AssignmentReport-Group1

March 5, 2021

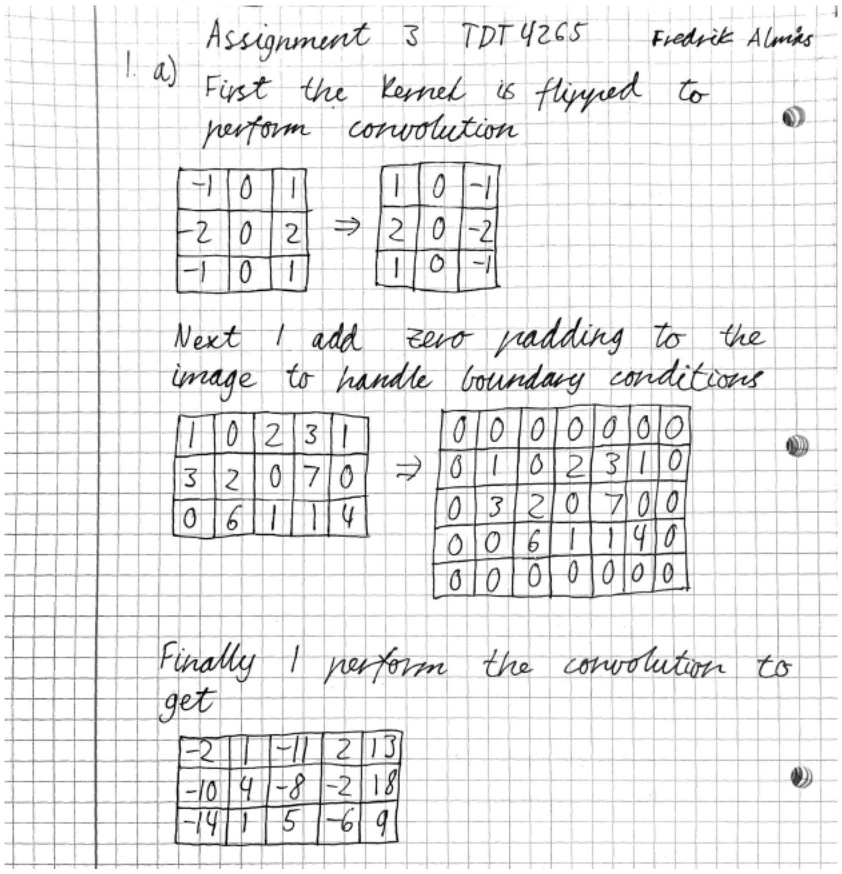
1 Assignment 3 Report

This is an outline for your report to ease the amount of work required to create your report. Jupyter notebook supports markdown, and I recommend you to check out this cheat sheet. If you are not familiar with markdown.

Before delivery, **remember to convert this file to PDF**. You can do it in two ways: 1. Print the webpage (ctrl+P or cmd+P) 2. Export with latex. This is somewhat more difficult, but you'll get somehwat of a "prettier" PDF. Go to File -> Download as -> PDF via LaTeX. You might have to install nbconvert and pandoc through conda; `conda install nbconvert pandoc`.

2 Task 1

2.1 task 1a)



Assignment 3 TDT 4265 Fredrik Almås

1. a) First the Kernel is flipped to perform convolution

-1	0	1
-2	0	2
-1	0	1

$\Rightarrow$

1	0	-1
2	0	-2
1	0	-1

Next I add zero padding to the image to handle boundary conditions

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

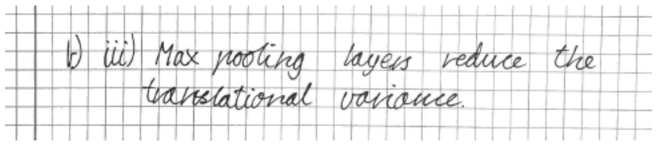
$\Rightarrow$

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

Finally I perform the convolution to get

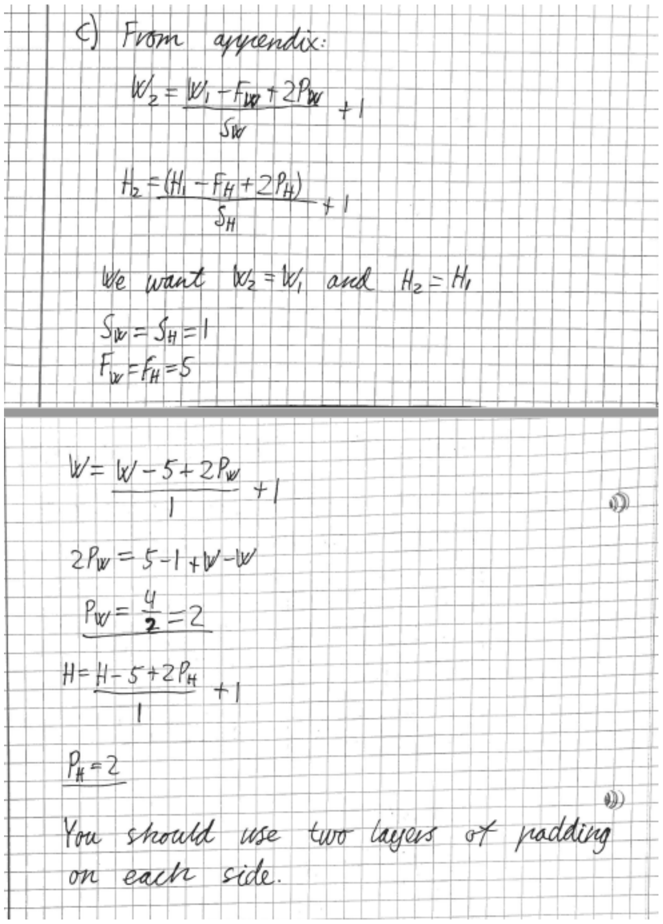
-2	1	-11	2	13
-10	4	-8	-2	18
-14	1	5	-6	9

2.2 task 1b)



b) (iii) Max pooling layers reduce the translational variance.

2.3 task 1c)


c) From appendix:

$$W_2 = \frac{W_1 - F_W + 2P_W}{S_W} + 1$$

$$H_2 = \frac{(H_1 - F_H + 2P_H)}{S_H} + 1$$

We want $W_2 = W_1$ and $H_2 = H_1$

$S_W = S_H = 1$

$F_W = F_H = 5$

$$W = \frac{W - 5 + 2P_W}{1} + 1$$

$2P_W = 5 - 1 + W - W$

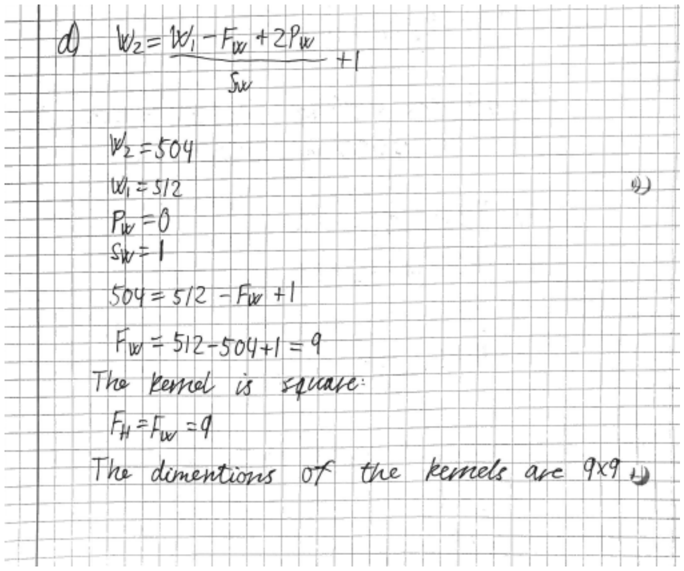
$\underline{P_W = \frac{4}{2} = 2}$

$$H = \frac{H - 5 + 2P_H}{1} + 1$$

$\underline{P_H = 2}$

You should use two layers of padding on each side.

2.4 task 1d)


d) $$W_2 = \frac{W_1 - F_W + 2P_W}{S_W} + 1$$

$W_2 = 504$

$W_1 = 512$

$P_W = 0$

$S_W = 1$

$504 = 512 - F_W + 1$

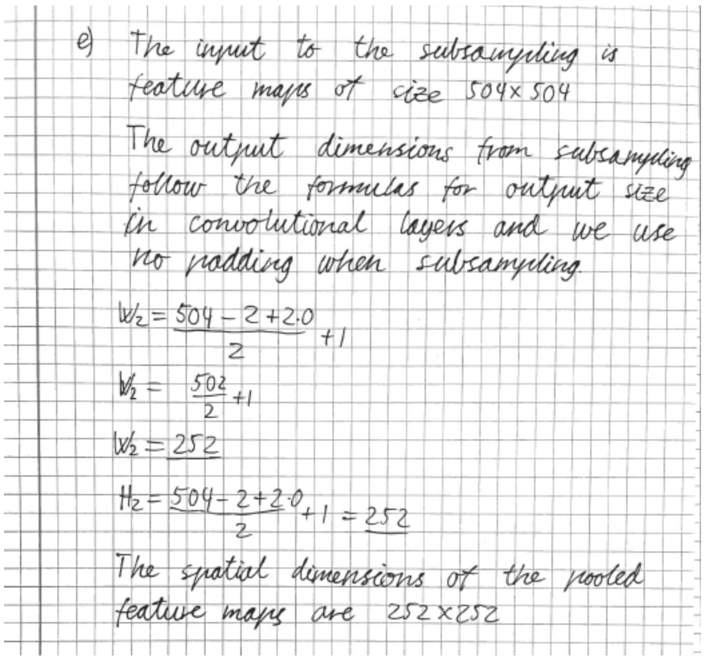
$F_W = 512 - 504 + 1 = 9$

The kernel is square:

$F_H = F_W = 9$

The dimentions of the kernels are 9x9

2.5 task 1e)



e) The input to the subsampling is feature maps of size 504×504

The output dimensions from subsampling follow the formulas for output size in convolutional layers and we use no padding when subsampling.

$W_2 = \frac{504-2+2\cdot 0}{2}+1$

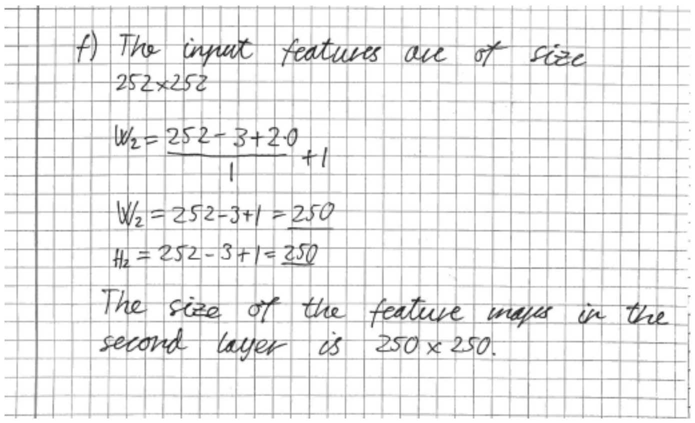
$W_2 = \frac{502}{2}+1$

$W_2 = \underline{252}$

$H_2 = \frac{504-2+2\cdot 0}{2}+1 = \underline{252}$

The spatial dimensions of the pooled feature maps are 252×252

2.6 task 1f)



f) The input features are of size 252×252

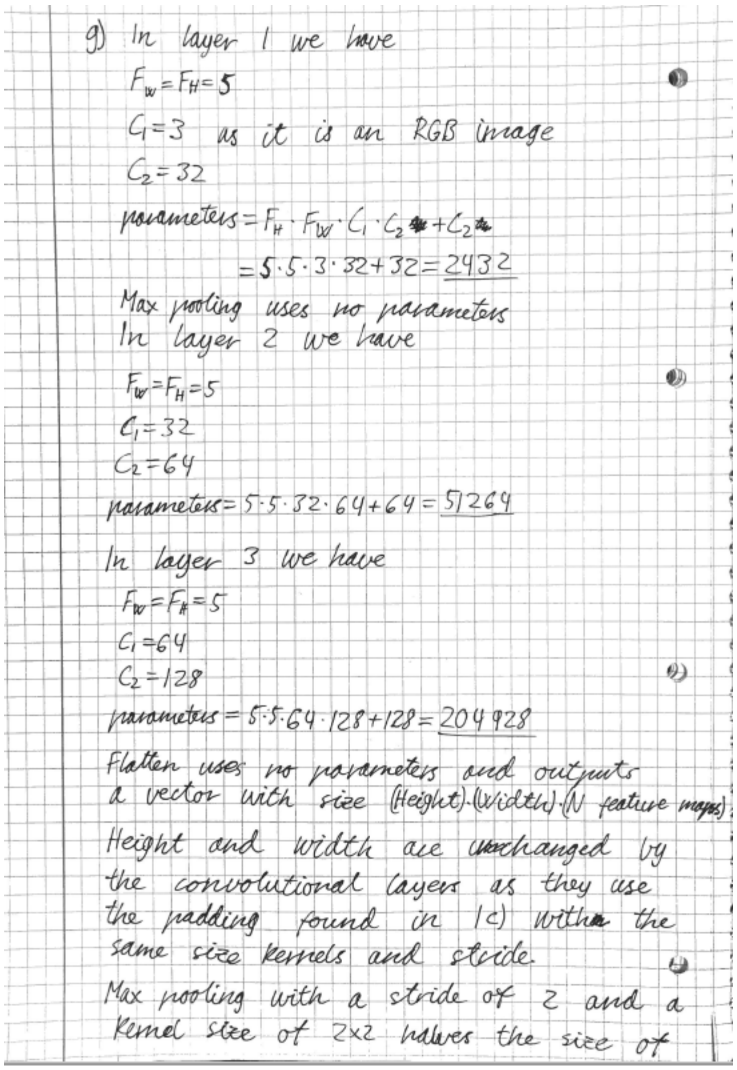
$W_2 = \frac{252-3+2\cdot 0}{1}+1$

$W_2 = 252-3+1 = \underline{250}$

$H_2 = 252-3+1 = \underline{250}$

The size of the feature maps in the second layer is 250 × 250.

2.7 task 1g)



g) In layer 1 we have

$F_W = F_H = 5$

$C_1 = 3$ as it is an RGB image

$C_2 = 32$

parameters $= F_H \cdot F_W \cdot C_1 \cdot C_2 + C_2$

$= 5 \cdot 5 \cdot 3 \cdot 32 + 32 = \underline{2432}$

Max pooling uses no parameters

In layer 2 we have

$F_W = F_H = 5$

$C_1 = 32$

$C_2 = 64$

parameters $= 5 \cdot 5 \cdot 32 \cdot 64 + 64 = \underline{51264}$

In layer 3 we have

$F_W = F_H = 5$

$C_1 = 64$

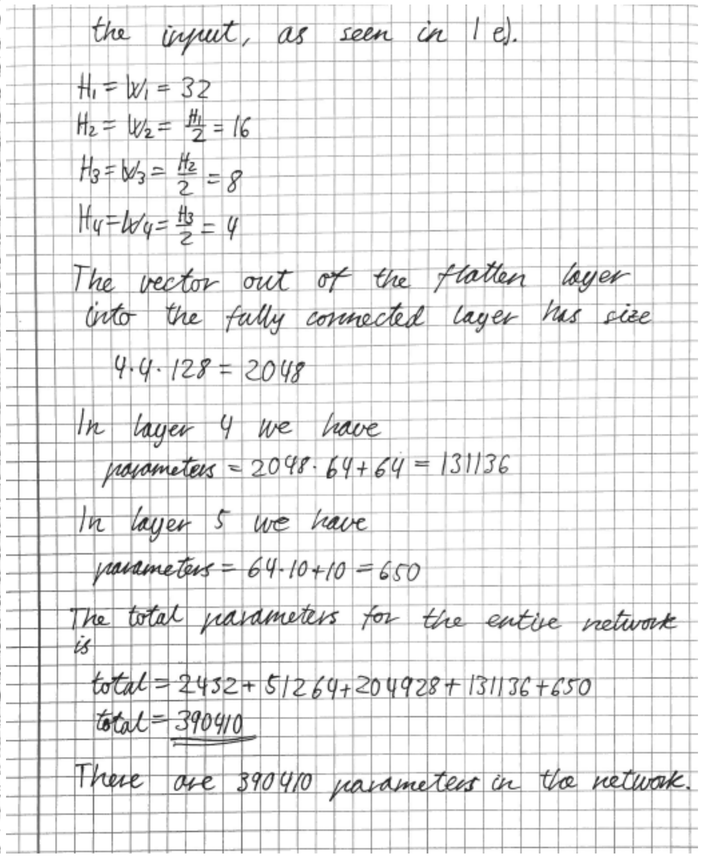
$C_2 = 128$

parameters $= 5 \cdot 5 \cdot 64 \cdot 128 + 128 = \underline{204928}$

Flatten uses no parameters and outputs a vector with size (Height)·(Width)·(N feature maps).

Height and width are unchanged by the convolutional layers as they use the padding found in 1c) with the same size kernels and stride.

Max pooling with a stride of 2 and a kernel size of 2x2 halves the size of



the input, as seen in 1e).

$H_1 = W_1 = 32$

$H_2 = W_2 = \frac{H_1}{2} = 16$

$H_3 = W_3 = \frac{H_2}{2} = 8$

$H_4 = W_4 = \frac{H_3}{2} = 4$

The vector out of the flatten layer into the fully connected layer has size

$4 \cdot 4 \cdot 128 = 2048$

In layer 4 we have

parameters $= 2048 \cdot 64 + 64 = 131136$

In layer 5 we have

parameters $= 64 \cdot 10 + 10 = 650$

The total parameters for the entire network is

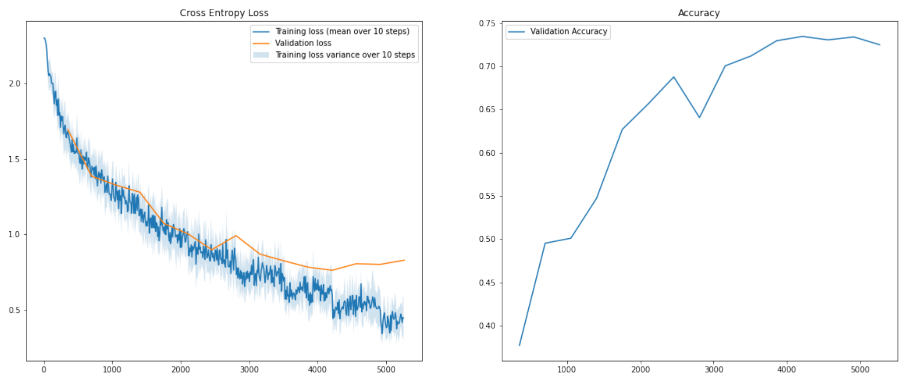
total $= 2432 + 51264 + 204928 + 131136 + 650$

total $= \underline{\underline{390410}}$

There are 390410 parameters in the network.

3 Task 2

3.0.1 Task 2a)



3.0.2 Task 2b)

The final training accuracy was 0.8353.
The final validation accuracy was 0.7344.
The final test accuracy was 0.7251.

4 Task 3

4.0.1 Task 3a)

For the first model I started by adding batch normalization. This increased the test accuracy, but not enough to get to 75% consistently. After that i tested different layer sizes and found that doubling the number of filters in the convolutional layers increased the accuracy further. Now the test accuracy was usually around 75% but not always. Finally i reduced the batch size to 32, which increased the accuracy further and gave a consistent test accuracy of 75%-77%.

The following table shows my finished first model, where i used SGD as optimizer, learning rate of 0.05, batch size of 32 and default weight initialization. The convolutional layers used a 5x5 kernel, a stride of 1 and 2 layers of padding. The maxpool layers used a 2x2 kernel with a stride of 2 and no padding.

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
1	Conv2d	64	ReLU
1	BatchNorm2d	64	-
1	MaxPool2d	-	-
2	Conv2d	128	ReLU
2	BatchNorm2d	128	-
2	MaxPool2d	-	-
3	Conv2d	256	ReLU

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
3	BatchNorm2d	256	ReLU
3	MaxPool2d	-	-
———	———	———	———
	Flatten		
4	Fully-Connected	64	ReLU
4	BatchNorm1d	64	-
5	Fully-Connected	10	Softmax

For the second model I started by switching to the Adam optimizer. To get the new optimizer to converge I also had to reduce the learning rate to 0.001. Next I tested some different network architectures and ended up using a structure with two convolutional layers for each pooling layer. The model was not reaching satisfactory accuracy and it stopped very early, usually after 4 epochs. To avoid finding local minimas which makes it stop early I added dropout after every layer, but figured out the model performed better when I only had dropout after each convolutional layer and not in the fully connected layer. The dropout worked better when it had a low probability parameter, where the best value was 10%. The model was not performing well enough yet, with a test accuracy of 72%. The next change was reducing the kernel size of the convolutional layers to 3, which resulted in a test accuracy of 75% consistently. Finally I added batch normalization to see if it had as much impact as in model 1, and it increased the accuracy to 79%.

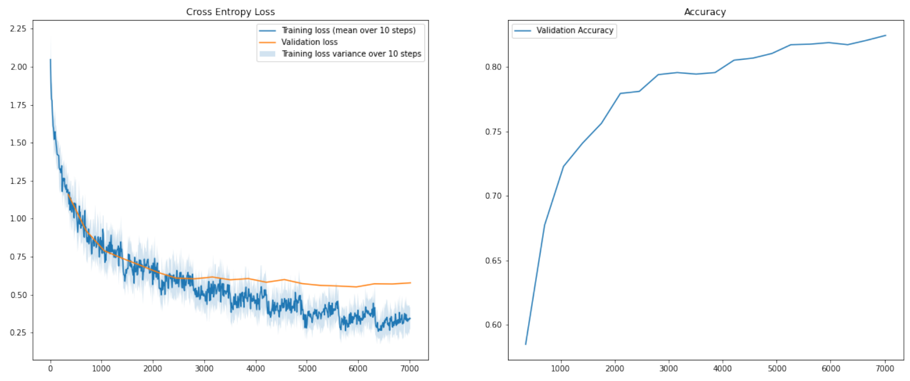
The following table shows my finished second model, where i used Adam as optimizer, learning rate of 0.001, batch size of 64 and default weight initialization. The convolutional layers used a 3x3 kernel, a stride of 1 and 1 layer of padding. The maxpool layers used a 2x2 kernel with a stride of 2 and no padding. The dropout layers used a probability of 10%.

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
1	Conv2d	32	ReLU
1	Conv2d	32	ReLU
1	BatchNorm2d	32	-
1	MaxPool2d	-	-
1	Dropout	-	-
2	Conv2d	64	ReLU
2	Conv2d	64	ReLU
1	BatchNorm2d	64	-
2	MaxPool2d	-	-
2	Dropout	-	-
3	Conv2d	128	ReLU
3	Conv2d	128	ReLU
1	BatchNorm2d	128	-
3	MaxPool2d	-	-
3	Dropout	-	-
———	———	———	———

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
	Flatten		
4	Fully-Connected	64	ReLU
5	Fully-Connected	10	Softmax

4.0.2 Task 3b)

Model	Training loss	Training accuracy	Validation accuracy	Test accuracy
Model 1	0.4064	0.8647	0.7554	0.7522
Model 2	0.3024	0.8935	0.7954	0.7907



4.0.3 Task 3c)

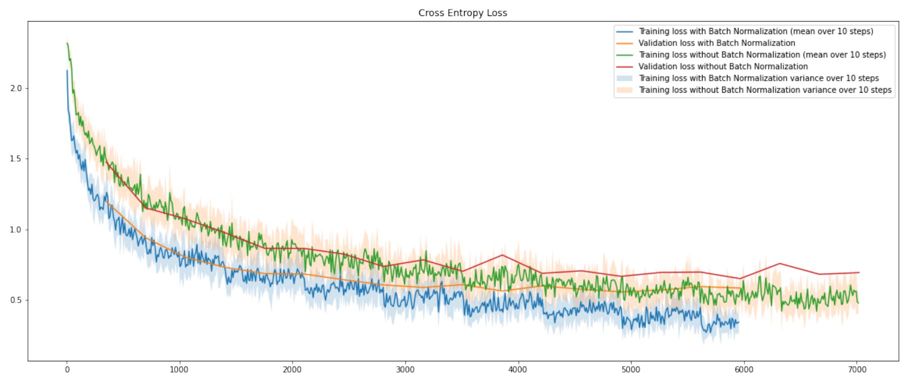
I saw improvements with adding more nodes per layer, using batch normalization and using dropout. More nodes resulting in an improvement suggests that there were details in the data not being learned by the model from task 2, the model might have been underfitted, and the extra complexity from adding more nodes fixed this. The method which resulted in the greatest improvement was definitely batch normalization. It resulted in faster and more stable learning in a way no other method could. I am not entirely sure why, but it could be due to the normalization reducing the randomness introduced into the model from weight initialization. Dropout improved the model because it reduced overfitting drastically and reduced the risk of stopping too early due to local minima.

Changes which did not improve the model was reducing the amount of nodes per layer and adding extra layers. As noted the model from task 2 showed signs of underfitting which means reducing complexity by reducing the amount of nodes would introduce even more underfitting. Starting out I thought adding extra layers would introduce the necessary model complexity, however the models became worse when i tried it. An explanation could be that the feature maps became too small to conserve the important information, as they would only have a size of 2x2 with an extra convolutional and maxpool layer. Adding another fully connected layer also reduced the model performance, which suggests that the underfitting was happening in the feature extractor and not in the classifier. Adding another fully connected layer in the classifier might have introduced overfitting in the classifier to reduce performance.

4.0.4 Task 3d)

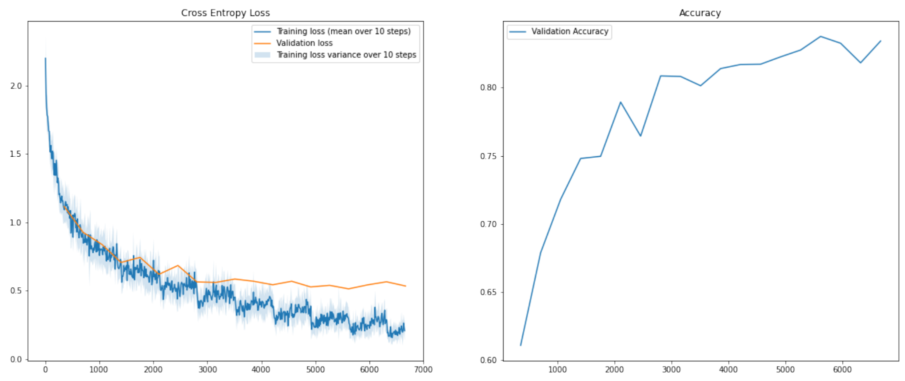
The method i saw the largest amount of improvement with was batch normalization.

Model 2 before and after batch normalization:



4.0.5 Task 3e)

Implemented in Task3_bestmodel.ipynb



The final test accuracy was 0.8117.

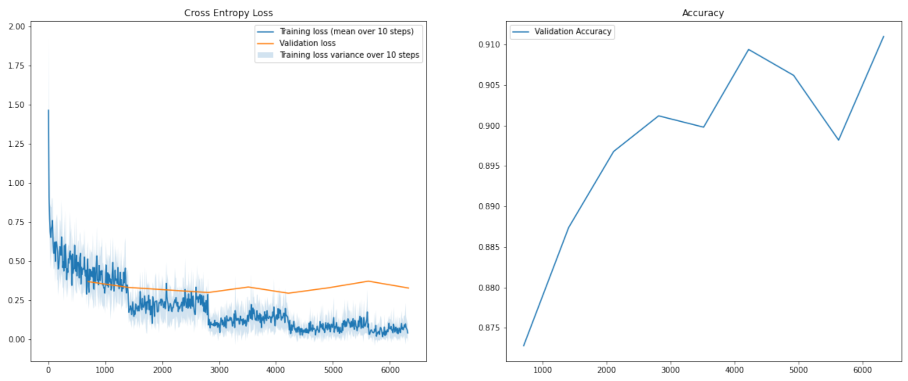
4.0.6 Task 3f)

The final model shows signs of overfitting as we can clearly see the validation loss stagnating while the training loss keeps reducing. I tried reducing the overfitting by increasing the dropout probability, which made the validation loss follow the training loss longer, but that also reduced the test accuracy thus making a worse model.

5 Task 4

5.1 Task 4a)

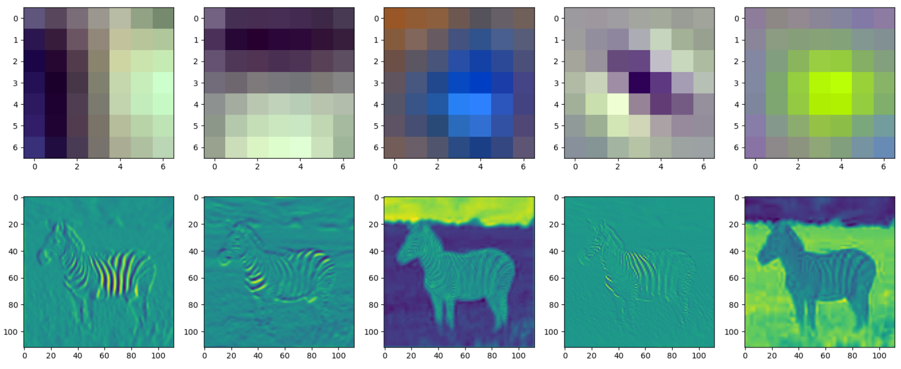
Implemented in Task4_transfer_learning.ipynb I used the Adam optimizer, a batch size of 32, learning rate of $5 * 10^{-4}$ and the only new augmentation was resizing the images and using the different mean and standard deviation found on https://s.ntnu.no/pytorch-ImageNet-normalization.



The final test accuracy with transfer learning was 0.8872.

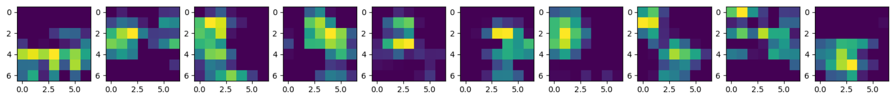
5.2 Task 4b)

The first filter on index 14 looks like a vertical edge detector, the second on index 26 looks like a horizontal edge detector and the fourth on index 49 looks like a diagonal edge detector. The remaining two filters visualized look like different variations of gaussian blurs with different focal points and colors.



5.3 Task 4c)

At this point the feature maps are unrecognizeable, but probably contain a lot of information needed for the classification module.


0
2
4
6
0.0
2.5
5.0

```
[ ]:
```