1. Problem Statement

In the tutorial, we did BPNN (Back-Propagation Neural Network), CNN (Convolutional Neural Network) and RNN (Recursive Neural Network) using MINST datasets to detect handwritten digits. On this specific problem, we will use the same problem, which is detecting handwritten digits. However, on this problem, rather than using MNIST datasets, we will use EMNIST datasets.

Extended MNIST (EMNIST) is made from the NIST special database. The datasets that we are using are directly compatible with the original MNIST dataset. However, on this problem, the dataset is extended (Bhattacharyya, 2020).

EMNIST has six different splits of datasets: ByClass, ByMerge, Balanced, Letters, Digits, and MNIST. On this problem, we will use the EMNIST split by MNIST dataset. This dataset contains handwritten digit datasets directly compatible with the original MNIST dataset used in Tutorial Lab 3. EMNIST MNIST dataset has 70,000 characters with ten balanced classes (Cohen et al., 2017).

The main difference between MNIST and EMNIST-MNIST datasets is that some of the numbers are rotated, and some are not rotated. This problem may reduce agent accuracy due increase in complexity in the datasets.

2. Selection of AI technique

On this problem, two options are suggested to solve this problem. The first one is the Back Propagation Neural Network (BPNN), the other is Convolutional Neural Network (CNN). On selecting the AI technique, we will test the bot technique in the same situation. Each of the AI techniques will have EPOCH = 1 and Learning Rate = 0.001. Moreover, all the setups will be the same according to the Tutorial week 3 Lab. The previous setup can be used because MNIST and EMNIST MNIST datasets are compatible with each other.

There will be three tests conducted. There are two main factors that we can see, which are loss and accuracy. Both variables are very important to get a good AI agent to be used in real life. To have a better testing environment, each test will require the user to restart the jupyter session before having the test session.

Accuracy is a measurement of classification model performance—accuracy counts whether the predicted value is equal to true. While a loss is a probability of uncertainty of prediction based on prediction varies from the true values. All agent training goal is to minimize loss.

		BPNN	*CNN*
First Test	**Accuracy**	96.47 %	97.45 %
	Loss	0.124	0.407
Second Test	**Accuracy**	96.67 %	98.02 %
	Loss	0.117	0.398
Third Test	**Accuracy**	96.55 %	97.59 %
	Loss	0.120	0.386
Total	**Accuracy**	289.69%	293.06%
	Loss	0.361	1.191
Average	**Accuracy**	96.563 %	97.687 %
	Loss	0.120	0.397

After testing, we can see that both BPNN and CNN can do their job pretty well. Both of them have high accuracy and low loss, which can be used to solve this problem. However, there will only be one neural network to be chosen. BPNN has lower accuracy, which ends up with an average of 96.5 %, while CNN has a higher accuracy of 97.7 %. On the other hand, BPNN has a very low loss = 0.12, while CNN has a higher loss value of nearly 0.4.

Considering the gap between 2 variables between BPNN and CNN, we can see a big gap between the two techniques used. If we see the accuracy difference, it has a slight difference between both AI techniques; however, if checked based on the Loss value, BPNN won over CNN with a considerable amount with a difference of nearly 0.28 value.

Based on the data gathered after testing, we can see that both techniques can be used and do pretty well in solving this matter. However, BPNN still has better performance.

3. Explanation of AI technique

Back-Propagation Neural Networks (BPNN)

Neural network based on the error rate obtained in the previous epoch or iteration. Back Propagation connects the information with an error that is generated when a guess is made. This method reduces the error rate, which refers to loss function (Abhishek, 2020).

There are three types of layers in Neural Network: Input layer, Hidden layer, and Output Layer. BPNN usually consist of 4 layers which are 1 x input layer, 2 x hidden layer, and 1 x output layer.

The input layer will receive the input from the preconnected path. Then Input layer will deal with customizing the weight of the input. The weight of the input is selected randomly.

Output is calculated for every neuron from the input layer. Errors will be evaluated from the output. In order to reduce the error rate, weight adjustments can be made. Weight adjustment can be made by moving back from the hidden layer to the output layer. Then the agent will repeat the process until found desired output.

The main advantages of BPNN are the AI technique is fast and simple. This AI technique is also flexible and works efficiently. The disadvantage of this AI technique is, very sensitive to noisy data. Depending on the data, there is a chance for performance issues ("Backpropagation neural network : Types, advantages & disadvantages," 2021).

4. Solution Result

This AI agent can be used to detect handwritten digits without caring about the angle of the digits. Using EMNIST (Extended MNIST), it has handwritten digits that are rotated around to increase the complexity. Datasets that are used are EMNIST MNIST. This dataset has 70,000 characters with ten balanced classes.

To solve this problem, I decided to go with BPNN. BPNN can provide accuracy of around 96.6 % and has a low Loss value of around 0.12. CNN can also be used to solve this problem. However, after testing both AI techniques, BPNN still has better performance with a lower loss value. However, CNN has better performance in terms of agent accuracy.

5. References

Abhishek, G. (2020, July 17). *Difference between ANN, CNN and RNN*. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-ann-cnn-and-rnn/

Backpropagation neural network : Types, advantages & disadvantages. (2021, July 15). WatElectronics.com. https://www.watelectronics.com/back-propagation-neural-network/

Bhattacharyya, J. (2020, November 10). *6 MNIST image datasets that data scientists should be aware of (With Python implementation)*. Analytics India Magazine. https://analyticsindiamag.com/mnist/

Cohen, G., Afshar, S., Tapson, J., & Van Schaik, A. (2017, April 4). *The EMNIST dataset*. NIST. https://www.nist.gov/itl/products-and-services/emnist-dataset

6. Screenshot

a. BPNN

i. First Test

```
In [11]: trained_model = train_model(net_work, train_loader, LR, EPOCH)
```

```
[1,100] loss:0.757
[1,200] loss:0.281
[1,300] loss:0.228
[1,400] loss:0.222
[1,500] loss:0.188
[1,600] loss:0.145
[1,700] loss:0.136
[1,800] loss:0.120
[1,900] loss:0.124
```

Test Model

We can use `test_model(trained model, test loader)`.

```
In [12]: test_model(trained_model, test_loader)
```

```
correct1:  tensor(9647)
Test acc: 0.9647
```

ii. Second Test

```
In [11]: trained_model = train_model(net_work, train_loader, LR, EPOCH)
```

```
[1,100] loss:0.816
[1,200] loss:0.294
[1,300] loss:0.221
[1,400] loss:0.196
[1,500] loss:0.158
[1,600] loss:0.149
[1,700] loss:0.137
[1,800] loss:0.124
[1,900] loss:0.117
```

Test Model

We can use `test_model(trained model, test loader)`.

```
In [12]: test_model(trained_model, test_loader)
```

```
correct1:  tensor(9667)
Test acc: 0.9667
```

iii. Third Test

```
In [11]: trained_model = train_model(net_work, train_loader, LR, EPOCH)
```

```
[1,100] loss:0.781
[1,200] loss:0.300
[1,300] loss:0.212
[1,400] loss:0.188
[1,500] loss:0.174
[1,600] loss:0.155
[1,700] loss:0.134
[1,800] loss:0.118
[1,900] loss:0.120
```

Test Model

We can use `test_model(trained model, test loader)`.

```
In [12]: test_model(trained_model, test_loader)
```

```
correct1:  tensor(9655)
Test acc: 0.9655
```

b. CNN

i. First Test

```
In [11]: trained_model = train_model(net_work, train_loader, LR, EPOCH)
```

```
[1,100] loss:1.765
[1,200] loss:0.999
[1,300] loss:0.734
[1,400] loss:0.618
[1,500] loss:0.542
[1,600] loss:0.479
[1,700] loss:0.443
[1,800] loss:0.437
[1,900] loss:0.407
```

Test Model

We can use `test_model(trained model, test loader)`.

```
In [12]: test_model(trained_model, test_loader)
```

```
correct1:  tensor(9745)
Test acc: 0.9745
```

ii. Second Test

```
In [11]: trained_model = train_model(net_work, train_loader, LR, EPOCH)
```

```
[1,100] loss:1.734
[1,200] loss:0.936
[1,300] loss:0.696
[1,400] loss:0.558
[1,500] loss:0.506
[1,600] loss:0.477
[1,700] loss:0.446
[1,800] loss:0.409
[1,900] loss:0.398
```

Test Model

We can use `test_model(trained model, test loader)`.

```
In [12]: test_model(trained_model, test_loader)
```

```
correct1:  tensor(9802)
Test acc: 0.9802
```

iii. Third Test

```
In [11]: trained_model = train_model(net_work, train_loader, LR, EPOCH)
```

```
[1,100] loss:1.823
[1,200] loss:0.983
[1,300] loss:0.693
[1,400] loss:0.581
[1,500] loss:0.529
[1,600] loss:0.453
[1,700] loss:0.436
[1,800] loss:0.420
[1,900] loss:0.386
```

Test Model

We can use `test_model(trained model, test loader)`.

```
In [12]: test_model(trained_model, test_loader)
```

```
correct1:  tensor(9759)
Test acc: 0.9759
```