Image Processing - Assignment 1

Spatial Filtering

Task 1: Theory [1.5pt]

A digital image is constructed from an image sensor. An image sensor outputs a continuous voltage waveform that represents the image, and to construct a digital image, we need to convert this continuous signal. This conversion involves two processes: sampling and quantization.

a) **[0.1pt] Explain in one sentence what sampling is.**
Sampling discretizes what intervals of the continuous signal to process, thereby determining the spatial resolution of the digitized image.

b) **[0.1pt] Explain in one sentence what quantization is.**
Quantization is discretizing the continuous values of one sample, into one value by for example rounding, determining the color intensity.

c) **[0.2pt] Looking at an image histogram, how can you see that the image has high contrast?**
If all the pixels have almost the same intensity values (x-axis), this means that the whole picture will have a very uniform color, rendering it in a low contrast. On the contrary, if the intensity values of the pixels are spread uniformly (meaning a lot of pixels in different colors), the contrast is high.

d) **[0.5pt] Perform histogram equalization by hand on the 3-bit (8 intensity levels) image in Figure 1a Your report must include all the steps you did to compute the histogram, the transformation, and the transformed image. Round down any resulting pixel intesities that are not integer (use the floor operator).**
H(r) = number_of_r / number_of_pixels => r_n / 15
F(r) = F_r-1 + p(r)
T(r) = floor((L-1)*F(r)), L=8

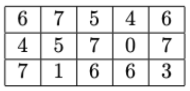
r	H[r]	p[r]	F[r]	T[r]
0	1	0.067	0.067	0
1	1	0.067	0.133	0
2	0	0.000	0.133	0
3	1	0.067	0.200	1
4	2	0.133	0.333	2
5	2	0.133	0.467	3
6	4	0.267	0.733	5
7	4	0.267	1.000	7

Output image:

5	7	3	2	5
2	3	7	0	7
7	0	5	5	1

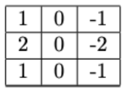
e) **[0.1pt] What happens to the dynamic range if we apply a log transform to an image with a large variance in pixel intensities? Hint: A log transform is given by s = c · log(1 + r), where r and s is the pixel intensity before and after transformation, respectively. c is a constant.**
The log transformation causes the bright/high intensities to be squeezed, and the low/dark intensities to be widened (the inverse log does of course the opposite). So this transformation should only be applied where low pixel values are more frequent than higher ones, otherwise this would cause a lot of information loss. In an image with a large variance in pixel intensities, the higher pixel intensities would become indistinguishable.

f) [0.5pt] Perform spatial convolution by hand on the image in Figure 1a using the kernel in Figure 1b. The convolved image should be 3 ×5. You are free to choose how you handle boundary conditions, and state how you handle them in the report.



6	7	5	4	6
4	5	7	0	7
7	1	6	6	3

(a) A 3 × 5 image.



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

(b) A 3 × 3 Sobel kernel.

Figure 1: An image I and a convolutional kernel K. For the image, each square represents an image pixel, where the value inside is the pixel intensity in the $[0, 7]$ range (3-bit).

I will be using the zero padding method, and remember to rotate the kernel 180 degrees before beginning:

Rotated Kernel:

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

Spatial convoluted image:

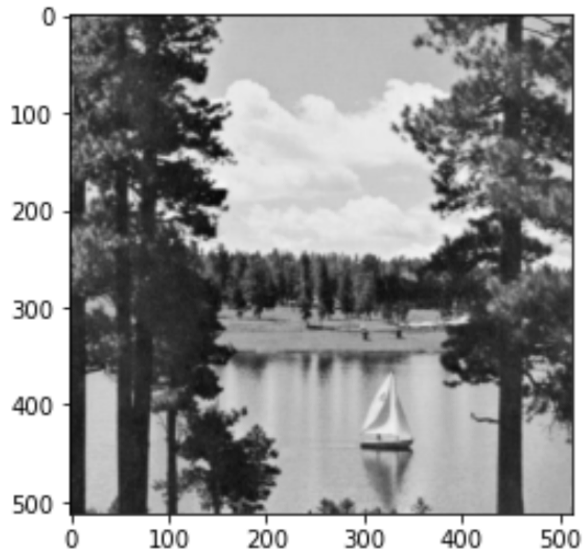
0*6 + 2*7 + 0*4 + 1*5	-2*6 + 0*7 + 2*5 + -1*4 + 0*5 + 1*7	-2*7 + 0*5 + 2*4 + -1*5 + 0*7 + 1*0	-2*5 + 0*4 + 2*6 + -1*7 + 0*0 + 1*7	-2*4 + 0*6 + -1*0 + 0*7
0*6 + 1*7 + 0*4 + 2*5 + 0*7 + 1*1	-1*6 + 0*7 + 1*5 + -2*4 + 0*5 + 2*7 + -1*7 + 0*1 + 1*6	-1*7 + 0*5 + 1*4 + -2*5 + 0*7 + 2*0 + -1*1 + 0*6 + 1*6	-1*5 + 0*4 + 1*6 + -2*7 + 0*0 + 2*7 + -1*6 + 0*6 + 1*3	-1*4 + 0*6 + -2*0 + 0*7 + -1*6 + 0*3
0*4 + 1*5 + 0*7 + 2*1	-1*4 + 0*5 + 1*7 + -2*7 + 0*1 + 2*6	-1*5 + 0*7 + 1*0 + -2*1 + 0*6 + 2*6	-1*7 + 0*0 + 1*7 + -2*6 + 0*6 + 2*3	-1*0 + 0*7 + -2*6 + 0*3

=

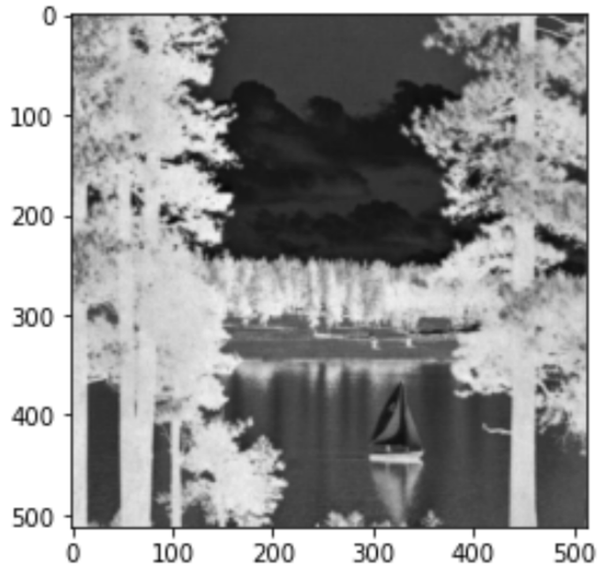
19	1	-11	2	-8
18	4	-8	-2	-10
7	1	5	-6	-12

Task 2: Programming [1.0pt]

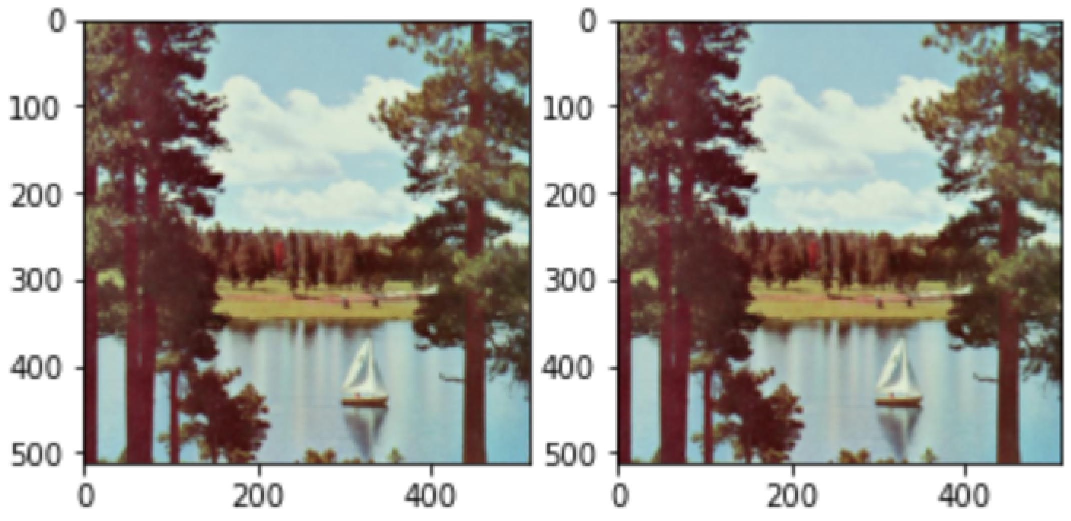
a)



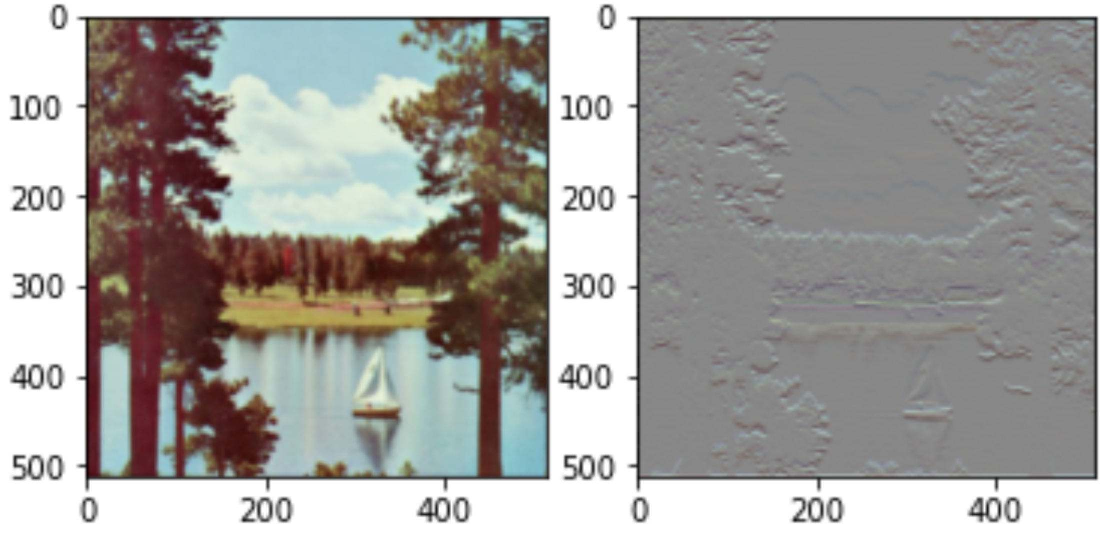
b)



c) Before:



After:



Neural Networks

Task 3: Theory [1.0pt]

a) **[0.1pt] A single-layer neural network is a linear function. Give an example of a binary operation that a single-layer neural network cannot represent (either AND, OR, NOT, NOR, NAND, or XOR).**

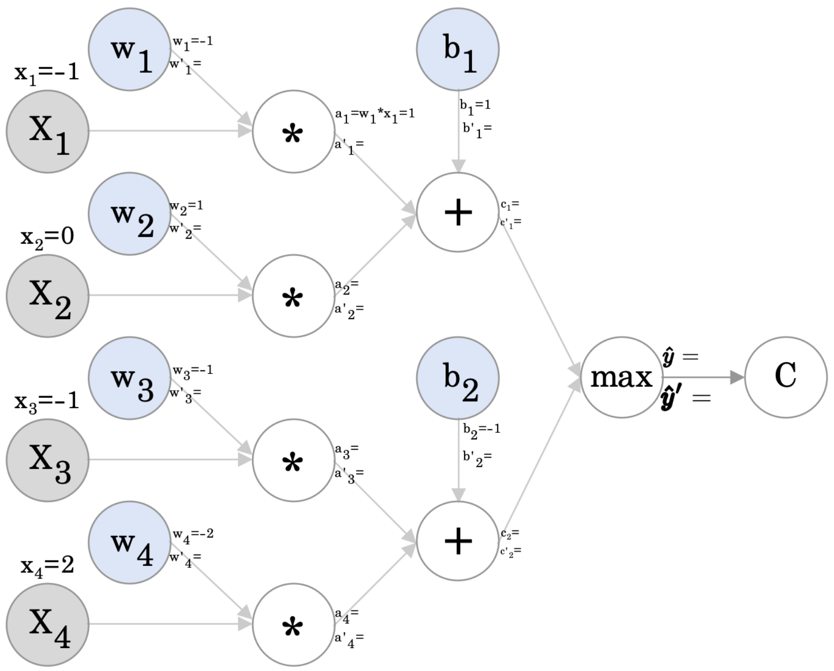
	AND	OR	NOT	NOR	NAND	XOR
LINEAR	YES	YES	YES	YES	YES	NO

XOR is the only operator of the given ones that a single-layer neural network cannot represent since it is not linearly separable.

b) **[0.1pt] Explain in one sentence what a hyperparameter for a neural network is. Give two examples of a hyperparameter.**
Hyperparameters are variables that are set before training, and determine how the network is trained and structured. Learning rate is a hyperparameter which determines how fast the network will learn during the gradient descent. The number of hidden layers is also a hyperparameter which needs to be set before the training begins.

c) **[0.1pt] Why is the softmax activation function used in the last layer for neural networks trained to classify objects?**
Because the output needs to be normalized to a probability distribution for the network to be able to classify the output.

d) **[0.5pt] Figure 2 shows a simple neural network. Perform a forward pass and backward pass on this network with the given input values. Use Equation 3 as the cost function and let the target value be y = 1. Find and report the final values for $\frac{\partial C}{\partial w1}$, $\frac{\partial C}{\partial w2}$, $\frac{\partial C}{\partial w3}$, $\frac{\partial C}{\partial w4}$, $\frac{\partial C}{\partial b1}$ and $\frac{\partial C}{\partial b2}$. Explain each step in the computation, such that it is clear how you compute the derivatives.**



In the forward propagation we start by looking at the input layer with x_n and calculate the sum of the inputs multiplied by the appropriate weights:

$a_i = w_i * x_i$

$a_1 = (-1) * (-1) = 1$

$a_2 = 1 * 0 = 0$

$a_3 = (-1) * (-1) = 1$

$a_4 = (-2) * 2 = -4$

$c_1 = (a_1 + a_2) + b_1$ = 1 + 0 + 1 = 2

$c_2 = (a_3 + a_4) + b_2$ = 1 + -4 + -1 = -4

$\hat{y} = max(c_1, c_2) = 2$

The next step is to apply the cost function on ŷ to calculate how much we have to calibrate our weights with gradient descent.

$$C(y_n, ŷ) = \frac{1}{2}(y_n - ŷ)^2 => C(1, 2) = 0.5$$

$$\frac{\partial C}{\partial y} = y_n - ŷ = 1 - 2 = -1$$

$$\frac{\partial C}{\partial c1} = \frac{\partial C}{\partial y} * \frac{\partial y}{\partial c1} = -1 * 1$$

$$\frac{\partial C}{\partial b1} = \frac{\partial C}{\partial c1} * \frac{\partial c1}{\partial b1} = -1 * 1 = -1$$

$$\frac{\partial C}{\partial a1} = \frac{\partial C}{\partial c1} * \frac{\partial y}{\partial a1} = -1 * 1$$

$$\frac{\partial C}{\partial w1} = \frac{\partial C}{\partial c1} * \frac{\partial w1}{\partial a1} = -1x_1 = 1$$

$$\frac{\partial C}{\partial w2} = \frac{\partial C}{\partial c1} * \frac{\partial w2}{\partial a2} = -1x_2 = 0$$

$$\frac{\partial C}{\partial c2} = \frac{\partial C}{\partial y} * \frac{\partial y}{\partial c2} = -1 * 1$$

$$\frac{\partial C}{\partial b2} = \frac{\partial C}{\partial c2} * \frac{\partial c2}{\partial b2} = -1 * 1 = -1$$

$$\frac{\partial C}{\partial w3} = \frac{\partial C}{\partial c2} * \frac{\partial w3}{\partial a3} = -1x_3 = 1$$

$$\frac{\partial C}{\partial w4} = \frac{\partial C}{\partial c2} * \frac{\partial w4}{\partial a4} = -1x_4 =- 2$$

e) **[0.2pt] Compute the updated weights w1, w3, and b1 by using gradient descent and the values you found in task d. Use α = 0.1**

$$w1 = w1 - \alpha\frac{\partial C}{\partial w1} = -1 - 0.1 * (-1) = -0.9$$

$$w3 = w3 - \alpha\frac{\partial C}{\partial w3} = -1 - 0.1 * (1) = -1.1$$

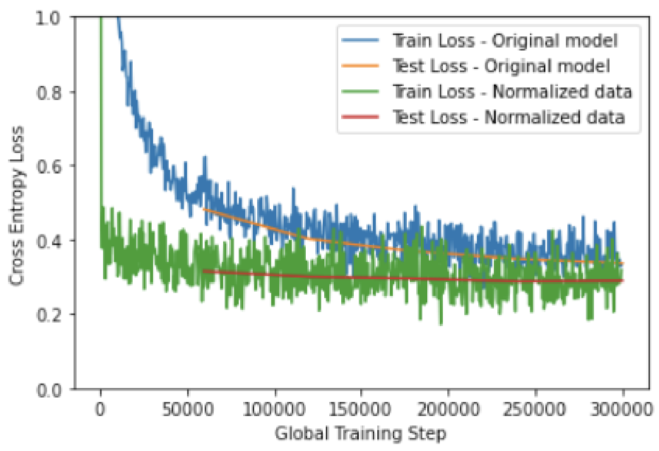
$$b1 = b1 - \alpha\frac{\partial C}{\partial b1} = 1 - 0.1 * (-1) = 1.1$$

Task 4: Programming [1.5pt]

a) **[0.4pt] Use the given starter code and train a single-layer neural network with batch size of 64. Then, normalize every image between a range of [-1. 1], and train the network again. Plot the training and validation loss from both of the networks in the same graph. Include the graph in your report. Do you notice any difference when training your network with/without normalization?**
Tip: You can normalize the image to the range of [-1, 1] by using an image transform. Use torchvision.transforms.Normalize with mean = 0.5, and std = 0.5, and include it after transforms.ToTensor(). From this task, use normalization for every subsequent task.
I noticed 2 differences, the first was that the model used longer time (3-4seconds, about twice as long) to finish the training:

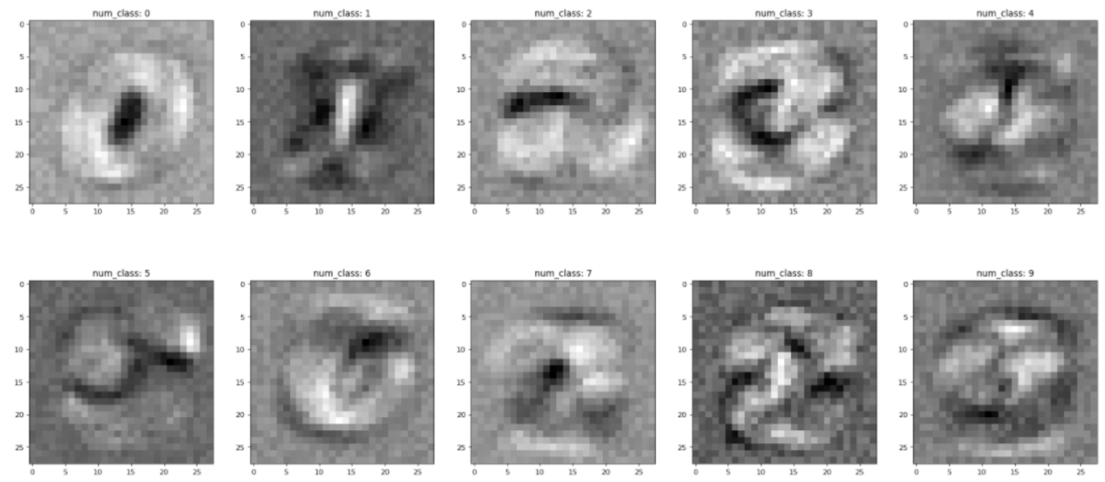
```
Training epoch 0: 100%|██████████| 938/938 [00:05<00:00, 184.88it/s]
Training epoch 1: 100%|██████████| 938/938 [00:05<00:00, 179.51it/s]
Training epoch 2: 100%|██████████| 938/938 [00:04<00:00, 187.85it/s]
Training epoch 3: 100%|██████████| 938/938 [00:05<00:00, 176.75it/s]
Training epoch 4: 100%|██████████| 938/938 [00:05<00:00, 185.35it/s]
Training epoch 0: 100%|██████████| 938/938 [00:08<00:00, 109.38it/s]
Training epoch 1: 100%|██████████| 938/938 [00:08<00:00, 108.86it/s]
Training epoch 2: 100%|██████████| 938/938 [00:08<00:00, 109.73it/s]
Training epoch 3: 100%|██████████| 938/938 [00:08<00:00, 109.51it/s]
Training epoch 4: 100%|██████████| 938/938 [00:08<00:00, 109.31it/s]
```

The second difference was the performance. The neural network trained on the normalized data learned quicker, and ended up with a higher accuracy on the test set.



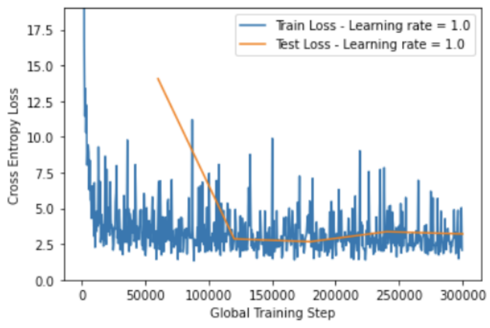
```
Final Orignal Test loss: 0.33633791579372563. Final Original Test accuracy: 0.9088
Final Normalized Test loss: 0.289234658050689. Final Normalized Test accuracy: 0.9179
```

b) **[0.4pt] The trained neural network will have one weight with shape [num classes, 28 × 28]. To visualize the learned weight, we can plot the weight as a 28 × 28 grayscale image. For each digit (0-9), plot the learned weight as a 28 × 28 image. In your report, include the image for each weight, and describe what you observe (1-2 sentences). Tip: You can access the weight of the fully connected layer by using the following snippet: weight = list(model.children())[1].weight.cpu().data**



We can easily recognise some of the numbers in the labeled images like num_class: 0, 1, 6 etc. But some of the images are unrecognisable like for example num_class 7 and 8.

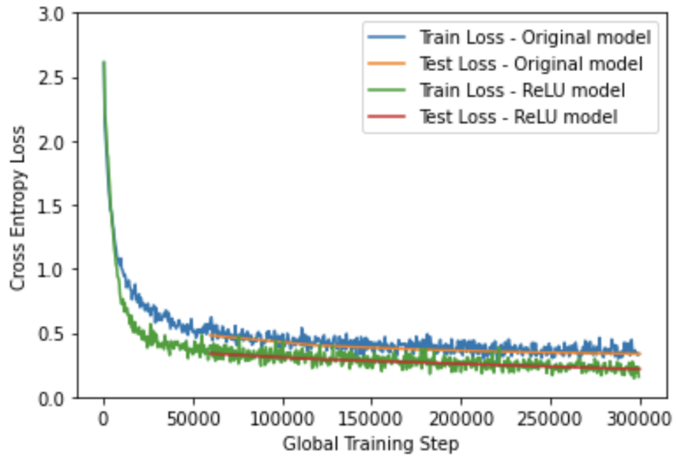
c) **[0.3pt] Set the learning rate to lr = 1.0, and train the network from scratch. Report the accuracy and average cross entropy loss on the validation set. In 1-2 sentences, explain why the network achieves worse/better accuracy than previously. Tip: To observe what is happening to the loss, you should change the plt.ylim argument.**



```
Final Test loss: 3.2117656088437125. Final Test accuracy: 0.8374. Average cross entropy loss on validation set: 5.234
```

The network achieves worse accuracy because when performing gradient descent, the gradient points in the steepest direction at the given point, but not necessarily to the local minima. Additionally when the Learning rate is too big, it can easily overshoot the local minima, which explains the suddenly higher train loss scores.

d) **[0.4pt] Include an hidden layer with 64 nodes in the network, with ReLU as the activation function for the first layer. Train this network with the same hyperparameters as previously. Plot the training and validation loss from this network together with the loss from task (a). Include the plot in your report. What do you observe?**
Learning rate: 0.0192



```
Final Orignal Test loss: 0.33633791579372563. Final Original Test accuracy: 0.9088
Final ReLU Test loss: 0.21712263930992337. Final ReLU Test accuracy: 0.9358
```

We can observe that combining all the improvements we have done so far with the ReLU hidden layer network resulted in the best score so far.