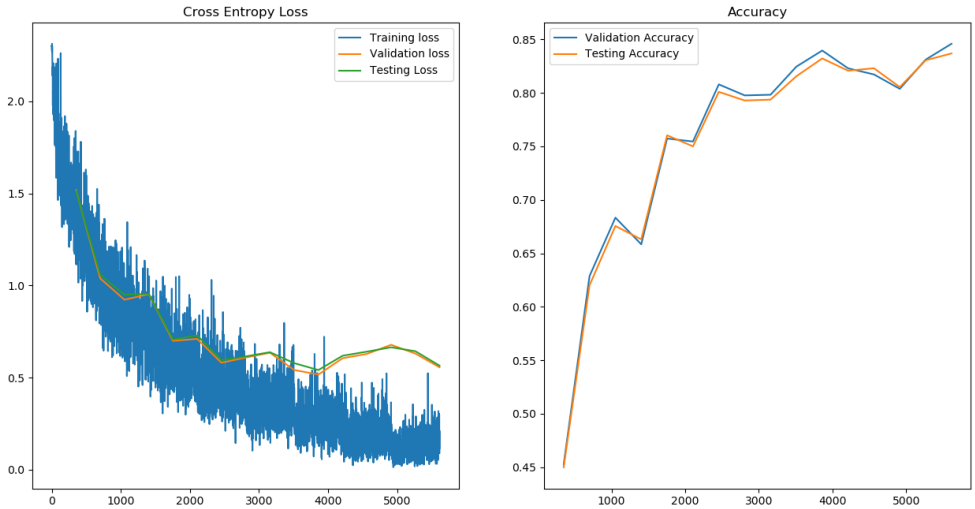

Figure 5: To the left we see the loss, and to the right we see the accuracy

Task 4: Transfer Learning with ResNet

Task 4a:

In the implementation of transfer learning with Restnet18 is in the file task4a.py. We used these hyperparameters:

Optimizer: Stochastich Gradient Descent
Batch size: 32
Learning rate: $5*10^{-4}$

No data augmentation is used. Here is the resulting accuracy and loss from the trained model.

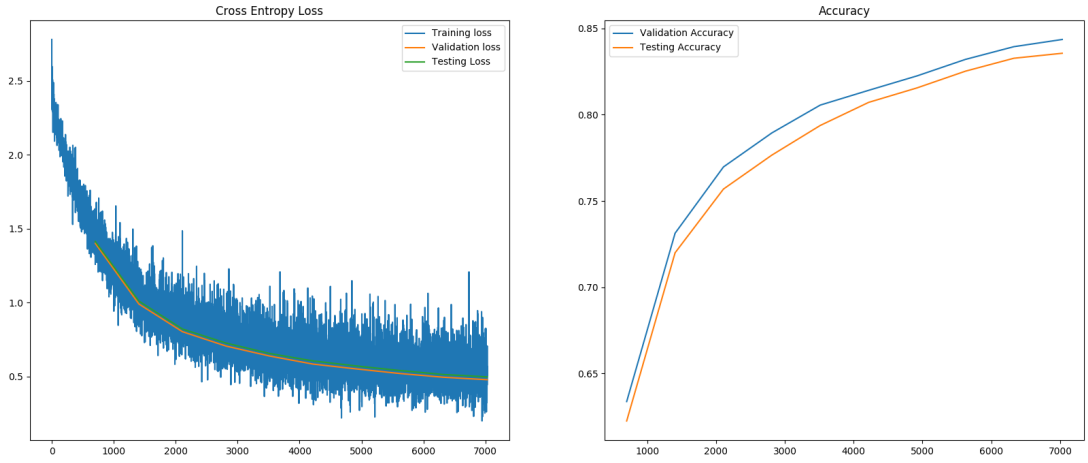

Figure 6: To the left we see the loss, and to the right we see the accuracy

Task 4b:

Here is the visualized filters for the indices [14, 26, 32, 49, 52] with the filter weights in the top row and the activation to corresponding filter below it in the bottom row.

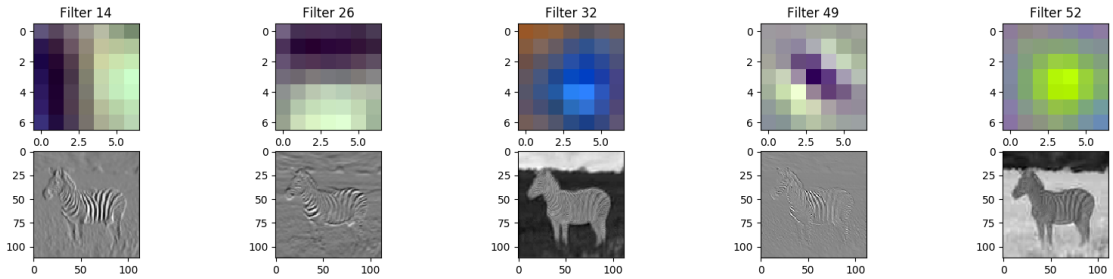

Figure 7: Visualization of some filters from the first conolutional layer in the trained model

All the filters seem to extract general features of the images. It looks like filter 14, 26 and 49 are extracting horizontal, vertical and diagonal edges respectively by looking at the stripes of the zebra in the activations of the filters. Filter 32 and 52 looks and behaves like color extracting filters for blue and green/yellow respectively, as the blue sky is white in filter 32 and the yellow grass is white in filter 52.

Task 4c:

Here is the visualization of the activations of the ten first filters from the last convolutional layer of the trained model.

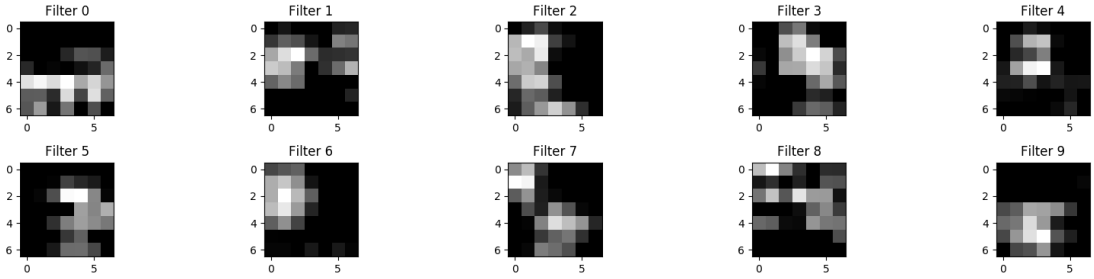

Figure 7: The zebra image passed through all the conolutional layers in the trained model

It is harder to see what the filters actually represent now that the image has passed through several convolutional layers and because of the fact that the image is way smaller in resolution then before. Our guess is that every filter is looking for some higher level feature, rather than the general features we see in task 4b.