NTNU
Norwegian University of
Science and Technology

TDT4265 - Computer Vision and Deep Learning

Assignment 3

Jakob Løver
Einar Henriksen

February 28, 2019

Contents

1 Convolutional Neural Networks

1.1 Task a

The hyperparameters that were used are shown in table 1.1.

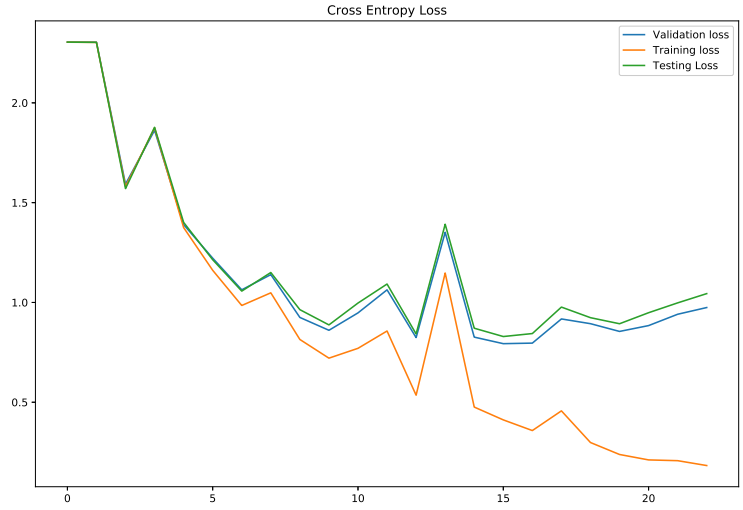

Figure 1.1: Loss when training the network given in the assignment

1.2 Task b

The training stopped early, and the test accuracy was 73.97%, the validation accuracy was 75.08%, and the training accuracy was 91.98%.

1.3 Task c

The number of parameters for a convolutional layer is calculated as ((kernel size)*input_channels+bias)*filters.

Epochs	100
Batch size	64
Learning rate	0.05
Early stop count	4

Table 1.1: Hyperparameters in task a

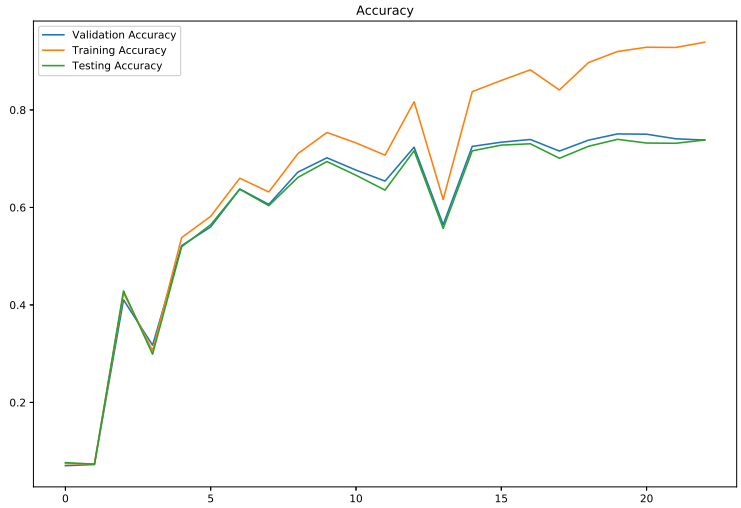

Figure 1.2: Accuracy for the network trained in task a

For our first layer that is ((5*5)*3+1)*32=2432 parameters.
The second layer has ((5*5)*32+1)*64=51264 parameters,
and the third one has ((5*5)*64+1)*128=204928 parameters.
The number of parameters in the fully connected layers are simply (inputs+bias)*outputs.
That means the first one has (128*4*4+1)*64=131136 and the last one has (64+1)*10=650 parameters.
This means the whole network has 390410 parameters.

2 Deep Convolutional Network for Image Classification

2.1 Task a

2.1.1 Network 1

This network was implemented with the hyperparameters listed in table 2.1, and with the architecture described in table 2.2. The dropout layer used $p = 0.5$, and the pooling layers used both a kernel size and stride of 2. Each convolutional layer uses a kernel size of 3 with a stride and padding of 1. We used one fully connected layer of 64 units with a LeakyReLU activation. The idea behind this was to prevent "dead neurons" in the fully connected layer.
We decided to use an exponentially decaying learning rate, with $\gamma = 0.95$. This allowed us to start learning with a fairly large learning rate. The weights in the network were initialized using Xavier initialization, with the SGD optimizer.

Epochs	10
Batch size	64
Initial learning rate	0.1
Early stop count	4

Table 2.1: Hyperparameters in task a for network 1

Layer Type	Number of Hidden Units	Activation Function
Conv2D	32	ReLU
BatchNorm2D	-	-
Conv2D	-	ReLU
BatchNorm2D	-	-
MaxPool2D	-	-
Dropout	-	-
Conv2D	64	ReLU
BatchNorm2D	-	-
Conv2D	-	ReLU
BatchNorm2D	-	-
MaxPool2D	-	-
Dropout	-	-
Conv2D	128	ReLU
BatchNorm2D	-	-
Conv2D	-	ReLU
BatchNorm2D	-	-
MaxPool2D	-	-
Dropout	-	-
Flatten	-	-
Dense	64	LeakyReLU
Dense	10	Softmax

Table 2.2: Network 1 architecture

2.1.2 Network 2

The second network has a few improvements over the previous. We used the same hyperparameters and optimizer as the previous network, but this network used kernel sizes of 5 with a padding of 2 for the convolutional layers. To change up the architecture, we dropped the fully connected layer with LeakyReLU activation. We also changed out the ReLU activation function with the ELU activation function, and each dropout layer has a 10% smaller chance of dropping a neuron than the previous dropout layer. The learn-

ing rate was also changed, where we went from an exponentially decaying learning rate to a learning rate that decays after the 3rd and 8th epoch by 50%.

2.2 Task b

Network	Train loss	Val loss	Train acc	Val acc	Test acc
1	0.5447	0.6389	80.65%	77.76%	77.16%
2	0.3819	0.5427	86.85%	81.48%	80.19%

Table 2.3: Performance comparison between the two networks

2.3 Task c

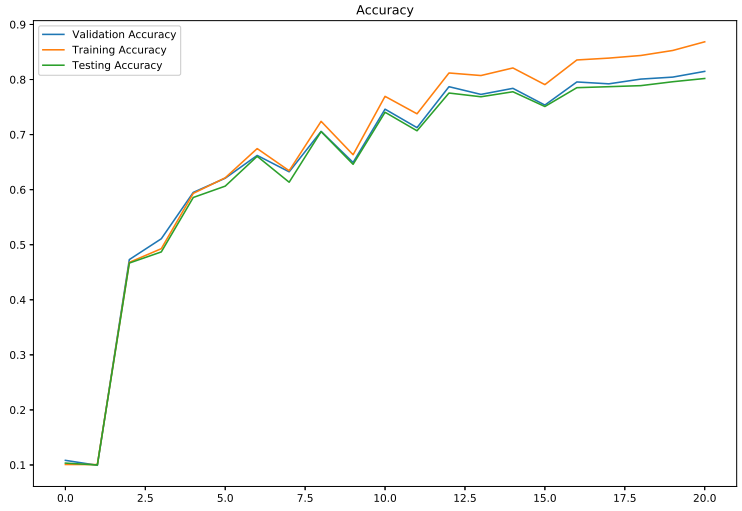

Figure 2.1: Accuracy for the second network

2.4 Task d

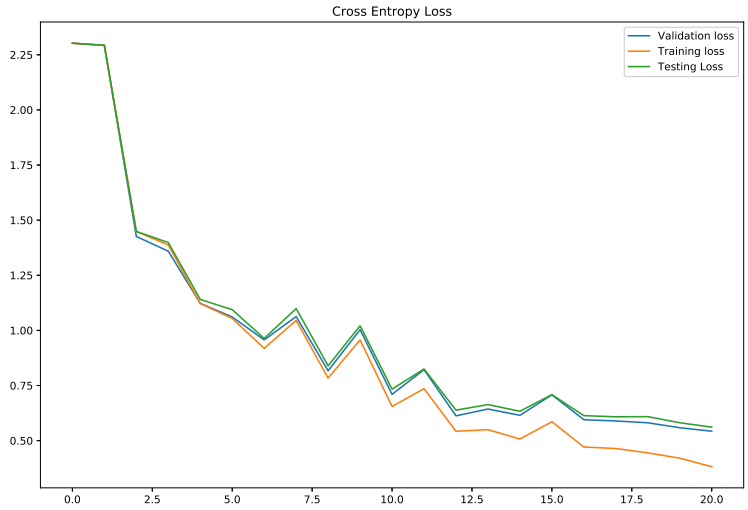

Figure 2.2: Loss for the second network

2.5 Task e

Among many things we tried out were:

- Batch normalization
- Dropout layers
- Strided convolutions vs pooling
- Adam optimizer
- Xavier initialization of weights
- Changing the batch size
- Exponentially decaying learning rate
- Multistep-decaying learning rate
- Varying amount of convolutional layers
- Changing the order of layers
- Varying amounts of fully connected layers
- ReLU activation function
- ELU activation function
- LeakyReLU activation function

We tried using batch normalization only (which helped greatly with overfitting),however, we still saw tendencies of overfitting. Introducing dropout layers after the activation functions helped remedy this, and the loss functions for both training and validation followed each other more closely. The best results of this were seen when introducing sequentially smaller chance of dropping a neuron for each dropout layer. This is because we allow the network to try out several different networks through randomly dropping some neurons. Strided convolutions definitely performed worse than using pooling layers. The accuracy significantly decreased, upwards of 10%. We don't know why this is the case, but strided convolutions may learn a different set of features that may not be suited for our architecture. Using an annealing learning rate was one of the things that helped the most. The network was quick to learn but often oscillated in loss towards the end of the ten epochs. The gradient may have been too large, such that we were "missing" the local minima we were trying to find.

3 Transfer Learning with ResNet

3.1 Task a

We used the Adam optimizer for this task, but we did not implement any data augmentation. The rest of the hyperparameters are shown in table 3.1.

3.2 Task b

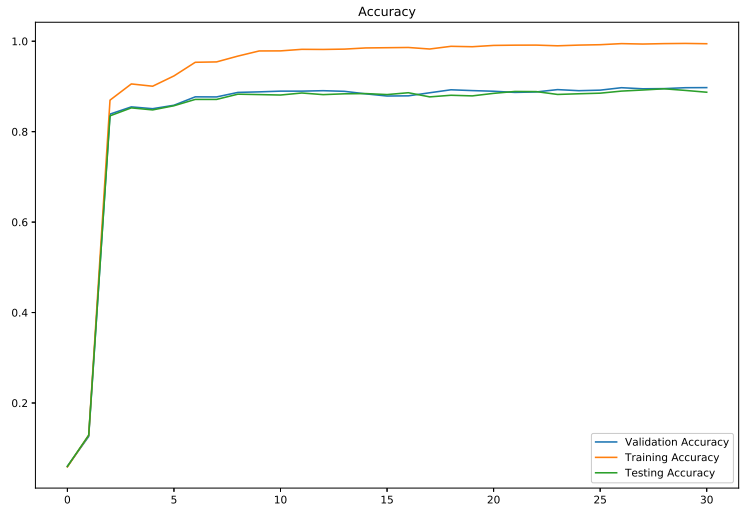

Figure 3.1: Accuracy using transfer-learning with Resnet18

Epochs	10
Batch size	32
Learning rate	0.0005
Early stop count	4

Table 3.1: Hyperparameters in task 3 a

3.3 Task c

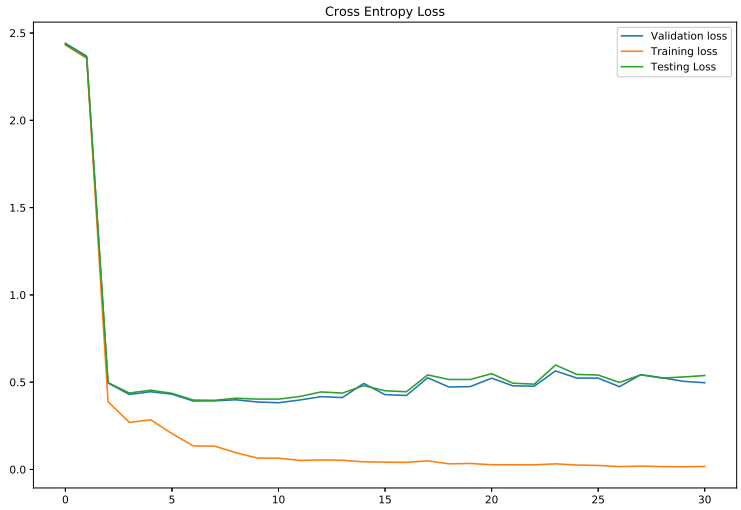

Figure 3.2: Training and validation loss when transfer-learning with Resnet18

3.4 Task d

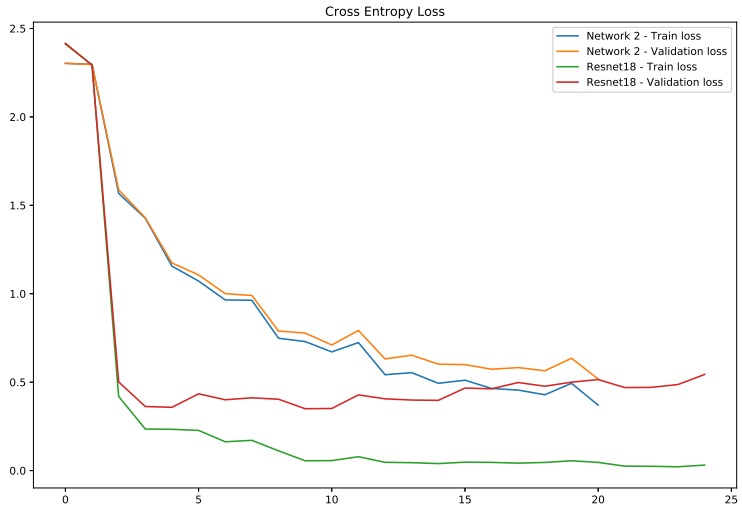

Figure 3.3: Comparison of loss

We observe that the fine tuning of the pre-trained network performs much better than our self-designed network, and converges much faster. It overfits after only a couple epochs. Resnet is pre-trained on the much larger dataset Imagenet, so this was expected.

3.5 Task e

Figure 3.4: Visualization of the output from the first convolutional layer in Resnet18

We see from the activations that the first convolutional layer recognizes simple lines and shapes. This is only for the first layer, the next convolutional layers will piece together this information to recognize more generalized shapes and objects.

3.6 Task f

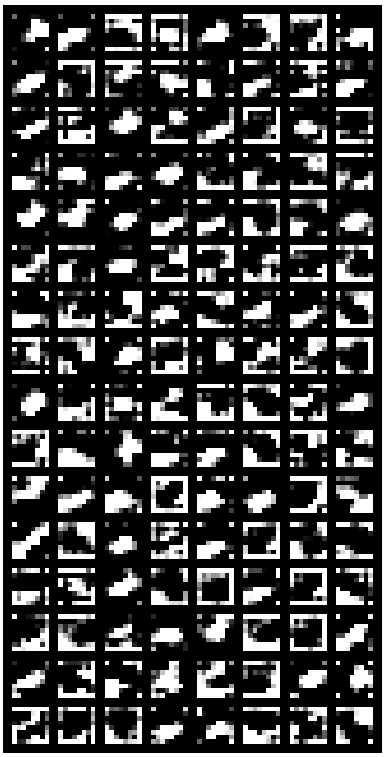

Figure 3.5: Visualization of 128 of filters in the last convolutional layer after transfer-learning with Resnet18

We observe the result from adding more filters to our convolutional layers. These recognize more abstract features than the initial convolutional layer.

Figure 3.6: Original image being passed through network

3.7 Task g

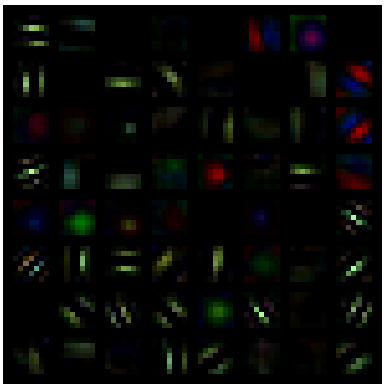

Figure 3.7: Visualization of the weights in the first convolutional layer in Resnet18

These filters recognize simple lines in our picture. We can see from the activations in figure 3.4 that for example the upper left filter recognizes horizontal lines, and the filter beneath it recognizes vertical lines.