AssignmentReport-Group187

March 8, 2024

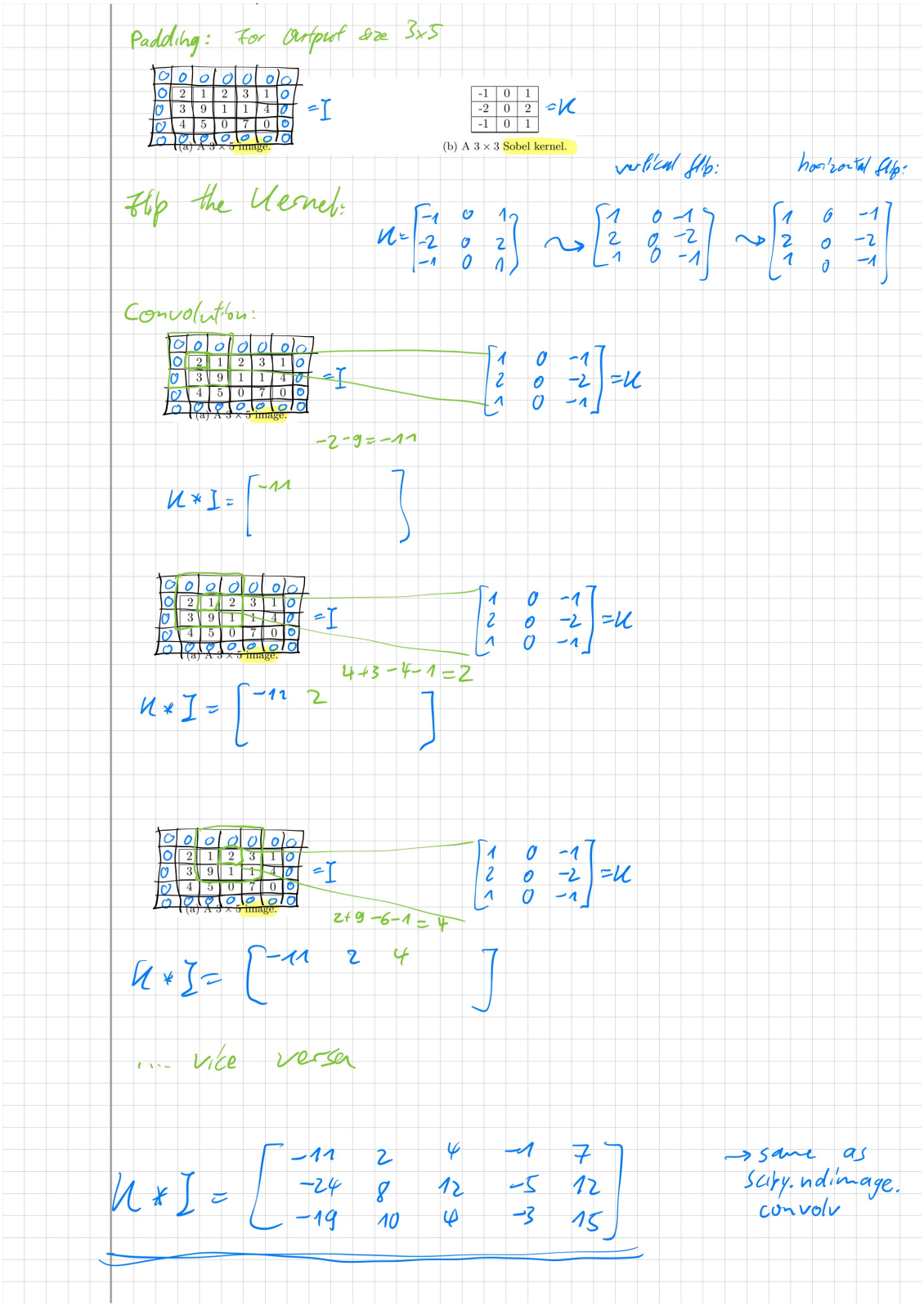
1 Assignment 1 Report

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)



2.2 task 1b)

iii) Max pooling reduces the sensitivity to a small translational pixel shift since the maximum value of a small area is taken, which is robust as long the translational shift is small.

2.3 task 1c)

using formula $W_2 = [(W_1 - F_W + 2P_W)/S_W] + 1$ and $F_W = 7, S_W = 1$ we obtain:

$P_w = \frac{1}{2}(S_w \cdot (w_n - 1) - w_1 + F_w) = \frac{1}{2}(F_w - 1) = 3$

Since the kernel is square, vice versa for H_2 and the required padding for each side is 3.

2.4 task 1d)

using the same equation as in 1c) and using that everything is square / quadratic, so W=H, we get:

$W_2 = \frac{W_1 - F + 2P}{S} + 1, F = S(1 - W_2) + W_1 - 2P = 1 - W_2 + W_1 = 5,$

with $W_1 = 512, W_2 = 508, P = 0, S = 1$. So the Kernel size is 5x5.

2.5 task 1e)

using the same equation and $F = 2, P = 0, S = 2$:

$W_3 = \frac{W_2 - F + 2P}{S} + 1 = 254$

So the pooled features maps are of size 254x254.

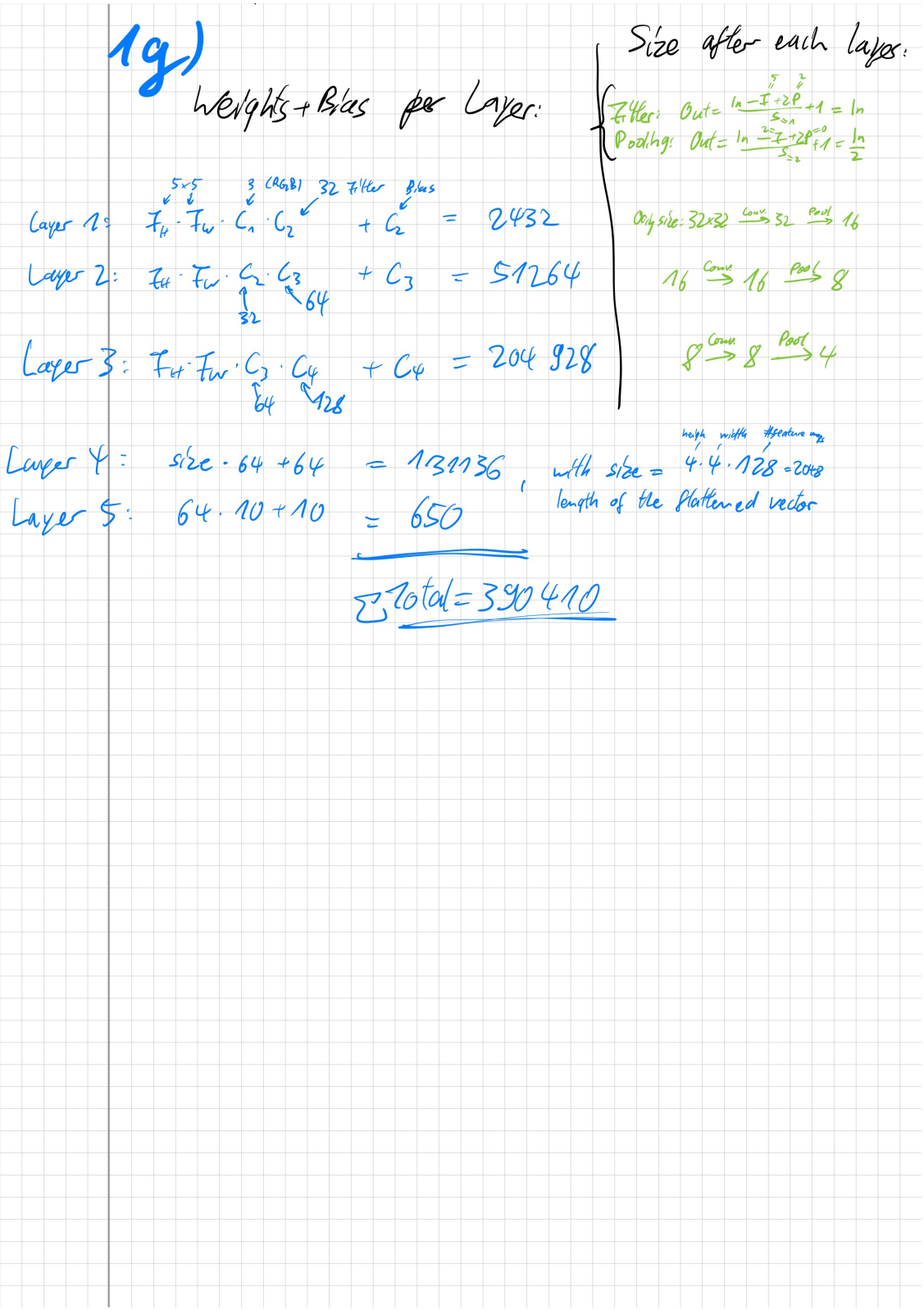
2.6 task 1f)

Using the same equation and $F = 3, P = 0, S = 1$:

$W_4 = \frac{W_3 - F + 2P}{S} + 1 = 252$

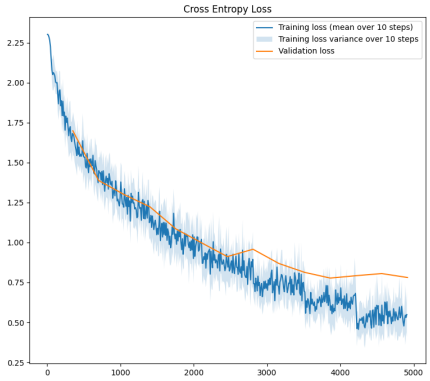
So the feature maps after convolution in the second layer have the size 252x252.

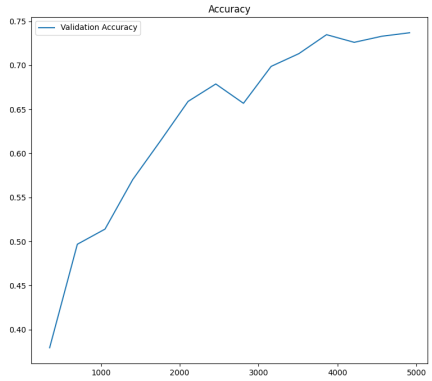
2.7 task 1g)


1g)
Weights + Biases per Layer:
Size after each layer:
Filter: Out = (In − I + 2P)/S + 1 = In
Pooling: Out = (In − I + 2P)/S + 1 = In/2
Layer 1: $F_H \cdot F_W \cdot C_1 \cdot C_2 + C_2 = 2432$
5×5, 3 (RGB), 32 filter, Bias
Orig size: 32×32 → Conv → 32 → Pool → 16
Layer 2: $F_H \cdot F_W \cdot C_2 \cdot C_3 + C_3 = 51264$
16 → Conv → 16 → Pool → 8
Layer 3: $F_H \cdot F_W \cdot C_3 \cdot C_4 + C_4 = 204928$
8 → Conv → 8 → Pool → 4
Layer 4: size · 64 + 64 = 131136, with size = 4 · 4 · 128 = 2048
height, width, #feature maps
length of the flattened vector
Layer 5: 64 · 10 + 10 = 650
Σ Total = 390410

3 Task 2

3.0.1 Task 2a)





3.0.2 Task 2b)

(see graph at Task 2a)

4 Task 3

4.0.1 Task 3a)

optimizer: Adam optimzer

learning rate: 0.001

batch size: 64

no special weight initialization and no regularization as the model achieved 80% before we came to considering these additions

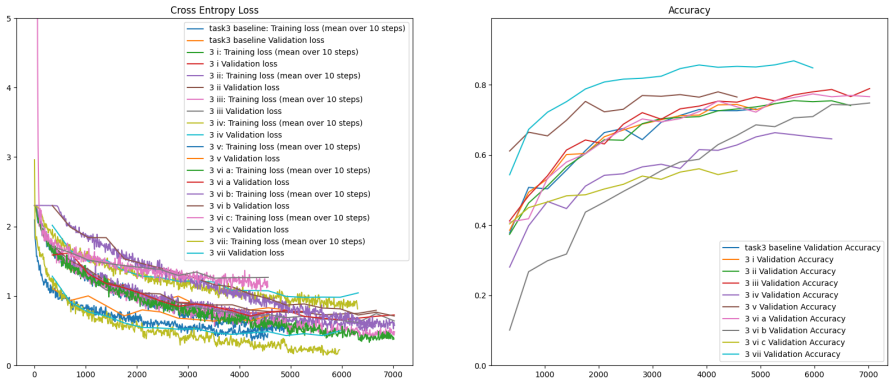
early_stop_count = 4

stride = 1; pooling_stride = 4; padding = 1

see task3.py for reference | Layer | LayerType | Number of Hidden Units/Filters | Activation function | |——-|————-|————————————|——————| | 1 | Conv2D | 64 | LeakyReLU | | 1 | MaxPool2d | - | - | | 1 | Conv2D | 64 | LeakyReLU | | 1 | MaxPool2d | - | - | | 1 | BatchNorm2d | 64 | - | | 2 | Conv2D | 128 | LeakyReLU | | 2 | MaxPool2d | - | - | | 2 | Conv2D | 128 | LeakyReLU | | 2 | MaxPool2d | - | - | | 2 | BatchNorm2d | 128 | - | | 3 | Conv2D | 256 | LeakyReLU | | 3 | MaxPool2d | - | - | | 3 | Conv2D | 256 | LeakyReLU | | 3 | MaxPool2d | - | - | | 3 | BatchNorm2d | 256 | - | | 4 | Conv2D | 1024 | LeakyReLU | | 4 | MaxPool2d | - | - | | 4 | Conv2D | 1024 | LeakyReLU | | 4 | MaxPool2d | - | - | | 4 | BatchNorm2d | 1024 | - | | 5 | Linear | 1024 | |

Layer	LayerType	Number of Hidden Units/Filters	Activation function
1	Conv2D	64	LeakyReLU
1	MaxPool2d	-	-
1	Conv2D	64	LeakyReLU
1	MaxPool2d	-	-
1	BatchNorm2d	64	-
2	Conv2D	128	LeakyReLU
2	MaxPool2d	-	-
2	Conv2D	128	LeakyReLU
2	MaxPool2d	-	-
2	BatchNorm2d	128	-
3	Conv2D	256	LeakyReLU
3	MaxPool2d	-	-
3	Conv2D	256	LeakyReLU
3	MaxPool2d	-	-
3	BatchNorm2d	256	-
4	Conv2D	1024	LeakyReLU
4	MaxPool2d	-	-
4	Conv2D	1024	LeakyReLU
4	MaxPool2d	-	-
4	BatchNorm2d	1024	-
5	Linear	1024	

4.0.2 Task 3b)

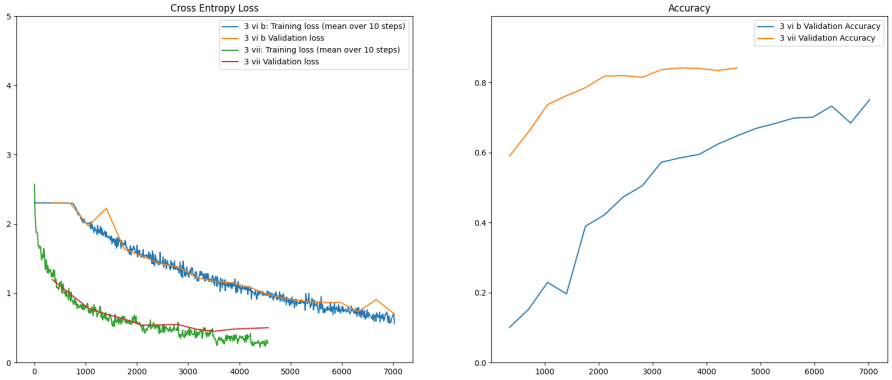


4.0.3 Task 3c)

The biggest improvement was observed in the last step of the incremental changes. So the change of the optimizer together with a different learning rate and adding batch normalization after each pooling layer made the biggest improvement. Batch normalization can help greatly as normalizing the data e.g. helps with the vanishing gradient problem.

Using LogSigmoid as an activation function did not work good at all, different cases requires different activation functions, so it seems LogSigmoid is just a bad activation function for this particular task, architecture and dataset.

4.0.4 Task 3d)



4.0.5 Task 3e)

The final model “3 vii” already did reach an accuracy of 80% (see previous plot)

4.0.6 Task 3f)

The loss curve for the final model “3 vii” gets slightly bigger after “initially converging”, this is an indicator for overfitting.

5 Task 4

5.1 Task 4a)

Hyperparameters:

- batch size = 32
- learning rate = 5E-4

Other settings:

- Optimizer: Adam
- use data augmentation, since it slighty improved the accuracy
- use transforms.Resize !
- changed mean and std to those of ImageNet dataset !

Resulting final accuracy of **90.4%**

Hyperparameters: - batch size = 32 - learning rate = 5E-4

Other settings: - Optimizer: Adam - use data augmentation, since it slighty improved the accuracy - use transforms.Resize ! - changed mean and std to those of ImageNet dataset !

Resulting final accuracy of 90.4%