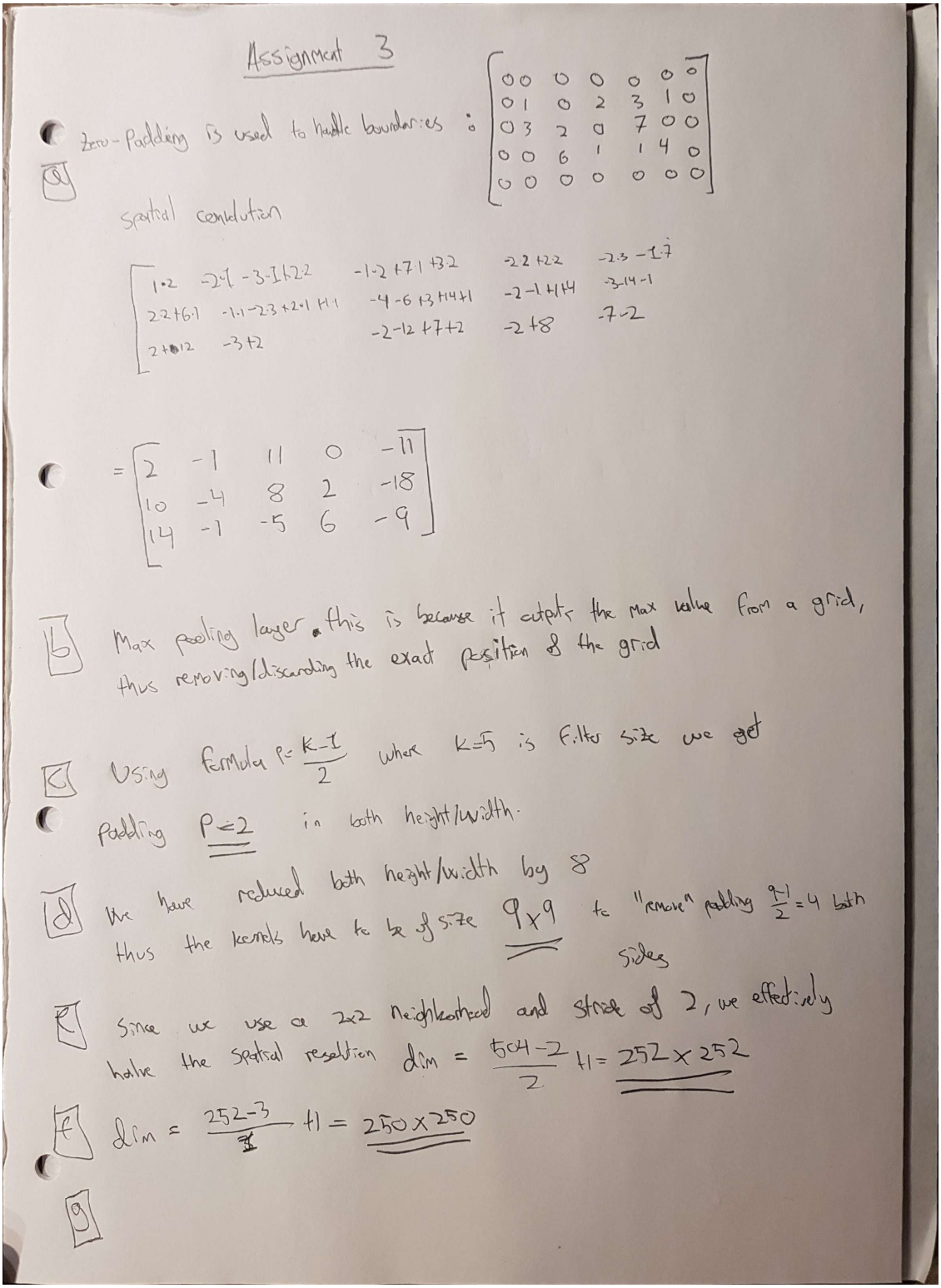
AssignmentReport-Group157

March 5, 2021

1 Assignment 1 Report

2 Task 1



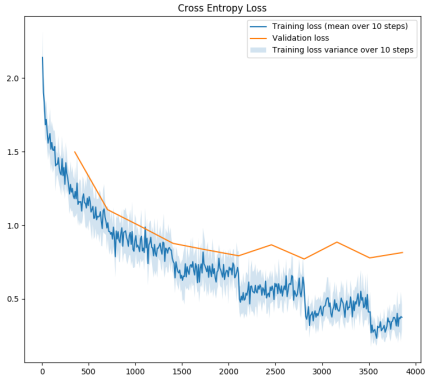
2.1 problem 1g

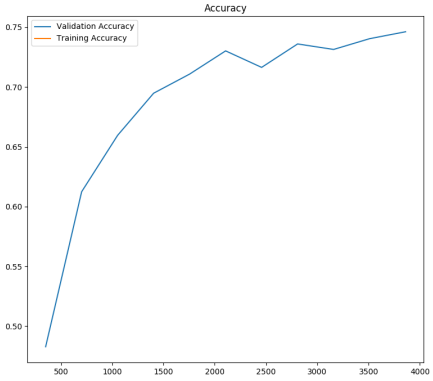
total amount of parameters equals: (5x5x3 + 1)x32 + (5x5x32 + 1)x64 + (5x5x64 + 1)x128 + (2048 + 1)x64 + (64+1)x10 = 390410

3 Task 2

3.0.1 Task 2ab)

using compute_loss_and_accuracy
Epochs: 8,
Validation Loss: 0.94
Validation Accuracy: 0.729
test loss: 0.9542929123227413
test accuracy: 0.7184





4 Task 3

4.0.1 Task 3b)

Network	1
Training loss	0.304
Training accuracy	0.940
Validation accuracy	0.785
Test accuracy	0.755

4.0.2 Task 3c)

One of the biggest problems for large neural nets with an abundance of parameters can be overfitting. In our first model, we tried to use dropout to address this problem. The idea is to randomly

drop units with a probablility of p=0.1, which effects regularization.

Batch normalization is also used to help prevent overfitting. It applies a transformation to maintain the activation mean=0, std=1. This to help address the problem of internal coviarate shift as well as regularization. Batch normalization seemed to be working very effectively at improving the accuracy of the models, perhaps because it gives a resistance to vanishing gradient problem as well as regularization.

Interestingly, the dropout seemed to barely have any positive effect on the test set. After some digging online, it seems according to 1 "Since convolutional layers have few parameters, they need less regularization to begin with. Furthermore, because of the spatial relationships encoded in feature maps, activations can become highly correlated. This renders dropout ineffective." The article proposes using dropout only on the fully-connected layers.

Model 1 The goal with this model was to experiment on Batch normalization, dropout, filter size and optimizers.

This model took use of batch normalization and dropout as seen in the table below, and reduced filter size to 3x3

The reduction of filter size seemed to improve the model by around ~0.007. This suggests that the most useful features in the images are the smaller features rather than the slightly larger ones. Reducing filter size also greatly improved training speed.

After 9 epochs it reaches Validation Loss: 0.71, Validation Accuracy: 0.785, Test Loss: 0.829 Test Accuracy: 0.755 For 5x5 filter size, we got the following Test Accuracy: 0.748 For adam optimizer, we got the following Test Accuracy: 0.7446

Both SGD and adam optimization techniques were tested. Adam was expected to perform better, but in actuality performs slightly worse. The Adam optimizer "combines the good properties of Adadelta and RMSprop and hence tend to do better for most of the problems" 2. The reason for why the SGD works better is not trivial. SGD tends to work better on passing local minima and since we are only running for up to 10 epochs, perhaps the Adam optimizer is stuck early in a local minima. One would assume that Adam would converge to a higher ceiling faster than SGD if given enough training. It could also be a problem of parameter tuning. Learning rate for the adam optimizer was chosen as lr=5e-3 as this gave quite good results.

Layer	Layer Type	Num Filters	Activation Function
1	Conv2d	in=3, out=32	ReLU
1	BatchNorm2d	32 features	
1	MaxPool2d	stride=2	
1	Dropout	p=0.1	
2	Conv2D	in=64, out=128	ReLU
2	BatchNorm2d	128 features	
2	MaxPool2d	stride=2	
2	Dropout	p=0.1	
3	Conv2D	in=128, out=256	ReLU
3	BatchNorm2d	256 features	
3	MaxPool2d	stride=2	
3	Dropout	p=0.1	

Layer	Layer Type	Num Filters	Activation Function
4	Linear	2048, 64	ReLU
4	Linear	64, 10	

Table	Training details
activation function	ReLU
optimizer	SGD and Adam
regularization	dropout
learning rate	5e-2 and 5e-3
batch size	64

For plot, see exercise 3d.

Model 2 The goal with this model was to experiment on Network architecture, activation functions and number of filters. The convolutional layers of the filter was increased and use respectively 64,128,256 filters. Chosing network architecture can be a bit of a "black art". In the second model, another convolutional layer was added to see if a deeper network would better the results.

The new activation function Mish gave better performance. It's implementation can be found in the source code. The reason for choosing Mish is simply that it is used in a lot of SOTA architectures like YOLOv4.

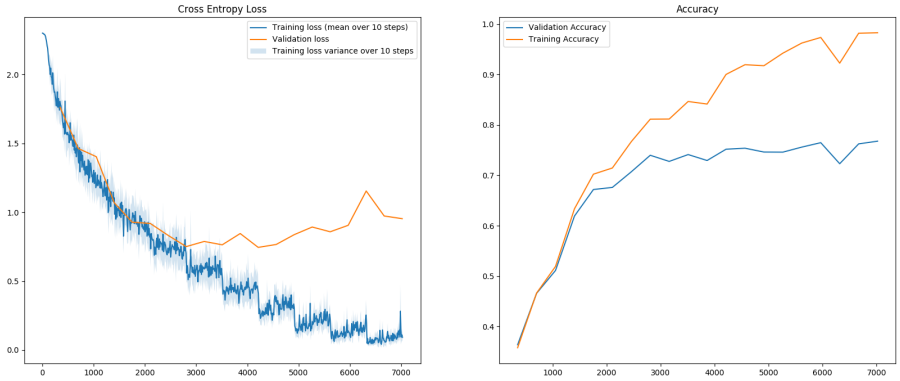
For the FC-layer, 3 showed that in order to obtain better performance, shallow CNNs require more nodes in FC layers. We have therefore added another linear layer in the classifier, and increased amount of neurons, as seen in the table below.

Epoch: 9, Batches per seconds: 55.71, Global step: 7020, Validation Loss: 1.04, Validation Accuracy: 0.757 1.0491873631492639 0.7587

Layer	Layer Type	Num Filters	Activation Function
1	Conv2d	in=3, out=32	Mish
1	BatchNorm2d	32 features	
1	MaxPool2d	stride=2	
1	Dropout	p=0.1	
2	Conv2D	in=64, out=128	Mish
2	BatchNorm2d	128 features	
2	MaxPool2d	stride=2	
2	Dropout	p=0.1	
3	Conv2D	in=128, out=256	Mish
3	BatchNorm2d	256 features	
3	MaxPool2d	stride=2	
3	Dropout	p=0.1	
4	Conv2D	in=128, out=256	Mish
4	BatchNorm2d	256 features	
4	MaxPool2d	stride=2	

Layer	Layer Type	Num Filters	Activation Function
4	Dropout	p=0.1	
5	Linear	2048, 1024	ReLU
5	Linear	1024, 512	ReLU
5	Linear	512, 10	ReLU

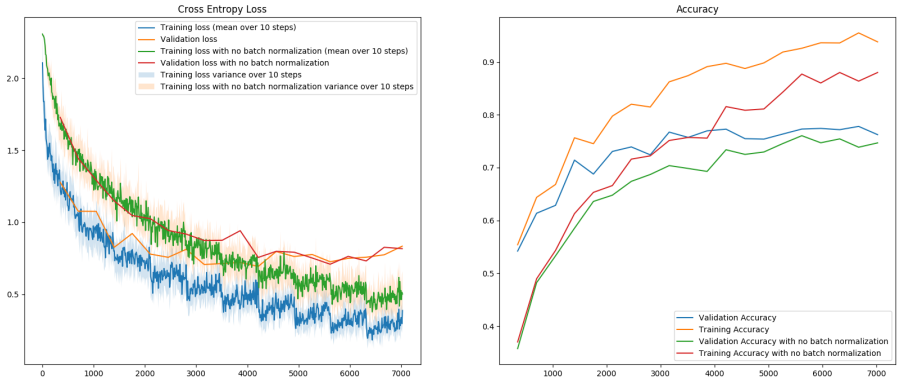
Table	Training details
activation function	ReLU
optimizer	SGD
regularization	dropout
learning rate	5e-2
batch size	64



4.0.3 Task 3d)

For the method/technique that you saw the largest amount of improvement on, include aplot of the train and validation loss before/after applying this technique (Similar to what you didin task 3 for assignment 2). Remember to include this in the same graph!

Batch normalization improved the model quite significantly. This is because it does a number of things for the network, as discussed earlier in the assignment. Below is teh graph for model 1 along with a plot of the same model without batch normalization



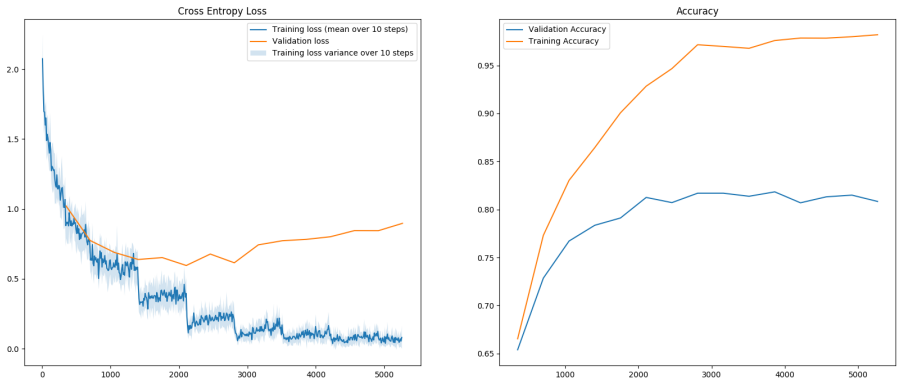
4.0.4 Task 3e)

Improve on your best model and reach an accuracy of 80% on the test set within 10 epochs.Include a plot of validation accuracy over training, and report your final test accuracy

Final Model For the final model, a network architecture similar to the first model was chosen. In addition, an extra convolution is added to each layer as this improved the accuracy even more. 3x3 kernel size was used.

Layer	Layer Type	Num Filters	Activation Function
1	Conv2d	in=3, out=32	
1	BatchNorm2d	32 features	
1			Mish
1	Conv2d	in=32, out=64	
1			Mish
1	MaxPool2d	stride=2	
2	Conv2D	in=64, out=128	
2	BatchNorm2d	128 features	
2			Mish
2	Conv2D	in=128, out=128	
2			Mish
2	MaxPool2d	stride=2	
3	Conv2D	in=128, out=256	
3	BatchNorm2d	128 features	
3			Mish
3	Conv2D	in=256, out=256	
3			Mish
3	MaxPool2d	stride=2	
4	Linear	2048, 1024	Mish
4	Linear	1024, 1024	Mish
4	Linear	1024, 10	Mish

Table	Training details
activation function	Mish
optimizer	Adam
regularization	batch normalization
learning rate	5e-4
batch size	64



4.0.5 Task 3f)

The model does show signs of overfitting, training accuracy is way higher than test accuracy. Adding more regularization like L2 would probably help with this.

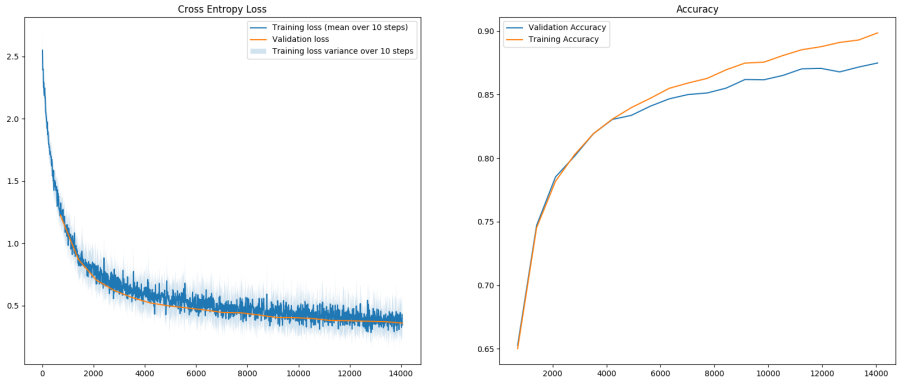
5 Task 4

5.1 Task 4a)

Used 112x112 transform and mean/std taken from https://s.ntnu.no/pytorch-ImageNet-normalization.

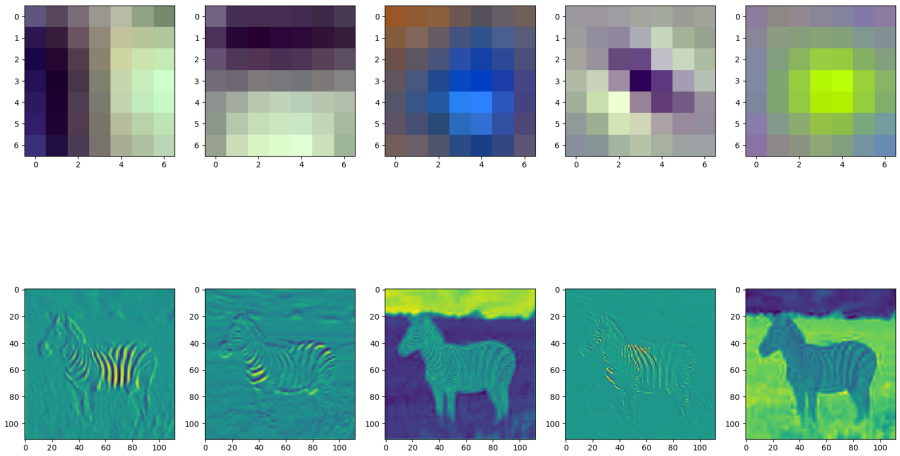
Table	Training details
optimizer	Adam
learning rate	5e-4
batch size	32

final test accuracy: 0.898



5.2 Task 4b)

For the first two filters it seems like the activations are on vertical and horizontal edges respectively. The 4th filter looks like it activates on diagonal edges, or at least some form of edge detection. For the 3rd and 5th filter it seems the activations perhaps comes from the color changes.



5.3 Task 4c)

After running through almost the whole network it is more difficult to see what the activations lay on. This may be more complex activations that are not readily understandable to us. We do see that the activations definetively are in the areas where the zebra is placed.