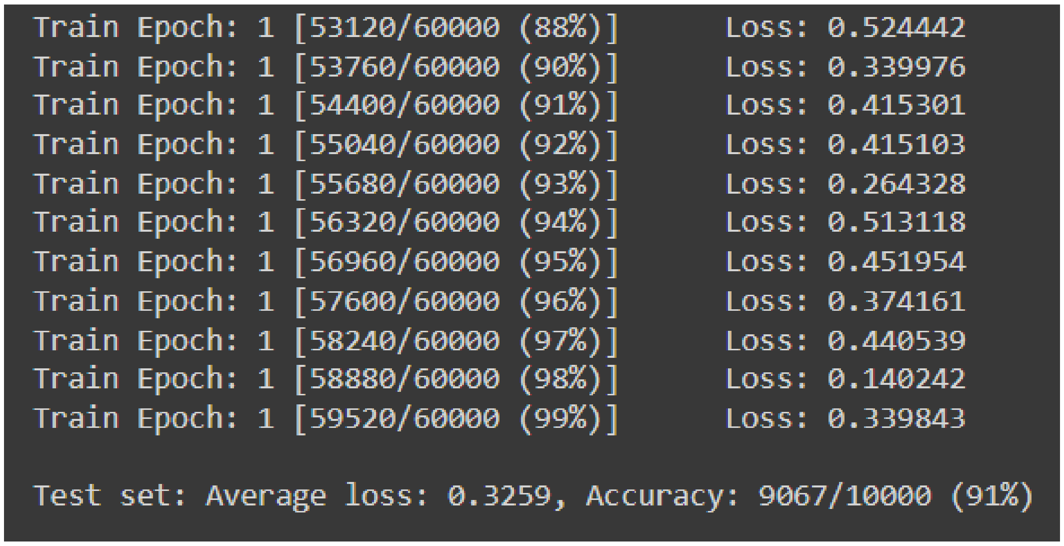
Shaun Noronha

Homework 4

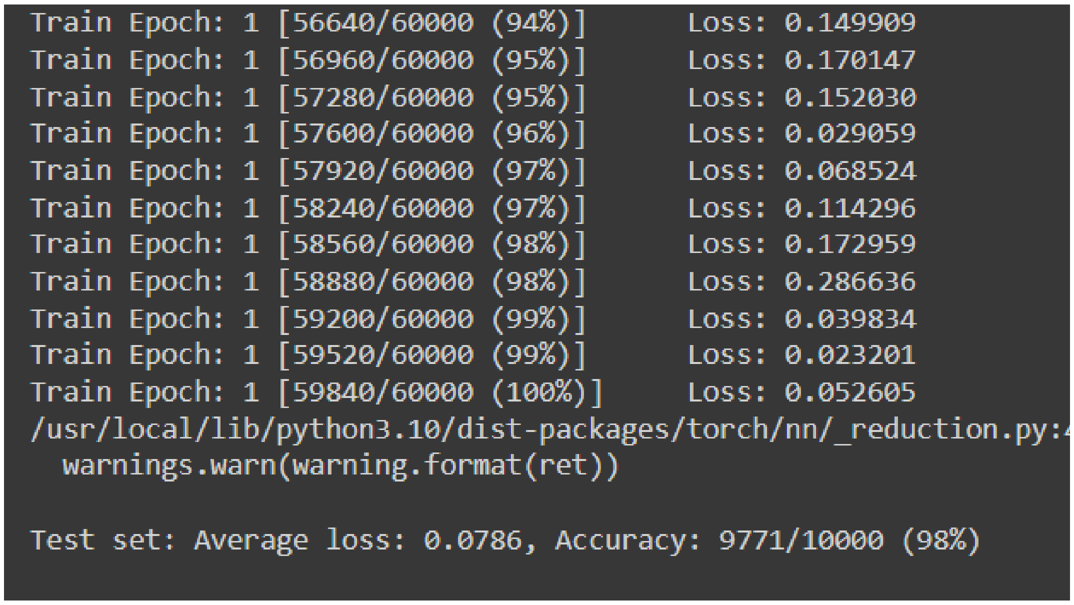
CS 539 – Machine Learning

The addition of a convolutional neural network (CNN) to the neural network architecture enhances its ability to understand and analyze images. In our model, we add CNN layers by first applying filters to the input images, which help identify important patterns. Then, pooling layers reduce the size of the features while preserving their essential information. Afterward, the flattened output is fed into traditional fully connected layers for classification. This sequential setup allows the CNN to effectively learn and recognize patterns in images, improving the model's accuracy in tasks like image classification. Following were the observations



```
Train Epoch: 1 [53120/60000 (88%)]      Loss: 0.524442
Train Epoch: 1 [53760/60000 (90%)]      Loss: 0.339976
Train Epoch: 1 [54400/60000 (91%)]      Loss: 0.415301
Train Epoch: 1 [55040/60000 (92%)]      Loss: 0.415103
Train Epoch: 1 [55680/60000 (93%)]      Loss: 0.264328
Train Epoch: 1 [56320/60000 (94%)]      Loss: 0.513118
Train Epoch: 1 [56960/60000 (95%)]      Loss: 0.451954
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.374161
Train Epoch: 1 [58240/60000 (97%)]      Loss: 0.440539
Train Epoch: 1 [58880/60000 (98%)]      Loss: 0.140242
Train Epoch: 1 [59520/60000 (99%)]      Loss: 0.339843

Test set: Average loss: 0.3259, Accuracy: 9067/10000 (91%)
```

<u>Without CNN</u>

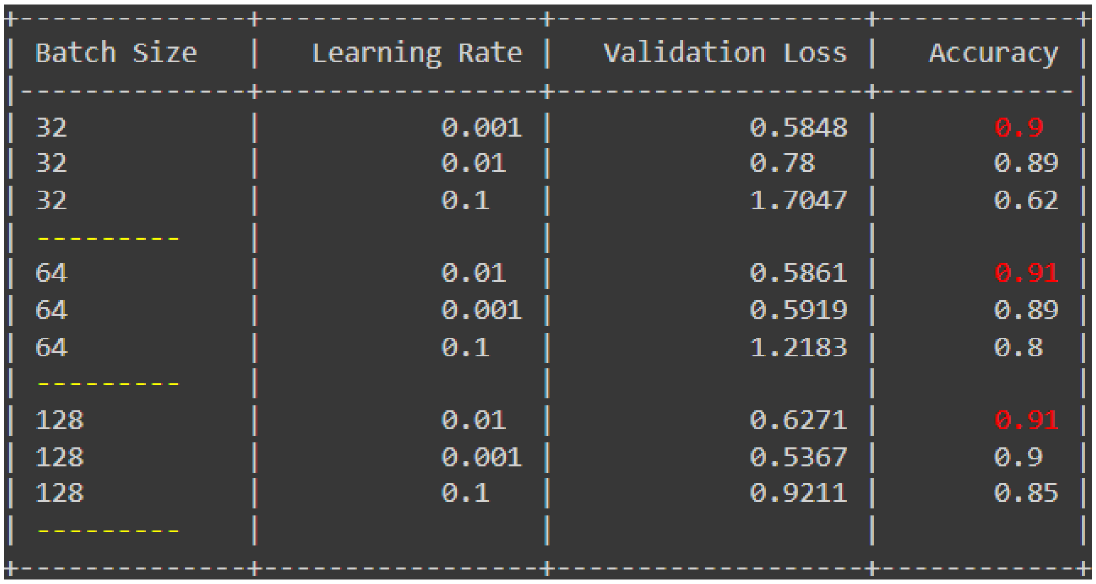

```
Train Epoch: 1 [56640/60000 (94%)]      Loss: 0.149909
Train Epoch: 1 [56960/60000 (95%)]      Loss: 0.170147
Train Epoch: 1 [57280/60000 (95%)]      Loss: 0.152030
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.029059
Train Epoch: 1 [57920/60000 (97%)]      Loss: 0.068524
Train Epoch: 1 [58240/60000 (97%)]      Loss: 0.114296
Train Epoch: 1 [58560/60000 (98%)]      Loss: 0.172959
Train Epoch: 1 [58880/60000 (98%)]      Loss: 0.286636
Train Epoch: 1 [59200/60000 (99%)]      Loss: 0.039834
Train Epoch: 1 [59520/60000 (99%)]      Loss: 0.023201
Train Epoch: 1 [59840/60000 (100%)]     Loss: 0.052605
/usr/local/lib/python3.10/dist-packages/torch/nn/_reduction.py:
  warnings.warn(warning.format(ret))

Test set: Average loss: 0.0786, Accuracy: 9771/10000 (98%)
```

<u>With CNN layer</u>

It can be inferred that introducing a convolutional neural network (CNN) significantly improved the model's performance. Without CNN, the accuracy on the test set was **91%**, while with CNN, the accuracy increased to **98%**. This demonstrates the effectiveness of CNNs in extracting and learning features from image data, leading to better classification accuracy. The lower average loss with CNN indicates that the model's predictions are closer to the ground truth labels, further supporting the notion that CNNs enhance the model's ability to understand and analyze image data.

To determine the best configuration from the provided experiments, let's analyze the results:

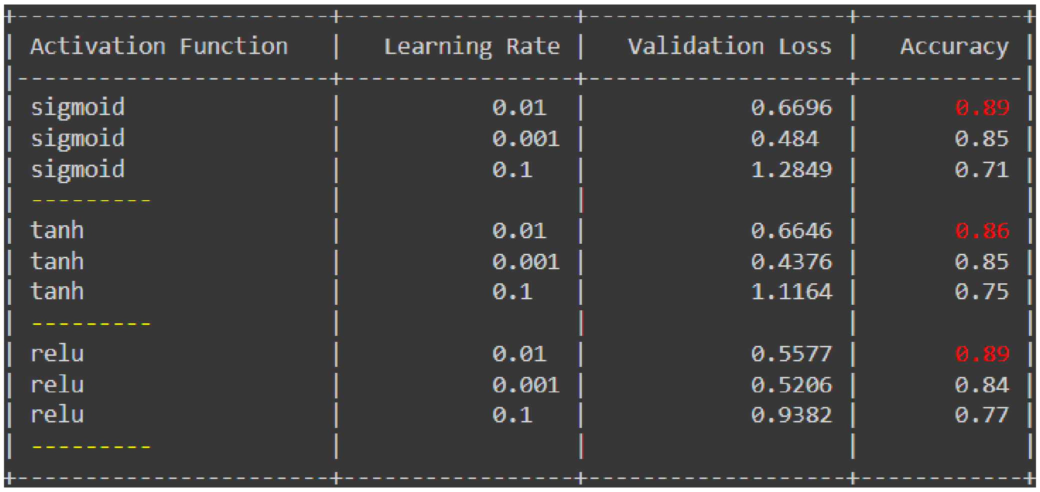
For batch size and learning rate experiments:



```
+--------------+----------------+------------------+------------+
| Batch Size   |  Learning Rate |  Validation Loss |   Accuracy |
|--------------+----------------+------------------+------------|
| 32           |          0.001 |           0.5848 |        0.9 |
| 32           |          0.01  |           0.78   |       0.89 |
| 32           |          0.1   |           1.7047 |       0.62 |
| ---------    |                |                  |            |
| 64           |          0.01  |           0.5861 |       0.91 |
| 64           |          0.001 |           0.5919 |       0.89 |
| 64           |          0.1   |           1.2183 |        0.8 |
| ---------    |                |                  |            |
| 128          |          0.01  |           0.6271 |       0.91 |
| 128          |          0.001 |           0.5367 |        0.9 |
| 128          |          0.1   |           0.9211 |       0.85 |
| ---------    |                |                  |            |
+--------------+----------------+------------------+------------+
```

Batch Size: 64
Learning Rate: 0.01
Validation Loss: 0.5861
Accuracy: 0.9100

For activation function and learning rate experiments:



```
+-----------------------+----------------+------------------+------------+
| Activation Function   |  Learning Rate |  Validation Loss |   Accuracy |
|-----------------------+----------------+------------------+------------|
| sigmoid               |          0.01  |           0.6696 |       0.89 |
| sigmoid               |          0.001 |           0.484  |       0.85 |
| sigmoid               |          0.1   |           1.2849 |       0.71 |
| ---------             |                |                  |            |
| tanh                  |          0.01  |           0.6646 |       0.86 |
| tanh                  |          0.001 |           0.4376 |       0.85 |
| tanh                  |          0.1   |           1.1164 |       0.75 |
| ---------             |                |                  |            |
| relu                  |          0.01  |           0.5577 |       0.89 |
| relu                  |          0.001 |           0.5206 |       0.84 |
| relu                  |          0.1   |           0.9382 |       0.77 |
| ---------             |                |                  |            |
+-----------------------+----------------+------------------+------------+
```

Activation Function: tanh
Learning Rate: 0.001
Validation Loss: 0.4376
Accuracy: 0.8500

Based on these results, the best configuration is:

Activation Function: tanh
Batch Size: 64
Learning Rate: 0.01
.

What we learned from the experiments:

- The choice of activation function significantly impacts the model's performance, with the tanh activation function outperforming relu and sigmoid in terms of validation loss and accuracy.
- A batch size of 64 seems to work well across different activation functions and learning rates.
- A moderate learning rate of 0.01 generally performs better compared to lower or higher learning rates, indicating the importance of proper learning rate selection.
- Validation loss and accuracy are essential metrics for evaluating model performance and guiding hyperparameter tuning decisions.