Deep Learning Lab 2 Report

Shusrith S

PES1UG22AM155

January 26, 2025

1 Introduction

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used collection of 70,000 grayscale images of handwritten digits, each of size 28x28 pixels, representing the digits 0 through 9. It is divided into a training set of 60,000 images and a test set of 10,000 images, making it a standard benchmark for evaluating machine learning models, particularly for image classification tasks.

2 Architecture and Framework used

A feed-forward neural network is used to classify these images and identify the digit represented in the image. A ReLU activation function is used in the model with the Adam optimizer. The TensorFlow library is used for its extensive support with various regularization techniques built-in with the layers.

The layer classes also come with the ability to easily modify initialization methods for the second part of the report. The dataset is loaded from TensorFlow and used directly with TensorFlow's MODEL.FIT() method. A modular approach is used throughout, utilizing a single class and modifying the members of the class for each regularization and initialization technique.

Since comparison plots are needed in this report, TensorBoard has been used extensively. It keeps track of training and validation loss during the training process and creates interactive plots and graphs. It also allows comparison between various runs, a key objective of this report.

All training was done on an Nvidia RTX 4050 laptop GPU. The hyperparameters maintained for all regularization techniques are learning rate of **0.001**, batch size of **512** and **20 epochs** each. The loss being used is **sparse categorical cross entropy**, because of the softmax activation applied to the final layer, considering only the positive class for loss.

3 Regularization Techniques

3.1 No Regularization

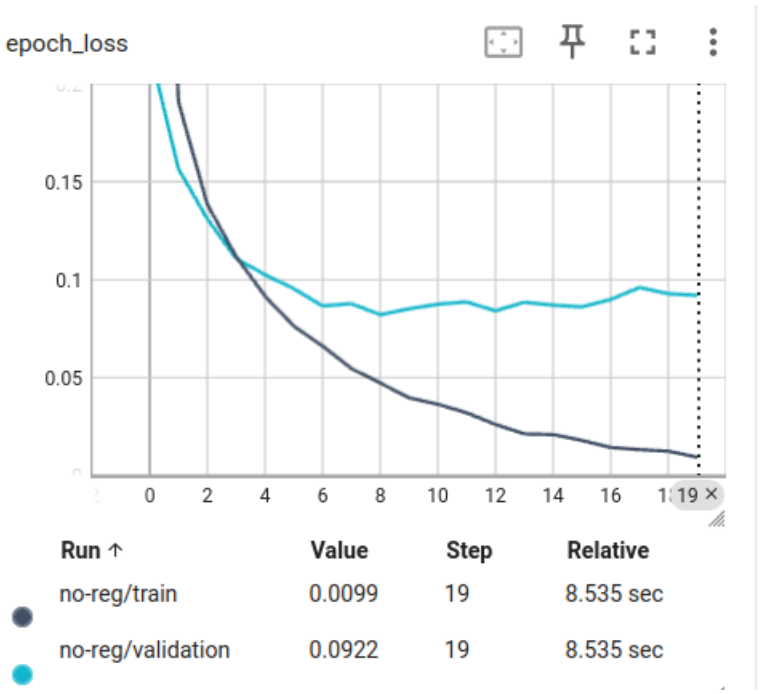

Figure 1: No regularization used

Loss Type	Loss Value
Train Loss	0.0.0099
Validation Loss	0.0922

Table 1: Train and Validation Loss Values

- Initially, no regularization technique is used.
- We can see that after 4 epochs, the validation loss starts to deviate from the train loss and in the end there is a significant deviation between the two.
- Even though the scale of the deviation is low, there is a significant amount to suggest that regularization is needed to reduce overfitting and improve the performance of the model.

3.2 L1 regularization

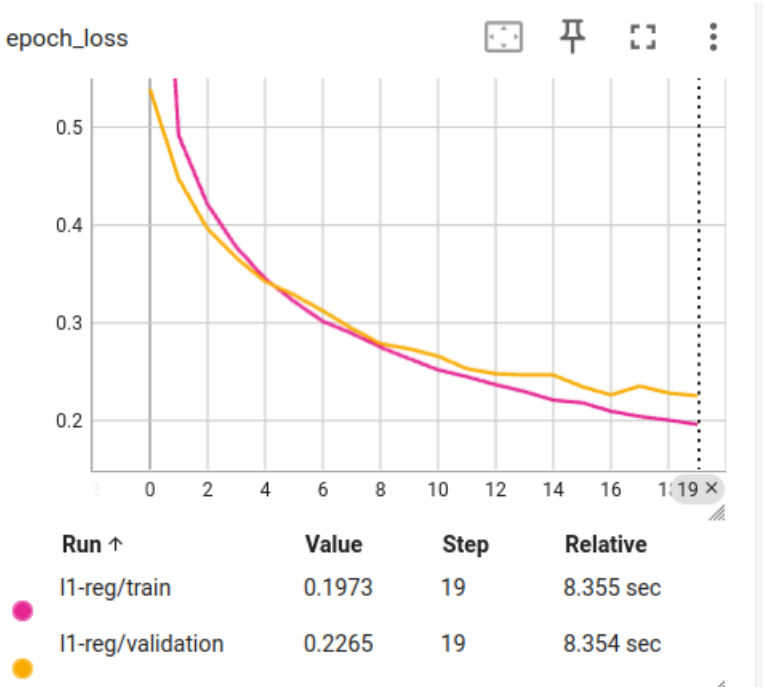

Figure 2: L1 regularization

Loss Type	Loss Value
Train Loss	0.1973
Validation Loss	0.2265

Table 2: Train and Validation Loss Values

- L1 regularization proved to be the best regularization method overall in terms of ensuring that train loss and val loss had similar or close values.
- Since L1 drives some of the weights to zero, it can be concluded that the presence of all neurons might have been causing the model to overfit and the selection that L1 performs proved to be effective in reducing the overfitting.
- It should also be noted that L1 had a much higher numerical value of loss in both training and validation.
- The selection of neurons implies that L1 requires more epochs to reach the same level of accuracy as some of the methods used.

3.3 L2 regularization

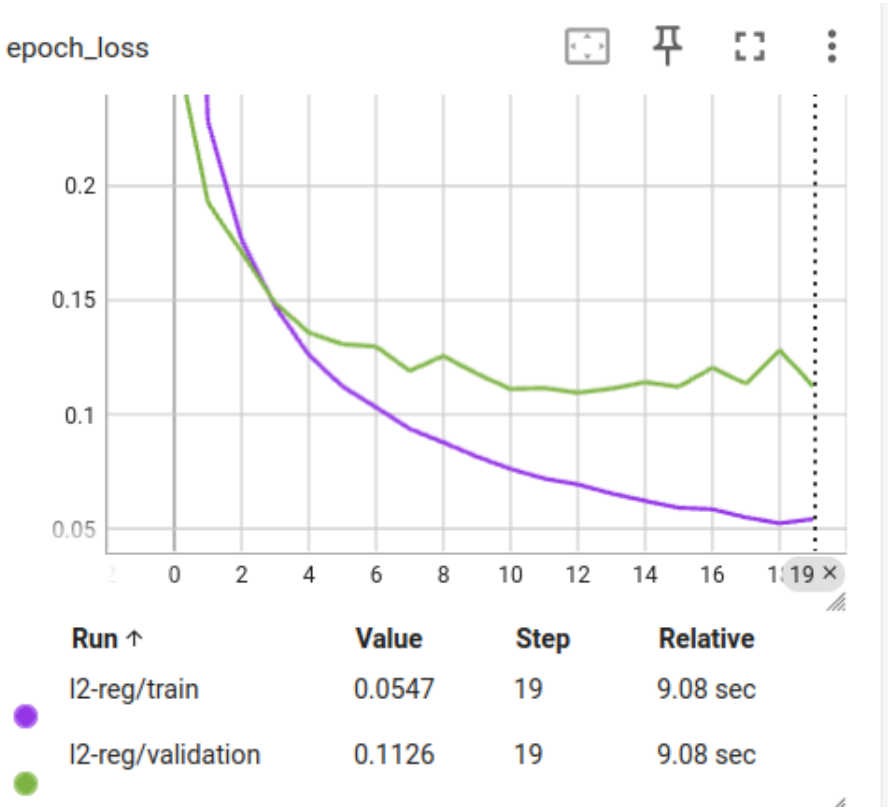

Figure 3: L2 regularization

Loss Type	Loss Value
Train Loss	0.0547
Validation Loss	0.1126

Table 3: Train and Validation Loss Values

- With L2 regularization, a slight deviation is seen in the curves of train and val loss, but the difference between the two is lower than it is when no regularization is used.
- The values of the loss is also much lower in comparison to L1 loss.
- It can be concluded that L2 doesn't penalize as strictly as L1 but also manages to reduce the degree of overfitting to a good extent.

3.4 Data Augmentation

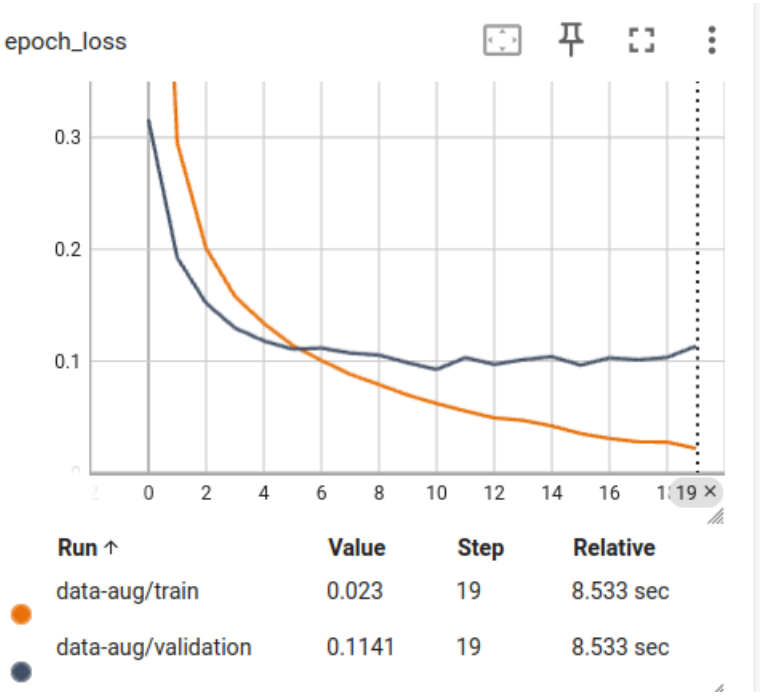

Figure 4: Data Augmentation

Loss Type	Loss Value
Train Loss	0.023
Validation Loss	0.1141

Table 4: Train and Validation Loss Values

- Data augmentation was done using TensorFlow's inbuilt RandomTranslation, RandomZoom, and RandomRotation functions.
- Translation and zoom were randomized at 10% and rotation was randomized to 10°.
- The graphs show us that this wasn't a particularly effective method to prevent overfitting in this scenario, given the large deviation in train and val loss curves.
- It is also possible that since data augmentation gives us variation in the samples, more epochs might be necessary to see the results that we desire and 20 are not sufficient.

3.5 Input noise

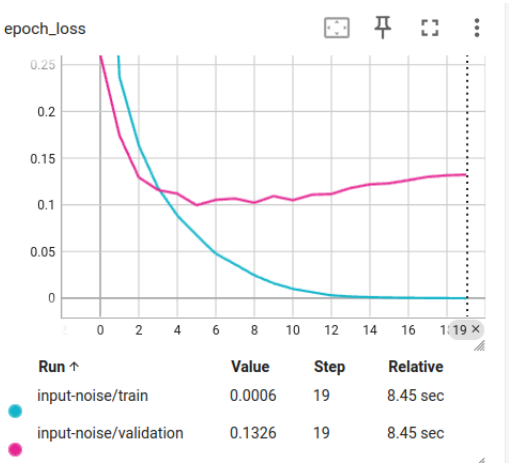

Figure 5: Input noise with variance 0.2

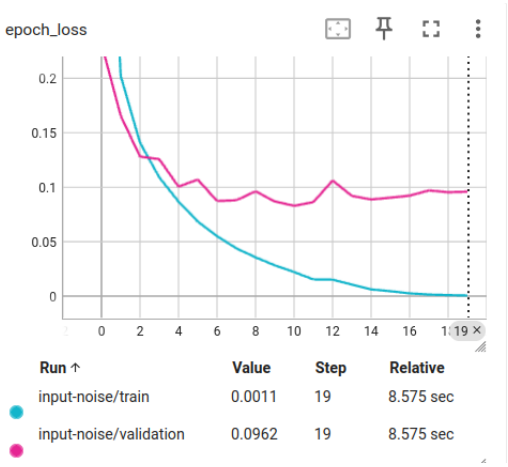

Figure 6: Input noise with variance 0.05

Loss Type	Loss Value
Train Loss	0.0006
Validation Loss	0.1326

Table 5: Variance 0.2

Loss Type	Loss Value
Train Loss	0.0006
Validation Loss	0.1326

Table 6: Variance 0.05

- Noise was added to the inputs, drawn from a gaussian distribution of mean 0 and variance 0.2.
- The graph shows us that train loss is extremely low and val loss is significantly higher.
- In this example, this occurs mainly because pixel values have been normalized between 0 and 1.
- As a result, when we add noise drawn from a distribution of mean 0 and variance 1 only to the train data and not to the val data, it can significantly offset the training data.
- Hence the model can learn to fit the noise in the train data but does not perform well enough on the test data.
- Adding noise to input can be used in different kinds of problems, that do not involve training data normalized between 0 and 1.
- It was also seen that when the variance was reduced to 0.05, the val loss came down to a lower value.

3.6 Output noise

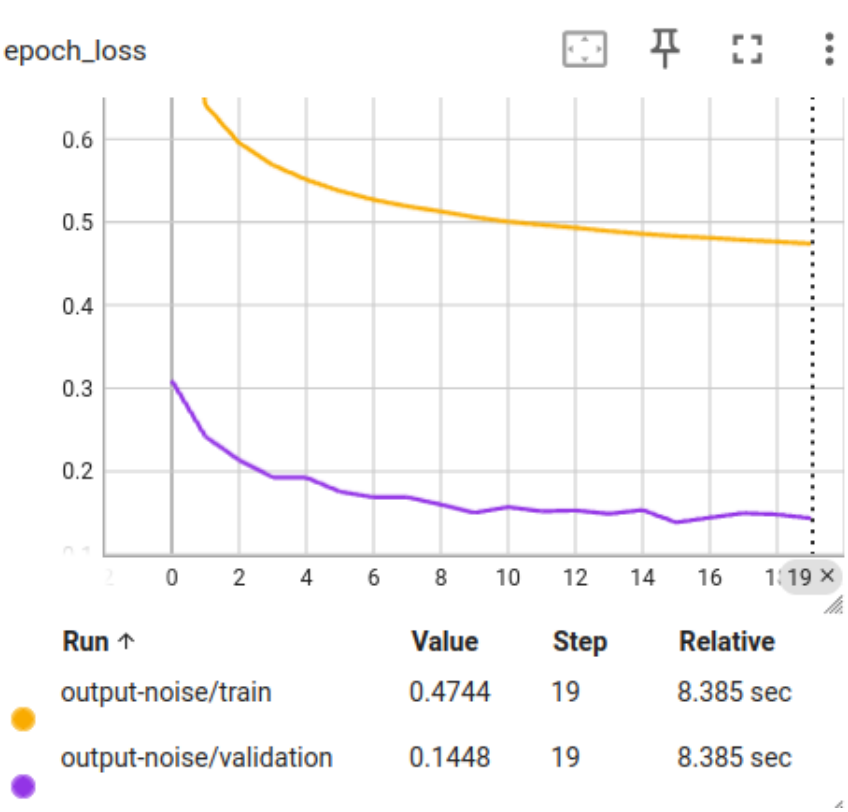

Figure 7: Output noise

Loss Type	Loss Value
Train Loss	0.4744
Validation Loss	0.1448

Table 7: Train and Validation Loss Values

- Output noise is added to the labels by first one-hot encoding all the train labels and then drawing some noise value from a gaussian distribution of mean 0 and variance 0.1.
- This noise is then subtracted from the label having value 1 and the noise divided by 9 is added to all the other labels having value 0.
- This is done only for the train data and not to the val data.
- We can see in the graph that the value of train loss is much higher than val loss.
- This is because categorical cross entropy is used as the loss function instead of sparse categorical cross entropy.
- As a result, the added noise all contribute to the loss value, resulting in a higher training loss.
- We can also see that val loss reduces uniformly without any increase, showing that there is no overfitting in the model.
- Adding noise to the output also has a validation accuracy of 0.98, the highest of all techniques used so far. Hence, we can conclude that this is a very effective method of regularization for this problem.

3.7 Dropout

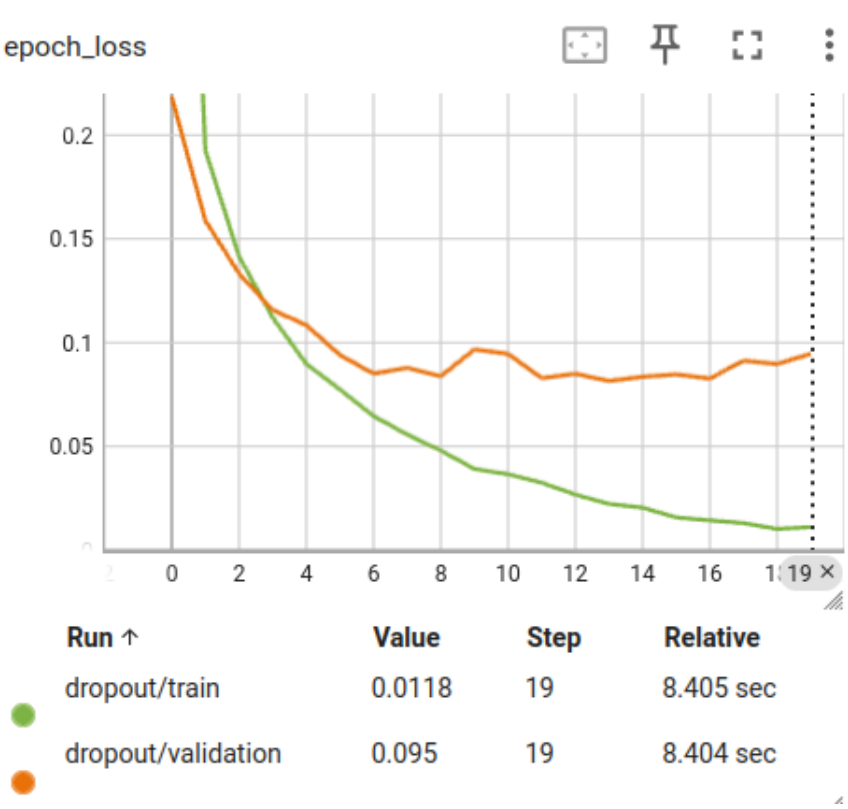

Figure 8: Dropout

Loss Type	Loss Value
Train Loss	0.0118
Validation Loss	0.095

Table 8: Train and Validation Loss Values

- Using a dropout of 50%, we can see that there is still a large deviation in the curves of train and val loss.
- This means that the dropout does not work particularly well in this example, possibly due to the low model complexity.
- Only three layers are used, each with 100 neurons, so usage of dropout does not really add any benefit in this scenario.
- Dropout tends to have more effect in deeper networks with more neurons and hence we can see a lack of improvement in this scenario.

4 Initialization Techniques

4.1 Zero initialization

Figure 9: Zero initialization

Loss Type	Loss Value
Train Loss	2.301
Validation Loss	2.301

Table 9: Train and Validation Loss Values

- Zero weight initialization prevents learning because all neurons in a layer receive identical gradients during backpropagation, causing them to update in the same way.
- This symmetry prevents the network from learning different features.
- This can be seen from the graph where both train and val loss are a single straight line that does not improve.

4.2 Ones initialization

Figure 10: Ones initialization

Loss Type	Loss Value
Train Loss	6223
Validation Loss	6089

Table 10: Train and Validation Loss Values

- Initializing weights with ones leads to very high loss because it causes the network to struggle with gradient updates.
- When all weights are set to one, the activations of neurons become very large, leading to excessively large gradients during backpropagation.
- This can cause the model to overshoot optimal weight values and result in poor convergence, resulting in a very high loss.

4.3 Random initialization

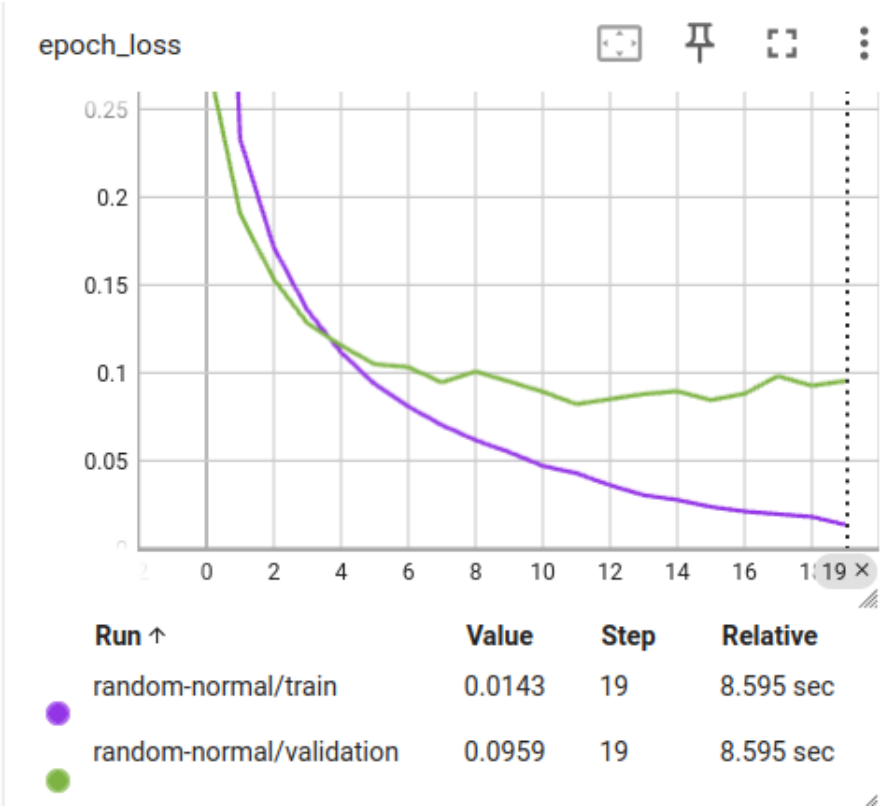

Figure 11: Random initialization

Loss Type	Loss Value
Train Loss	0.0143
Validation Loss	0.0959

Table 11: Train and Validation Loss Values

- Random initialization is preferred over ones or zeroes because it breaks the symmetry between neurons, allowing each neuron to learn different features.
- While zero or ones initialization leads to identical gradients and prevents effective learning, random initialization ensures that each neuron starts with unique weights, enabling the network to learn diverse patterns.
- This diversity helps the model converge properly and reduces the risk of high loss, facilitating better performance during training.

4.4 Glorot Normal initialization

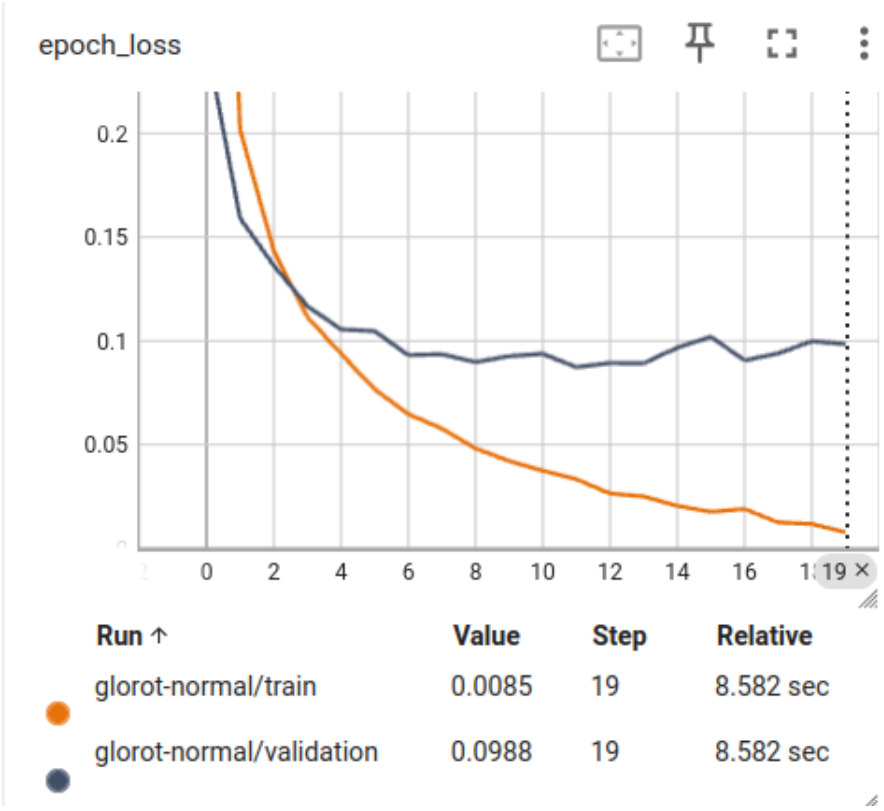

Figure 12: Glorot Normal initialization

Loss Type	Loss Value
Train Loss	0.0085
Validation Loss	0.0988

Table 12: Train and Validation Loss Values

- Glorot normal initialization improves upon random initialization by scaling weights based on the number of input and output neurons.
- Unlike purely random initialization, which can lead to exploding or vanishing gradients, Glorot normal ensures gradients remain stable, improving convergence and performance.
- This makes it more effective, especially for deep networks, compared to standard random initialization.
- This explains the lower train and validation loss in the model.

4.5 He Normal initialization

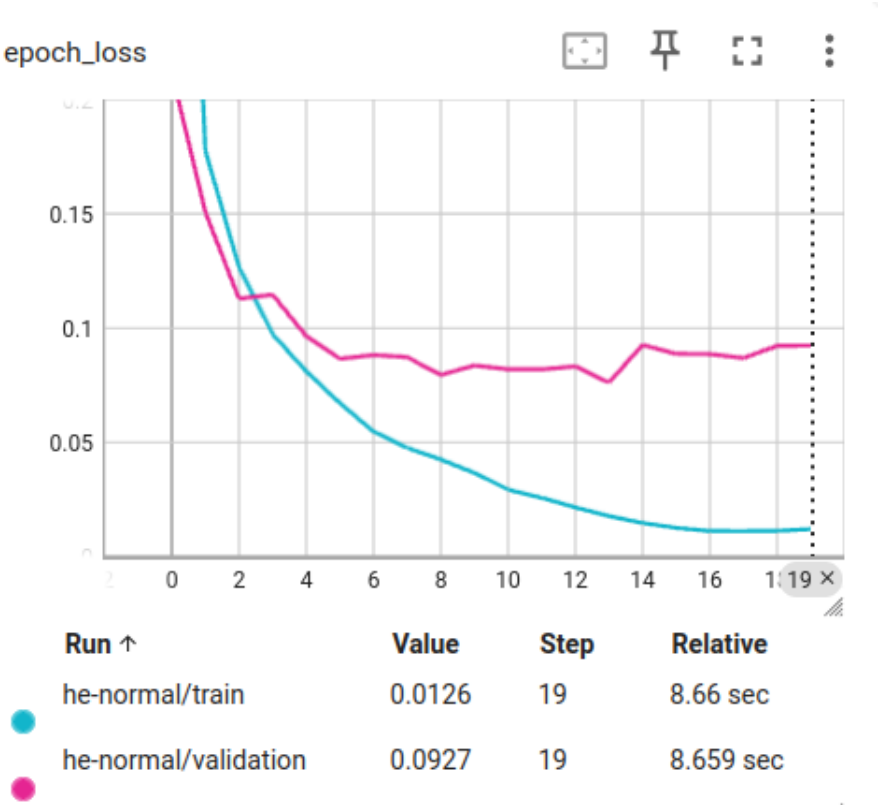

Figure 13: He Normal initialization

Loss Type	Loss Value
Train Loss	0.0126
Validation Loss	0.0927

Table 13: Train and Validation Loss Values

- He normal initialization is superior to random initialization for ReLU activations as it scales weights based on the number of input neurons, preventing exploding or vanishing gradients.
- Unlike random initialization, which risks unstable gradients, He normal ensures smoother convergence and better performance, especially in deep networks.