Image Processing - Assignment 1

Group 3
Martin Eek Gerhardsen

October 12

Department of Engineering Cybernetics

Contents

1 Spatial Filtering **1**
1.1 Task 1: Theory 1
1.2 Task 2: Programming 5

2 Neural Networks **9**
2.1 Task 3: Theory 9
2.2 Task 4: Programming 11

I =

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

1 Spatial Filtering

1.1 Task 1: Theory

a)

Sampling is the process of converting a continuos-time signal to a discrete-time signal, usually by measuring the continuos-time signal at specific points in time and extending this measurement over a set time step.

b)

Quantization is the process of constraining a signal from a larger to a smaller set of values, like mapping colours to the standard RGB range of 256 integer values.

c)

A high contrast image histogram would look similar to a dirac delta function, with most values grouped together around the same intensity.

d)

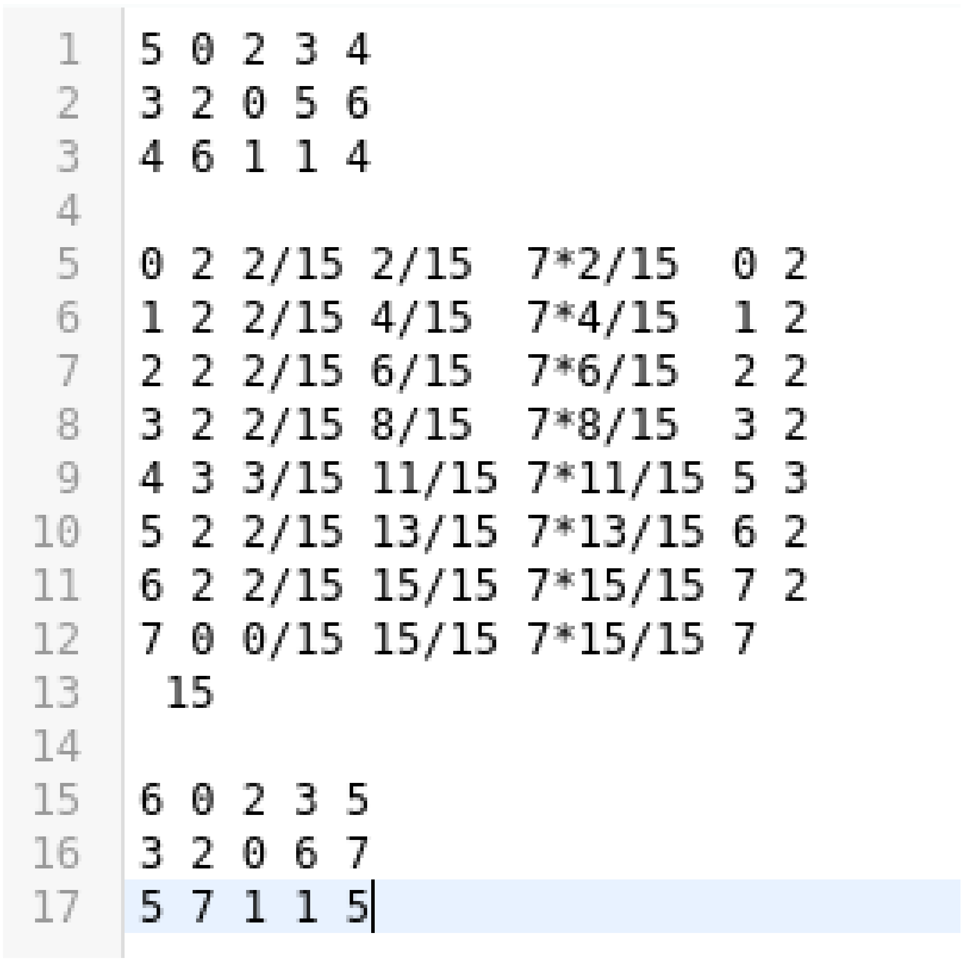
Couldn't be bothered to do this in Latex, so see fig. 1. The top matrix is the image before equalization, and the bottom is the image after equalization. The columns are the following: intensities, counts of intensities, pdf, cdf, cdf multiplied with scaling, intensity rounding, new intensity count. Then, using the intensity rounding and mapping the old intensities to the new, we get the final, equalized image.

e)

The log transform will increase the output intensity for low input intensities, and flatten for high input intensities. This will make it easier to see the lower ranges of the input intensities. If we then apply this transform to a high variance image, then the lower part of the dynamic range will be enhanced, while the higher part looses information.

f)

Using zero-padding for this convolution. I flipped the kernel and used correlation and matrix multiplication to get the convolution. Convolving section 1.1



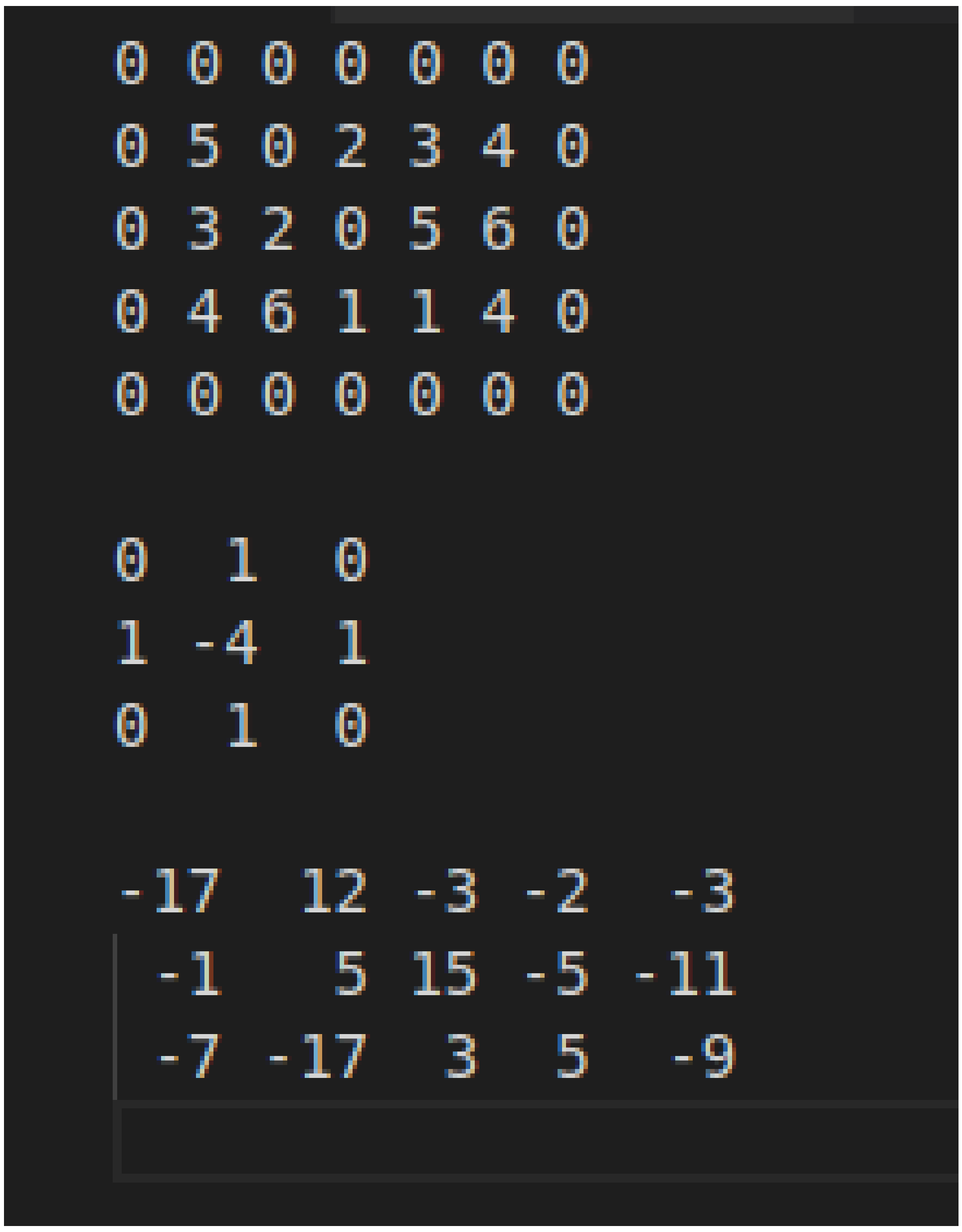
```
5 0 2 3 4
3 2 0 5 6
4 6 1 1 4

0 2 2/15 2/15  7*2/15  0 2
1 2 2/15 4/15  7*4/15  1 2
2 2 2/15 6/15  7*6/15  2 2
3 2 2/15 8/15  7*8/15  3 2
4 3 3/15 11/15 7*11/15 5 3
5 2 2/15 13/15 7*13/15 6 2
6 2 2/15 15/15 7*15/15 7 2
7 0 0/15 15/15 7*15/15 7
 15

6 0 2 3 5
3 2 0 6 7
5 7 1 1 5
```

Figure 1: Histogram equalization

K =

0	1	0
1	-4	1
0	1	0

and section 1.1. See fig. 2:



```
0 0 0 0 0 0 0
0 5 0 2 3 4 0
0 3 2 0 5 6 0
0 4 6 1 1 4 0
0 0 0 0 0 0 0

0  1  0
1 -4  1
0  1  0

-17  12 -3 -2  -3
 -1   5 15 -5 -11
 -7 -17  3  5  -9
```

Figure 2: Convolution by hand

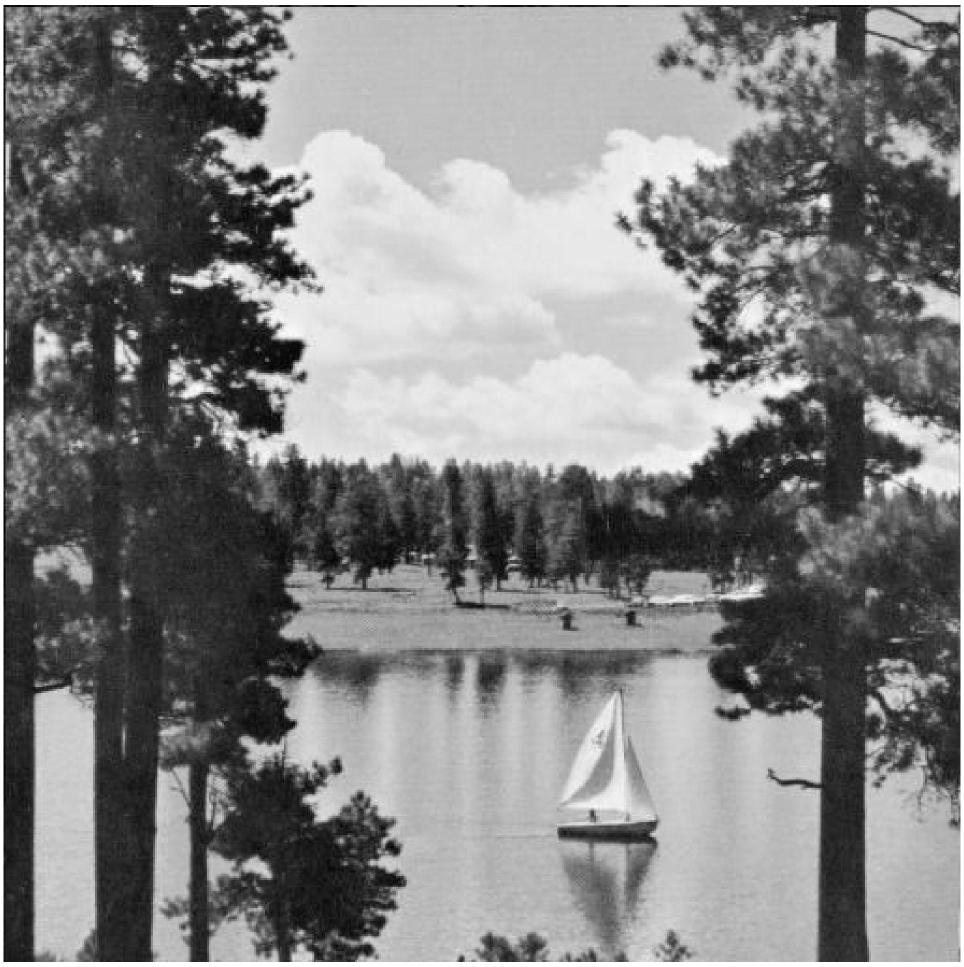


Figure 3: The grey scale version of the lake image

1.2 Task 2: Programming

a)

See the result from fig. 3.

b)

Se the result from fig. 4.

c)

See the results from fig. 5 and fig. 6.

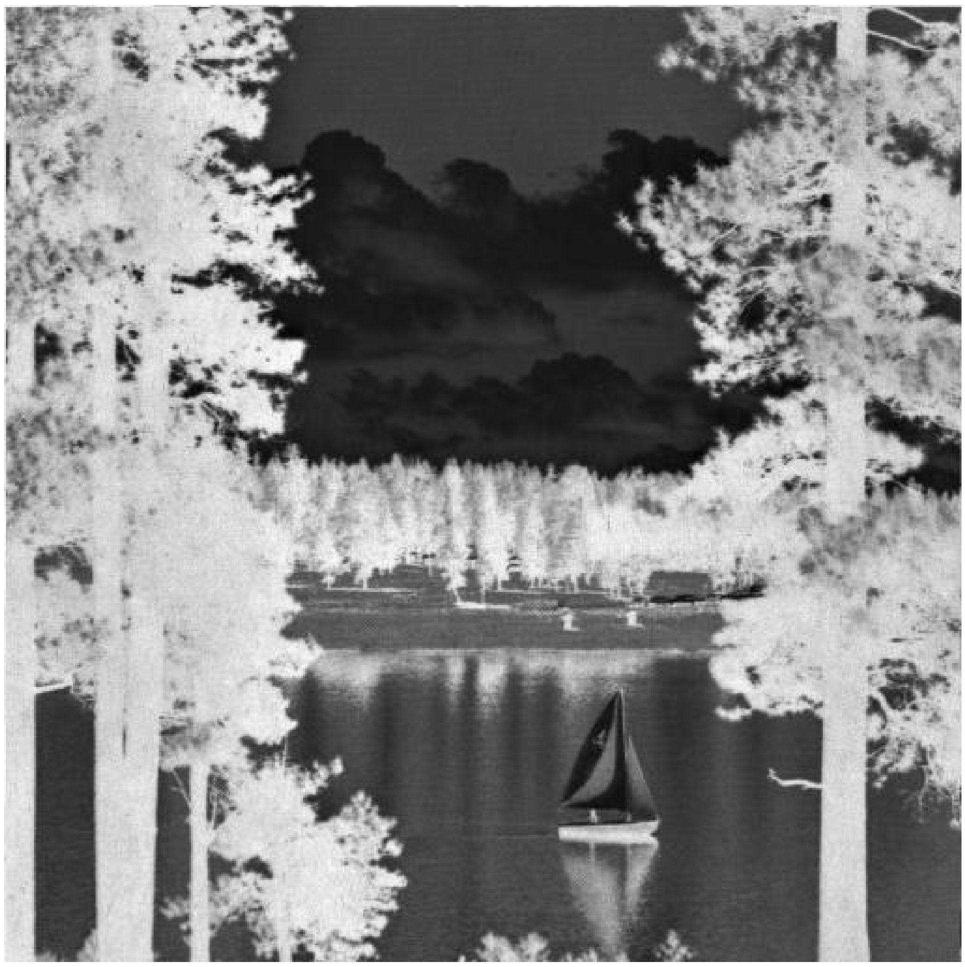


Figure 4: The inverse of the grey scale version of the lake image

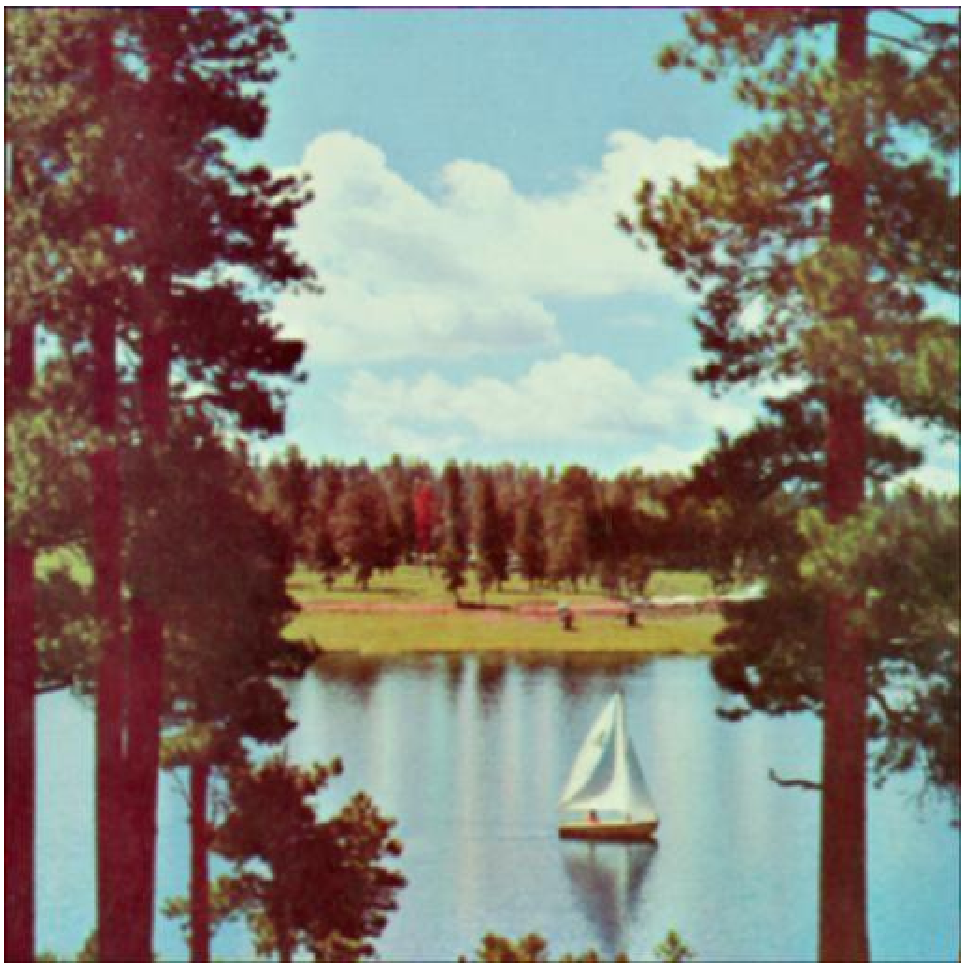


Figure 5: Convolution of the lake image with the kernel h_a.

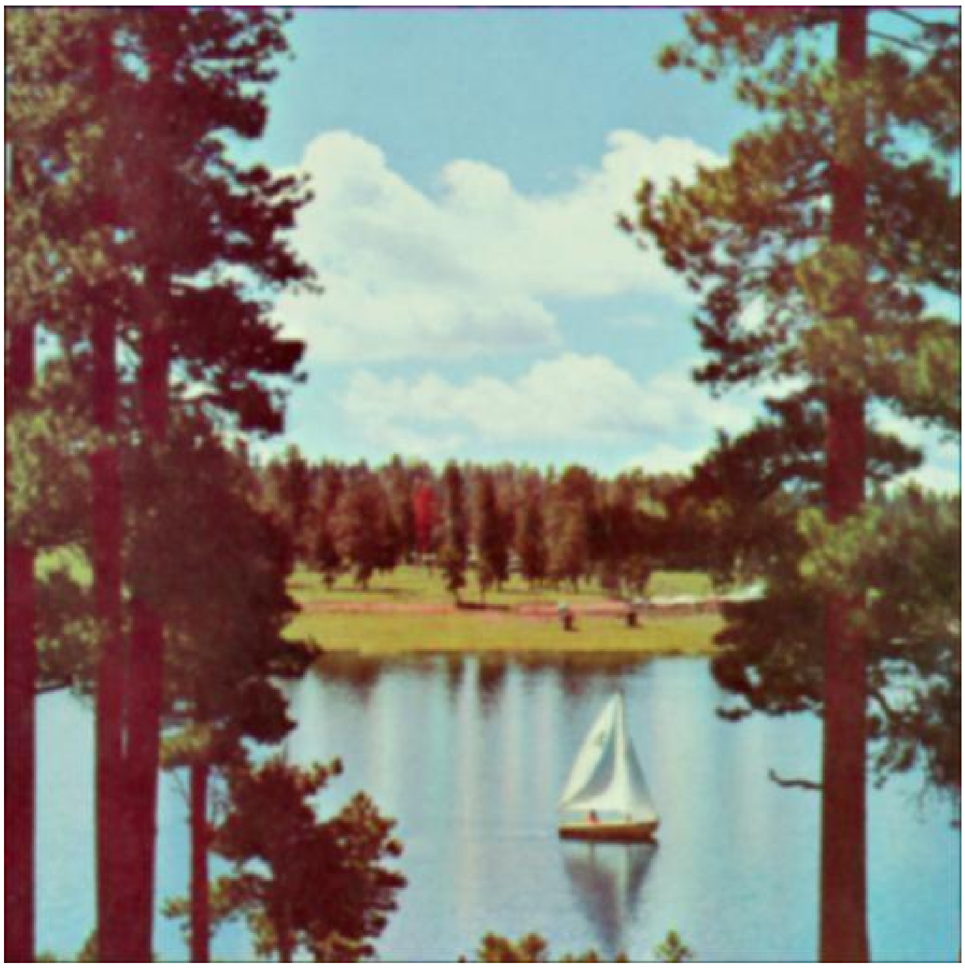


Figure 6: Convolution of the lake image with the kernel h_b.

2 Neural Networks

2.1 Task 3: Theory

a)

The binary operation XOR, or exclusive or, cannot be represented by a single-layer neural network. If we look at the two binary input values as the two digits of a binary number, we get the range $[0, 3]$. Then applying the XOR function for this range, we get $[0, 1, 1, 0]$, which obviously cannot be represented by a linear function.

b)

A hyperparameter is a parameter which is set before, and not changed by, the learning process. Batch size and learning rate are examples of hyperparameters.

c)

The softmax function is a smooth approximation of the arg max function. The values being applied to the are shifted into the range $[0, 1]$, so that they can be interpreted as probabilities, and further as a probability density function, where the sum of the result is 1.

d)

$$C(y_n, \hat{y}_n) = (y_n - \hat{y}_n)^2, \hat{y}_n = 1 \tag{1}$$

Using eq. (1), we calculate:

$$\begin{aligned}
w_1' &= \frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_1}\frac{\partial c_1}{\partial a_1}\frac{\partial a_1}{\partial w_1} = 2 * (y_n - \hat{y}_n) * 1 * 1 * w_1 = -2 * (y_n - 1) \\
w_2' &= \frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_1}\frac{\partial c_1}{\partial a_2}\frac{\partial a_2}{\partial w_2} = 2 * (y_n - \hat{y}_n) * 1 * 1 * w_2 = 2 * (y_n - 1) \\
w_3' &= \frac{\partial C}{\partial w_3} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_2}\frac{\partial c_2}{\partial a_3}\frac{\partial a_3}{\partial w_3} = 2 * (y_n - \hat{y}_n) * 1 * 1 * w_3 = -2 * (y_n - 1) \\
w_4' &= \frac{\partial C}{\partial w_4} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_2}\frac{\partial c_2}{\partial a_4}\frac{\partial a_4}{\partial w_4} = 2 * (y_n - \hat{y}_n) * 1 * 1 * w_4 = -4 * (y_n - 1) \\
b_1' &= \frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_1}\frac{\partial c_1}{\partial b_1} = 2 * (y_n - \hat{y}_n) * 1 * 1 = 2 * (y_n - 1) \\
b_2' &= \frac{\partial C}{\partial b_2} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_2}\frac{\partial c_2}{\partial b_2} = 2 * (y_n - \hat{y}_n) * 1 * 1 = 2 * (y_n - 1)
\end{aligned}$$

e)

Calculating the current value for y:

$$\begin{array}{llll} a_1 = 1, & a_2 = 0, & a_3 = 1, & a_4 = -4 \\ c_1 = 2, & c_2 = -4, & y_n = max(c_1, c_2) = 2 & \end{array}$$

$$\theta_{t+1} = w_t - \alpha \frac{\partial C}{\partial \theta_t} \tag{2}$$

Using eq. (2) and $\alpha = 0.1$, we get:

$$\begin{aligned} w_{1,t+1} &= w_{1,t} - \alpha \frac{\partial C}{\partial w_1} = -1 - 0.1 * (-2 * (2-1)) = -0.8 \\ w_{3,t+1} &= w_{3,t} - \alpha \frac{\partial C}{\partial w_3} = -1 - 0.1 * (-2 * (2-1)) = -0.8 \\ b_{1,t+1} &= b_{1,t} - \alpha \frac{\partial C}{\partial b_1} = 1 - 0.1 * 2 * (2-1) = 0.8 \end{aligned}$$

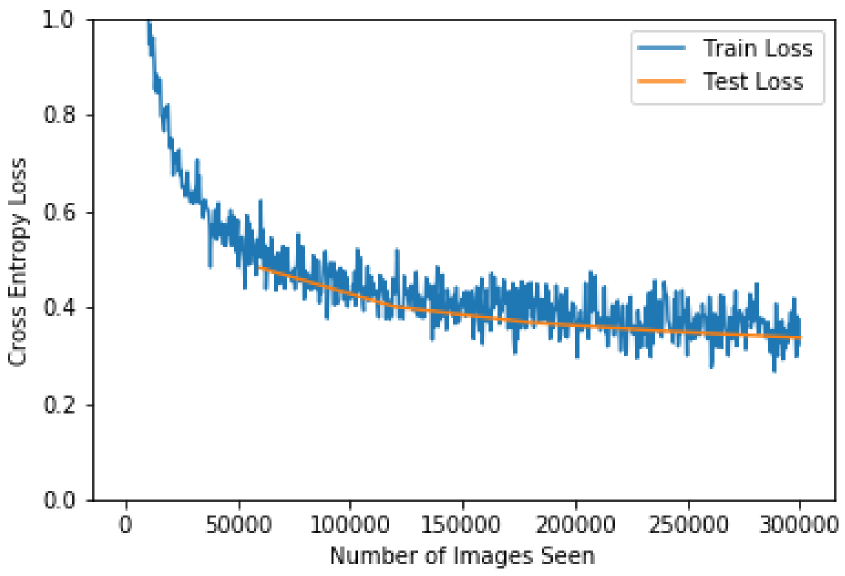


Figure 7: Training and test loss before the task

2.2 Task 4: Programming

a)

The original training loss can be seen fig. 7 and the normalized here: fig. 8.

b)

For each class, the gray-scale images were generated by normalizing the weights for each class and reshaping the weights inverse of the way we convert an image to the input layer. We can see the weight image in the figures: fig. 9, fig. 10, fig. 11, fig. 12, fig. 13, fig. 14, fig. 15, fig. 16, fig. 17 and fig. 18.

This weights give what points of the image should increase the probability of that specific class being the correct one. As we only have one layer, this is a mapping of the input image to the output, and we can see the shapes that are *extracted* by the weights, and we can note that they are similar to the number we are looking for. This can be especially well from fig. 9, with the large negative space in the center being important to determining if a number is zero or not.

c)

The accuracy is now way worse. This is due to the large learning rate. Even though a larger learning rate may give faster learning, it is also makes the

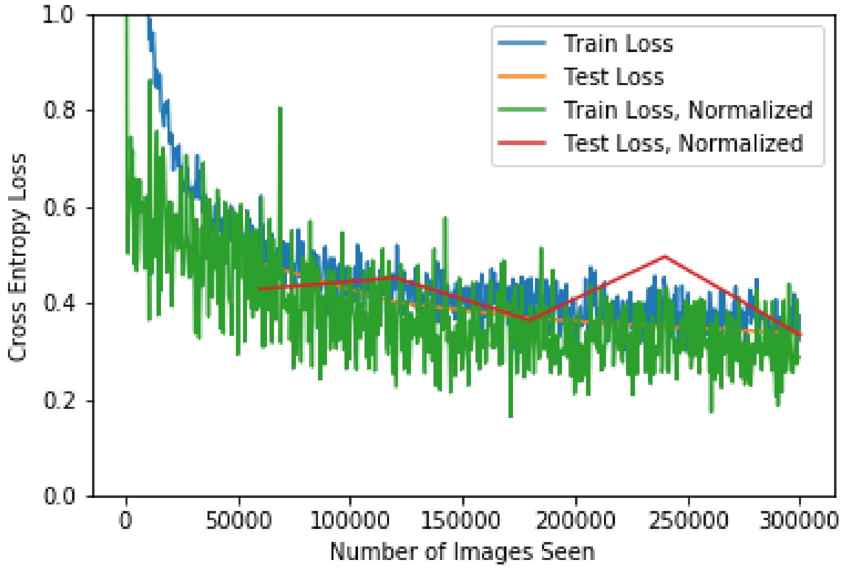


Figure 8: Original vs. normalized training and test loss

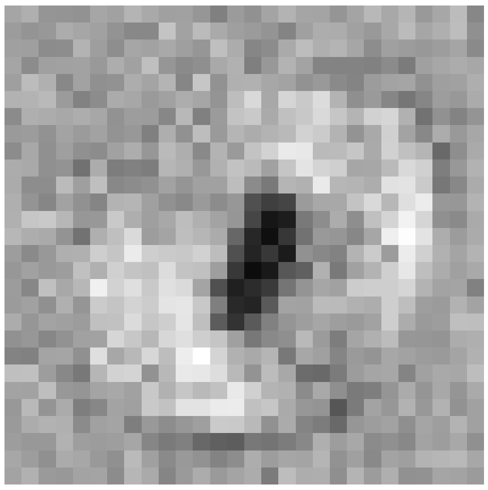


Figure 9: Class weight 0

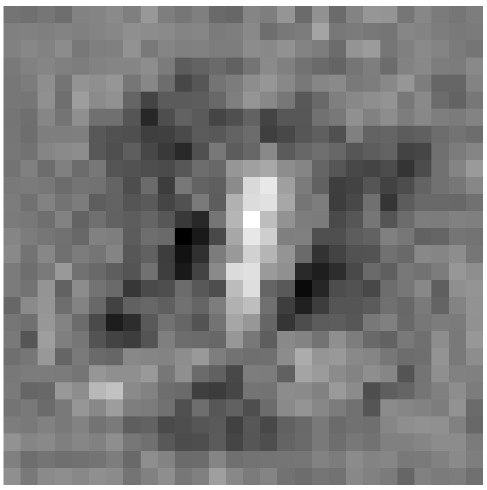


Figure 10: Class weight 1

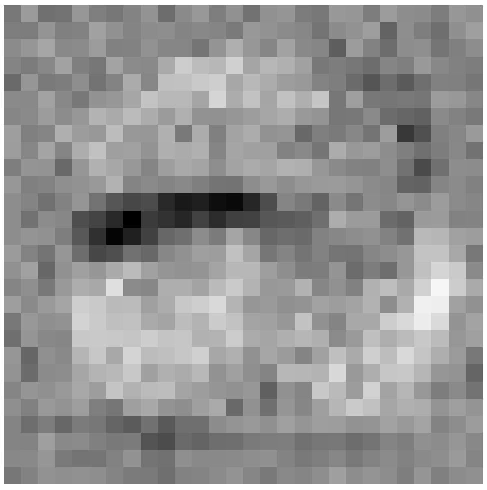


Figure 11: Class weight 2

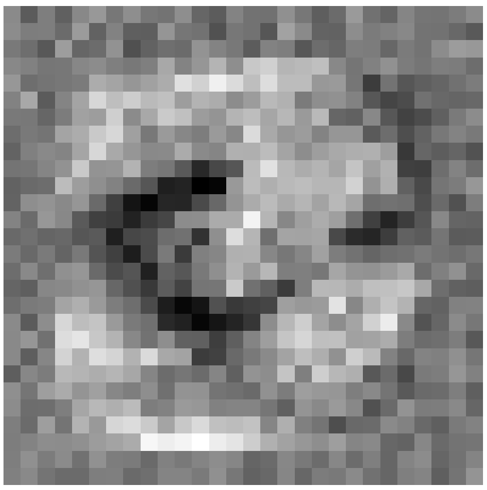


Figure 12: Class weight 3

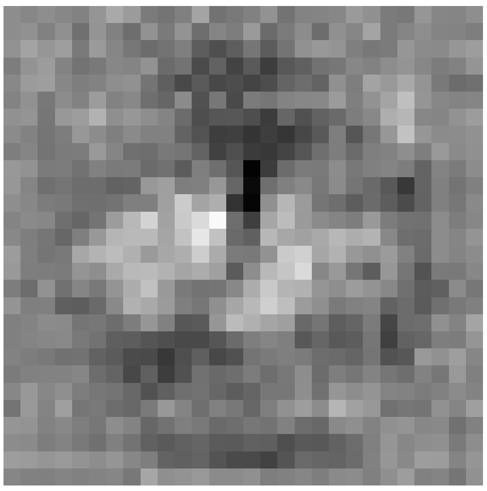


Figure 13: Class weight 4

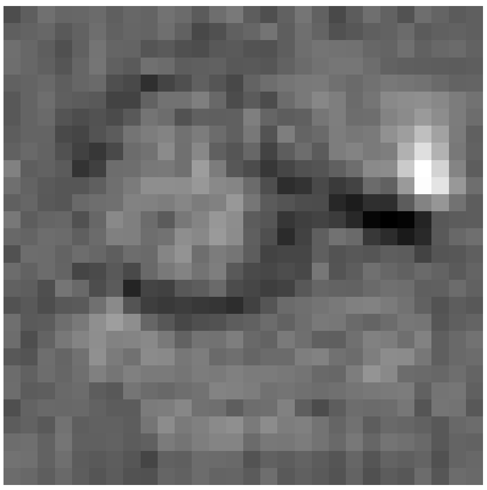


Figure 14: Class weight 5

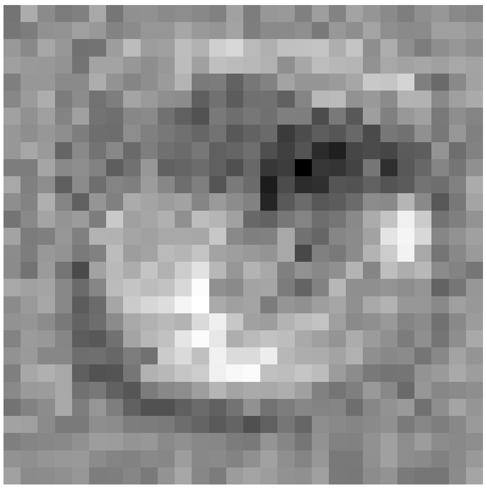


Figure 15: Class weight 6

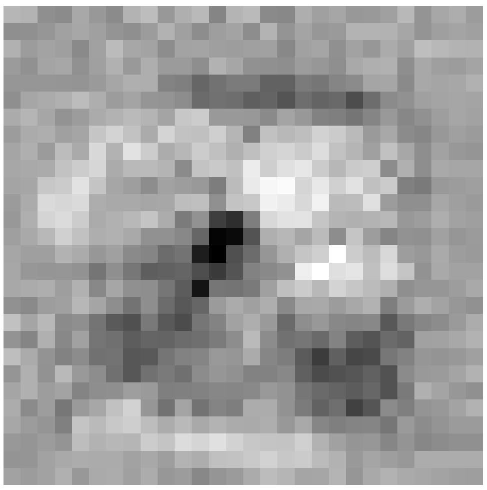


Figure 16: Class weight 7

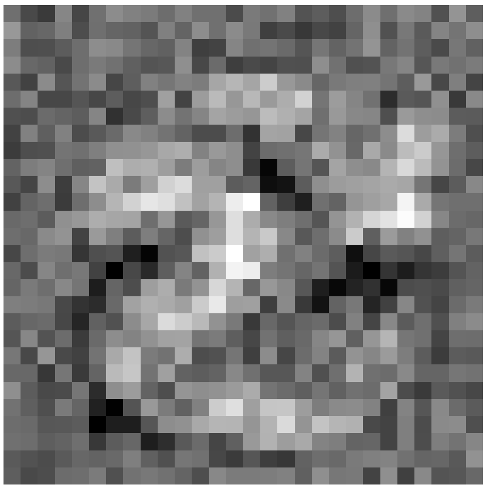


Figure 17: Class weight 8

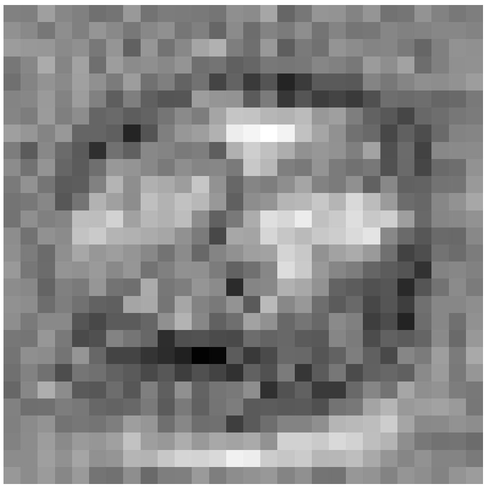


Figure 18: Class weight 9

gradient descent be less accurate, and we will not reach the correct weights to minimize the cost function.

d)

From fig. 20 we can see that the training loss and test loss end up converging quickly to a lower cross entropy loss than the original neural network of fig. 7 and fig. 8.

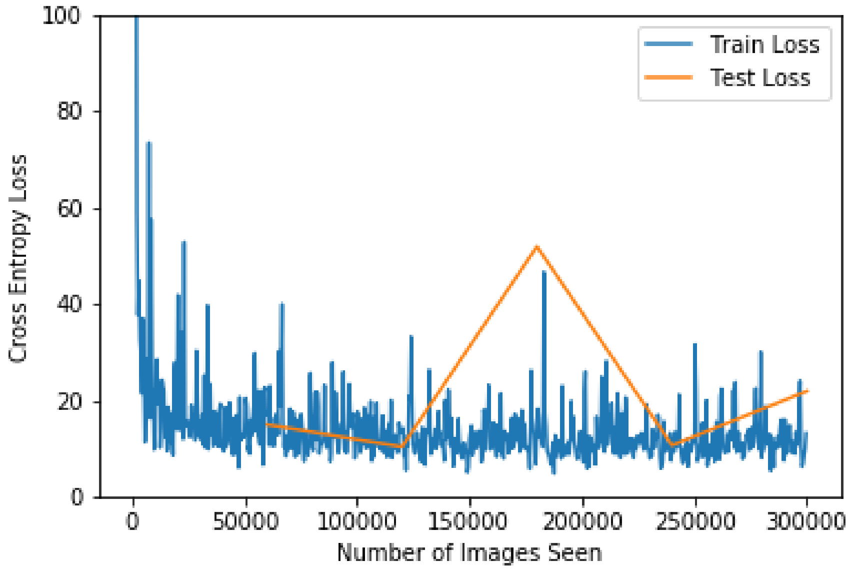


Figure 19: Training and test loss with learning rate 1

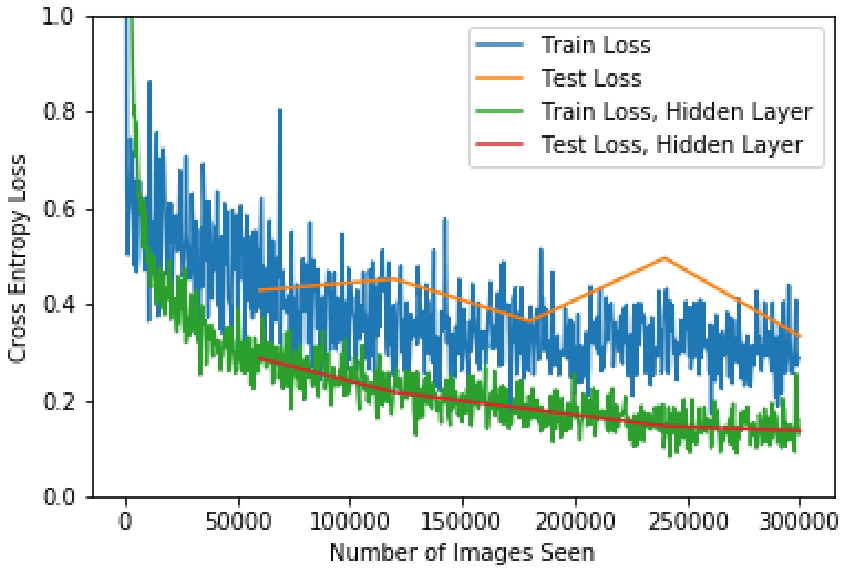


Figure 20: Training and test loss with new hidden layer