Image Processing and PyTorch - Report

Sivert Utne

November 15, 2021

Contents

Task 1 - Spatial Filtering

(a)

Sampling is converting a continous signal into a discrete signal, for instance digitizing a physical photo to virtual pixels.

(b)

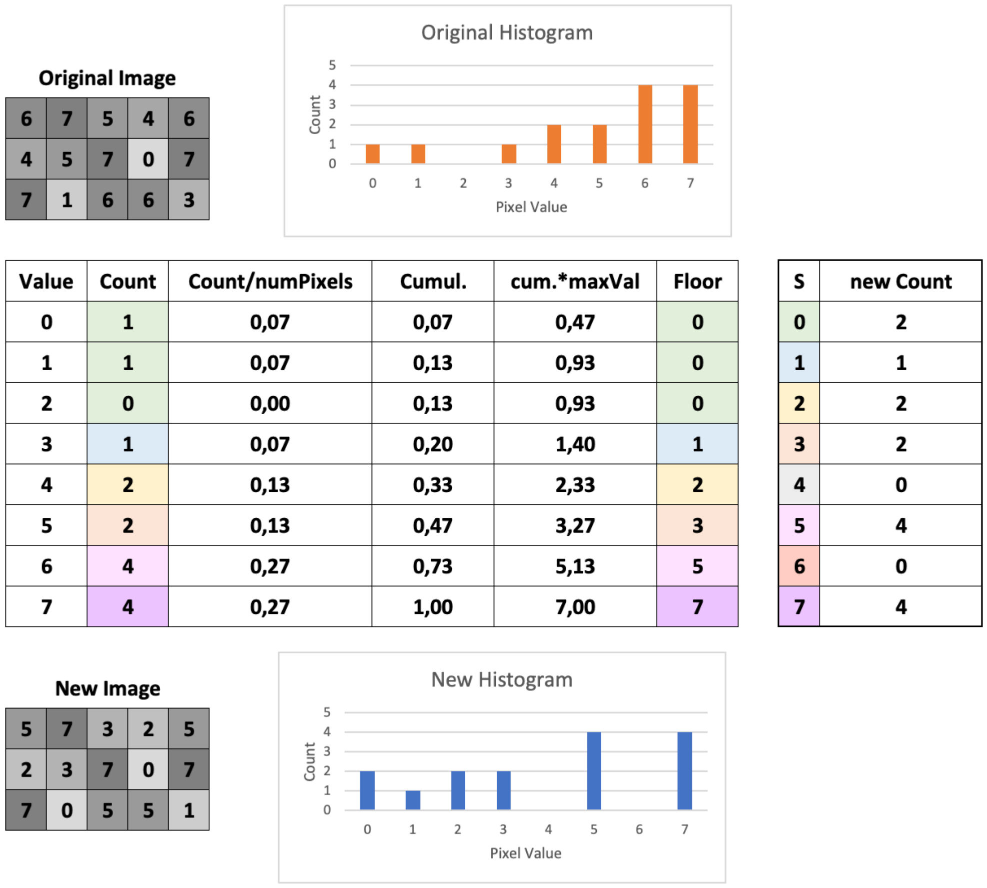
Quantization is mapping the values from the sampling to discrete values in a certain range, for instance to 8 bit color values.

(c)

By looking at an image histogram, you can identify the contrast level in the distribution of the values. An image with high contrast will have near linear cumulative histogram.

(d)

Figure 1 shows histogram equalization performed manually.



Original Image

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

Value	Count	Count/numPixels	Cumul.	cum.*maxVal	Floor
0	1	0,07	0,07	0,47	0
1	1	0,07	0,13	0,93	0
2	0	0,00	0,13	0,93	0
3	1	0,07	0,20	1,40	1
4	2	0,13	0,33	2,33	2
5	2	0,13	0,47	3,27	3
6	4	0,27	0,73	5,13	5
7	4	0,27	1,00	7,00	7

S	new Count
0	2
1	1
2	2
3	2
4	0
5	4
6	0
7	4

New Image

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

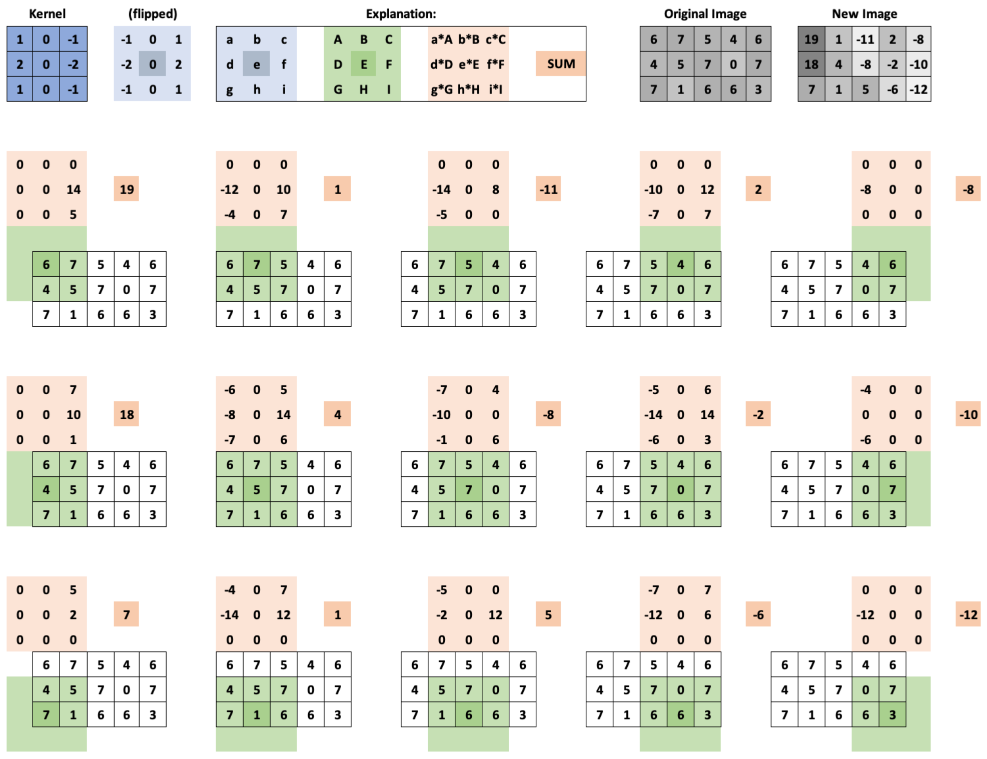
Figure 1: Histogram equalization

(e)

If we apply a log transform to an image with a large variance in pixel intensities the dynamic range will be compresses. This is because the log transform will map low values to a wider range, and high values to a smaller range.

(f)

The convolution is visualized in Figure 2, the multiplications and additions are not written out. The explanation part visualizes with colors how the values from the flipped kernel, and the respective area in the original image is multiplied and then summed together.



Kernel

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

(flipped)

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

Original Image

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

New Image

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

Figure 2: Kernel Applied Manually in Excel (pixels out of range are considered 0)

Task 2 - The Lake

(a)

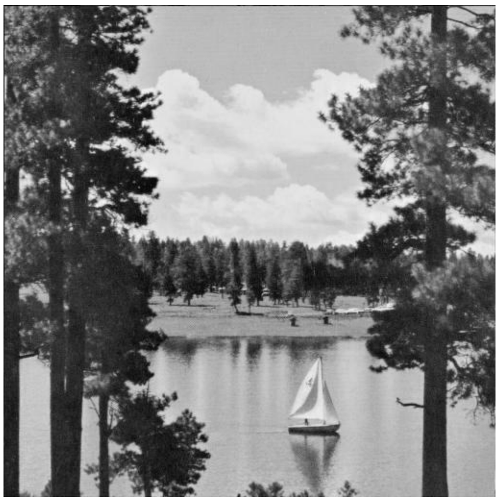


Figure 3: Grayscaled Image of the lake

(b)

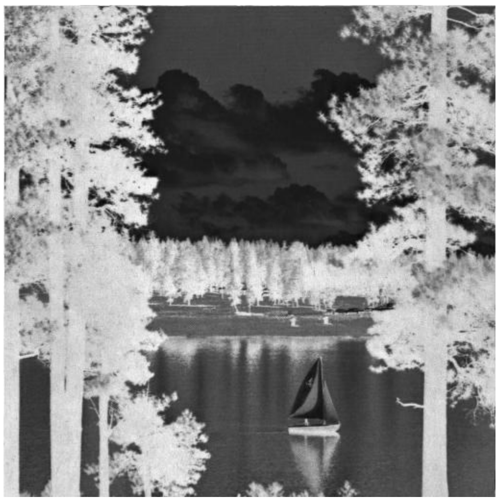
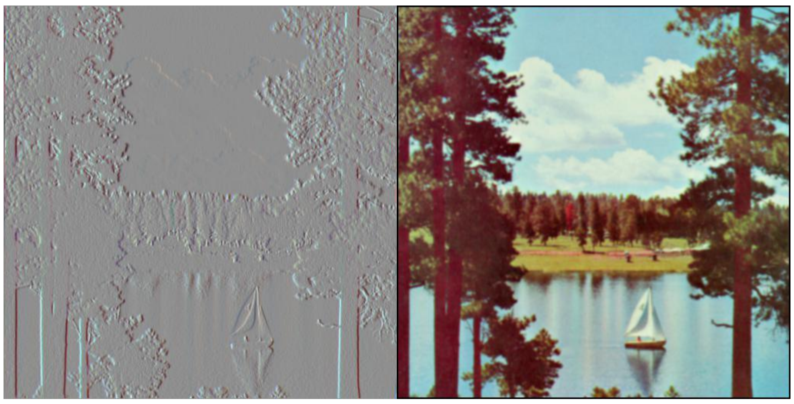


Figure 4: Invertion of the grayscaled image

(c)



(a) sobel kernel (b) smoothing kernel

Figure 5: The lake after applying the kernels.

Task 3 - Neural Networks

(a)

An example of what a single-layer neural network cannot represent is **XOR**. This is because XOR requires a non-linear function.

(b)

A hyperparameter is a parameter that defines the network in itself, or how it is trained. This can for instance be the number of layers, number of units per layer, learning rate etc.

(c)

Softmax is used to create a discrete probability distribution. This means that in the end the network has a number of values (between 0 and 1) that always add up to 1. This makes sure the model isn't thrown off by suddenly getting an extremely high value (that will make big changes in back propagation).

(d)

Performing forward pass on the network yields the following result (Figure 6):

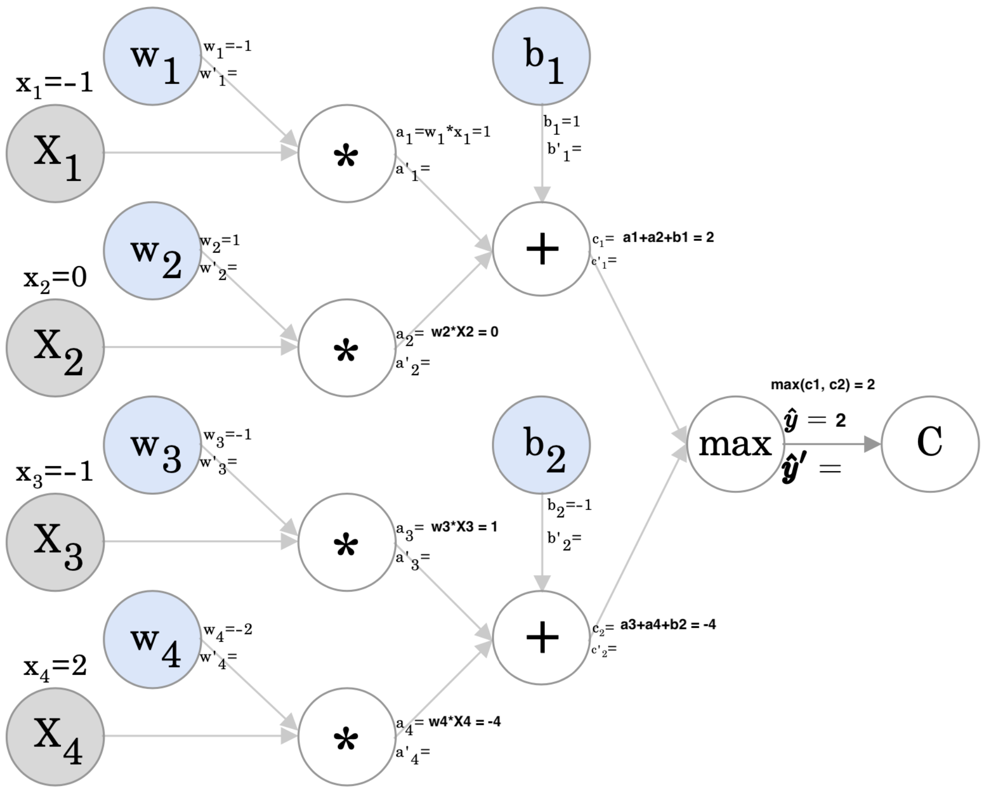


Figure 6: Network after Forward Pass

After that we can compute the final value of C and calculate the backwards pass. Using notation: $\frac{\vartheta C}{\vartheta x_n} = x'_n$

We have that: $y = 1$ and $C(y_n, \hat{y}_n) = \frac{1}{2}(y_n - \hat{y}_n)^2$ In other words we get:

$$C = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(1-2)^2 = \frac{1}{2}$$

From this ($C = \frac{1}{2}$) we get the following values:

$$\begin{aligned}
\hat{y}' &= -\hat{y} \cdot \frac{1}{2}(y - \hat{y}) && = -2 \cdot \frac{1}{2}(1-2) = 1 \\
c'_1 &= \hat{y}' \cdot \frac{\vartheta \hat{y}}{\vartheta c_1} && = 1 \cdot 1 = 1 \\
c'_2 &= 0 && \text{since } C = max(c_1, c_2) \text{ and } c_1 > c_2 \\
b'_1 &= c'_1 \cdot \frac{\vartheta c_1}{\vartheta b_1} && = 1 \cdot 1 = 1 \\
b'_2 &= c'_2 \cdot \frac{\vartheta c_2}{\vartheta b_2} && = 0 \cdot 1 = 0 \\
a'_1 &= c'_1 \cdot \frac{\vartheta c_1}{\vartheta a_1} && = 1 \cdot 1 = 1 \\
a'_2 &= c'_1 \cdot \frac{\vartheta c_1}{\vartheta a_2} && = 1 \cdot 1 = 1 \\
a'_3 &= c'_2 \cdot \frac{\vartheta c_2}{\vartheta a_3} && = 0 \cdot 1 = 0 \\
a'_4 &= c'_2 \cdot \frac{\vartheta c_2}{\vartheta a_4} && = 0 \cdot 1 = 0 \\
w'_1 &= a'_1 \cdot \frac{\vartheta a_1}{\vartheta w_1} && = 1 \cdot X_1 = -1 \\
w'_2 &= a'_2 \cdot \frac{\vartheta a_2}{\vartheta w_2} && = 1 \cdot X_2 = 0 \\
w'_3 &= a'_3 \cdot \frac{\vartheta a_3}{\vartheta w_3} && = 0 \cdot X_3 = 0 \\
w'_4 &= a'_4 \cdot \frac{\vartheta a_4}{\vartheta w_4} && = 0 \cdot X_4 = 0
\end{aligned}$$

Which leaves us with the following network in Figure 7:

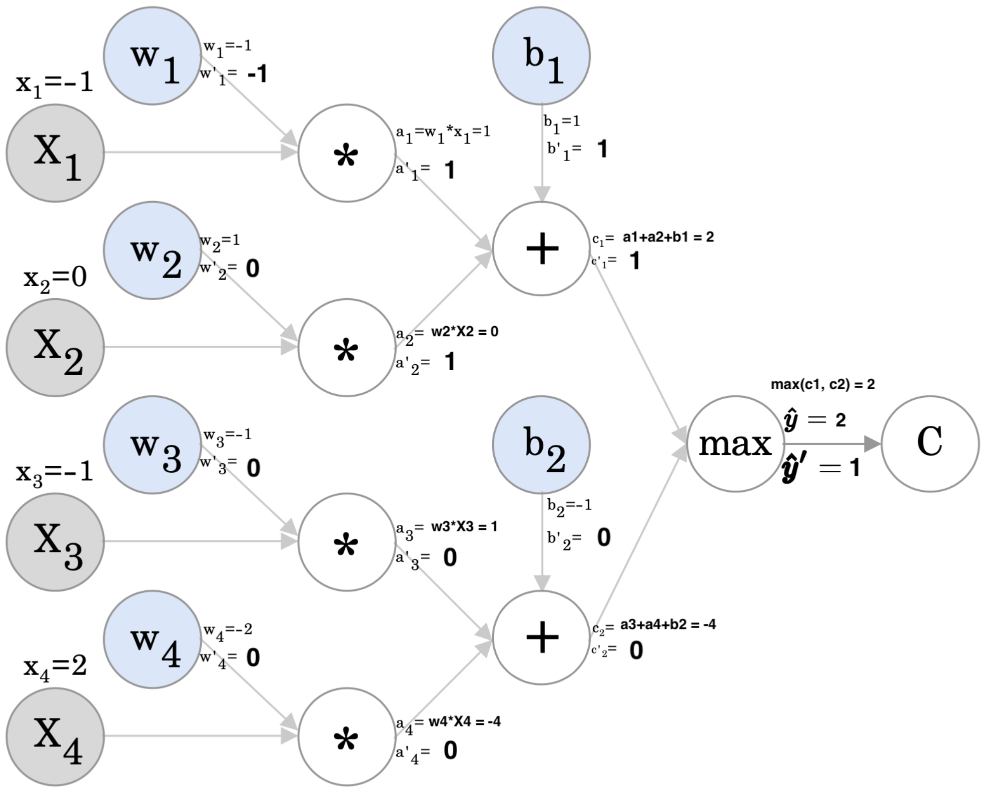


Figure 7: Network after Backward Pass

(e)

Updating the weights with $\alpha = 0.1$:

$$\begin{aligned}
w_1 = w_1 - \alpha \cdot w_1' &= -1 - 0.1 \cdot -1 = -0.9 \\
w_3 = w_3 - \alpha \cdot w_3' &= -1 - 0.1 \cdot 0 = -1 \\
b_1 = b_1 - \alpha \cdot b_1' &= 1 - 0.1 \cdot 1 = 0.9
\end{aligned}$$

Task 4 - MNIST

(a)

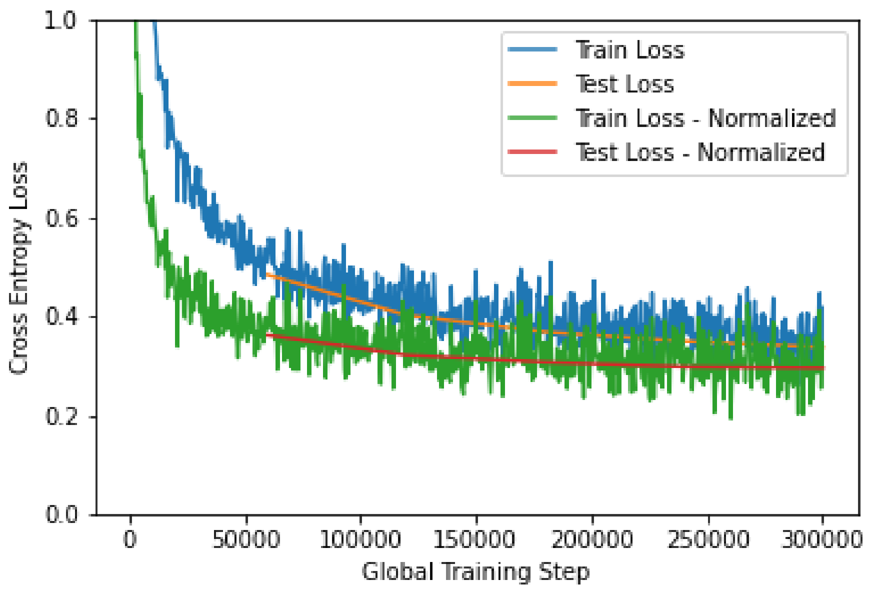


Figure 8: Normalized vs Not

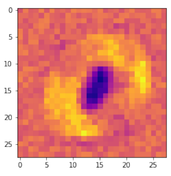
In Figure Figure 8 we see how the normalized images makes the model learn more quickly (as seen by the quick decrease in loss). Normalizing makes sure that all the images contain the same color-ranges, this consistency makes it easier for the model to recognize repeating patterns.

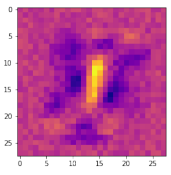
Table 1: Accuracy and Cross-Entropy Loss With and Without Normalization.

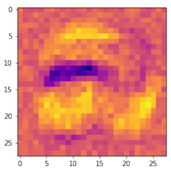
	Accuracy	Cross-Entropy Loss
Without Normalization	0.91	0.34
With Normalization	0.91	0.29
Difference	+0.0044	-0.0416

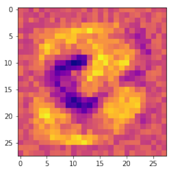
(b)

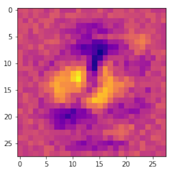
The weights are as seen below:

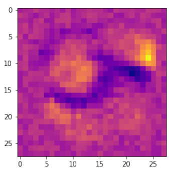

(a) 0

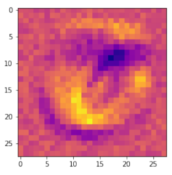

(b) 1

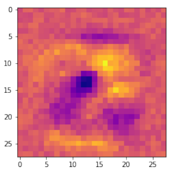

(c) 2

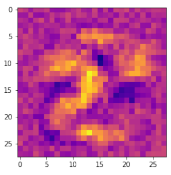

(d) 3


(e) 4


(f) 5


(g) 6


(h) 7


(i) 8

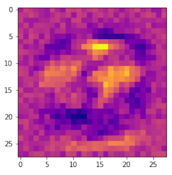

(j) 9

Figure 9: Model Weights After Training

What we are observing here is a kind of "heatmap" of where, for each of the 10 output variables, the weights put emphasis in the picture. For Figure 9b (Number 1), we can clearly see how the model highly favors pixels in the middle, while for Figure 9a (Number 0), it favors pixels NOT in the middle.

(c)

The results with learning rate of 1.0 is shown in figure Figure 10. The Learning rate tells the model how quickly it should change its values based on the results. We clearly see how the loss is moving wildly up and down. This is because the learning rate is too high and causes overfitting, where the model cannot converge on an optimal value because it changes too much.

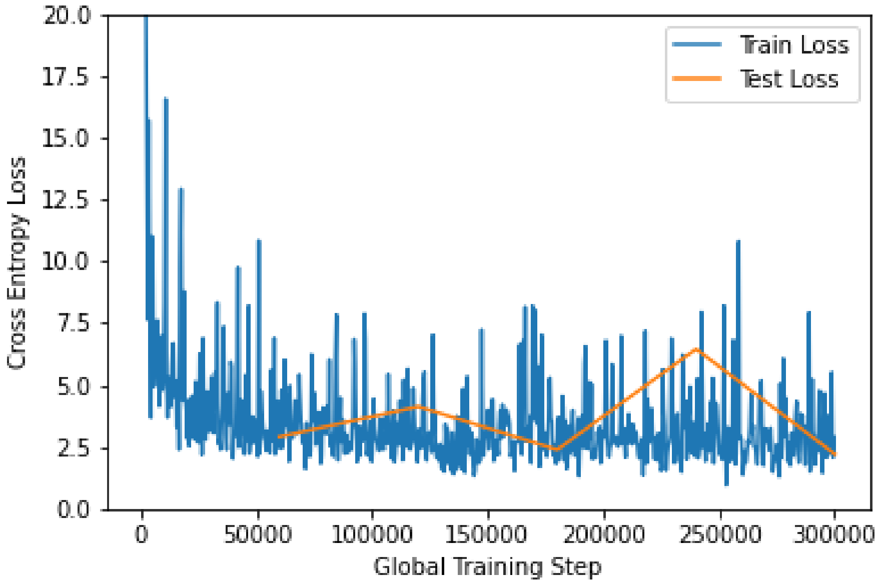


Figure 10: Learning Rate = 1

Table 2: Final Accuracy and Cross-Entropy Loss with Learning Rate = 1.0

Accuracy	Cross-Entropy Loss
0.89	2.21

(d)

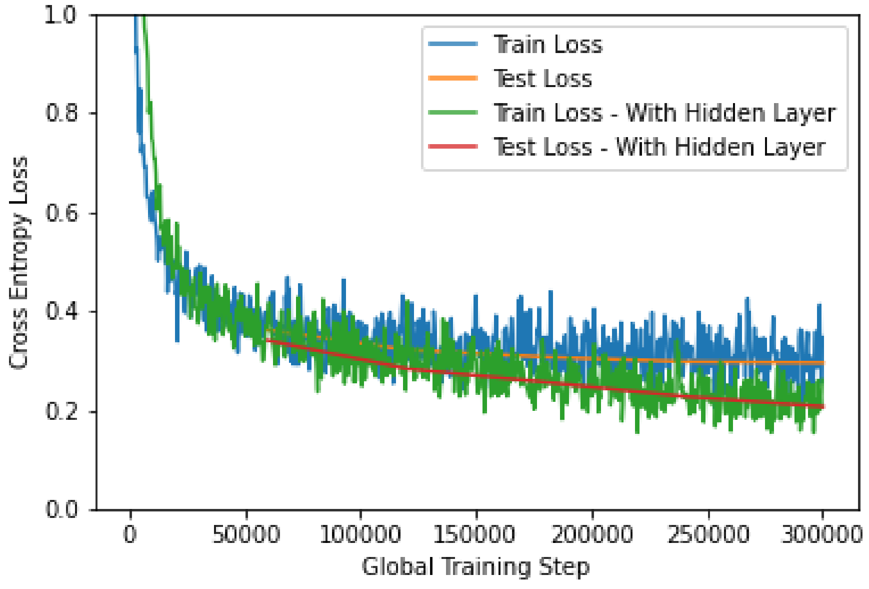


Figure 11: With vs Without Hidden Layer

Table 3: Final Accuracy and Cross-Entropy Loss With Hidden Layer

Accuracy	Cross-Entropy Loss
0.94	0.21

Comparing this directly to the loss and accuracy of the normalized model in Task A (Table 1), gives us:

Table 4: Difference in Accuracy and Cross-Entropy Loss for Task D and Task A

Accuracy	Cross-Entropy Loss
+0.0268	-0.0885

From the graph and the difference we see how the model with the hidden layer outperforms the model without it. It seems like the hidden layer allows the model to store more nuanced data (be more sensitive for small changes), where as in Task A the model quickly plateaus because of a lack of nodes.