Developing an ML system to solve a classification problem – Part 2 Laboratory: Handwritten Digit Recognition

Jedrzej Fratczak (2207175)

Contents

Introduction

This report will overview the work done in the lab as well as answer key question about using neural networks for solving a digit recognition problem. The questions answered will focus on the advantages and disadvantages of using specific neural networks algorithms, provide a visual representation of the logic used in such algorithms and provide insight into different ways the algorithm can be improved for further functionality.

List of the pros/cons of Neural Network for solving digit recognition problem.

Advantages

The main advantages of using neural networks for solving digit recognition are high accuracy, scalability, flexibility and ability for further expansion.

Deep learning models when trained with large data set such as the mnist one allows for a highly accurate result when trying to recognise a handwritten digit. This is obviously an advantage as the higher accuracy the better the model results in practice.

Because of the nature of the algorithm, it can handle large datasets. This means that by increasing the number of inputs we can create more complex and deeper networks for further tasks such as handwriting recognition showcasing its scalability.

Due to the nature of the software and how it detects different inputs such as, raw pixel data and even to some extent noisy images, algorithms operating on the same principle can be used across other tasks such as object detection or speech recognition. This showcases the flexibility of the neural networks in different tasks.

Lastly considering all the advantages we can conclude that the neural networks algorithm optimised for solving a digit recognition problem can be expanded further to increase its functionalities and allow for solving of bigger more complex problems.

Disadvantages

The main disadvantages of using neural network algorithms to solve digit recognition problems are the need for high computational resources, large data sets required for training and long training times.

Training the neural network algorithms takes a vast amount of computational power, using normal software this would result in long training times and inefficiency. To overcome that problem everything was compiled using google colab which allows for usage of external GPUs to do all the calculations. Without the use of external GPUs and instead using normal hardware the training time would be too long to be able to successfully complete the task.

Neural networks require large data sets to be able to have optimum performance. Thankfully for this task this was overcome by using an external dataset with enough training data to create a complete solution. However, if such a dataset is not available the results may be subpar. If the dataset is too small the software may memorise the examples instead of learning from them, this would result in the model doing exceptionally well on known images but heavily struggle on anything new.

Overall using neural network is greatly beneficial if the right conditions are achieved. However, if the user does not have the right hardware and the right database to train the model on often time the results will be undesirable.

How is the recognition / classification problem solved with the Neural Network?

Code explanation

Firstly, the MNIST dataset which contains 70,000 handwritten images if fetched. This can be seen below in figure 1.

```
mnist = fetch_openml('mnist_784', data_home='datasets', as_frame=False, cache=False)
```

Figure 1 - MNIST dataset fetched

Following that the dataset is split into the training data and the test data. The data is then normalised to ensure that the neural network works faster and prevents any issues from arising. Afterwards the data is reshaped to match the CNN, the dimensions are 28x28. Lastly the example images are displayed using matplotlib to ensure the data has been imported correctly.

Figure 2 - Data graphed for integrity testing

After the data integrity is checked, each image is manipulated to further fit the training algorithm. All the manipulation can be seen below in figure 3.

```
mnist_dim = X.shape[1]
hidden_dim = int(mnist_dim/8)
output_dim = len(np.unique(mnist.target))
mnist_dim, hidden_dim, output_dim
```

Figure 3 - Data manipulation

After the data is transformed the training stage can begin. For the two different algorithms there are different training parameters, for the neural network algorithm a higher learning rate is used with more epoch whereas for the convolutional neural network a smaller learning rate is used with less epochs. All the parameter assignments can be seen below in figure 4 and 5.

```
net = NeuralNetClassifier(
    ClassifierModule,
    max_epochs=100,
    lr=0.1,
    device=device,
)
```

Figure 4 - nn hyperparameters

```
cnn = NeuralNetClassifier(
    Cnn,
    max_epochs=20,
    lr=0.001,
    optimizer=torch.optim.Adam,
    device=device,
)
```

Figure 5 - cnn hyperparameters

Finally, after the data is trained, we can use the testing set to check the accuracy of the different neural network algorithms. After running the accuracy score for both we end up with 97.3% for the classic neural network and 99% for the convolutional neural network. These accuracy score show us that using the MNIST dataset the CNN algorithm performs better compared to the classic NN algorithm.

Lastly to fully test the accuracy new handwritten numbers were given to the different algorithms to see how the perform. These algorithms performed well but there is still space for further improvement by adjusting the hyperparameters and increasing the accuracy of the algorithms as well as making sure the new images given are in the 28x28 format.

Flowchart

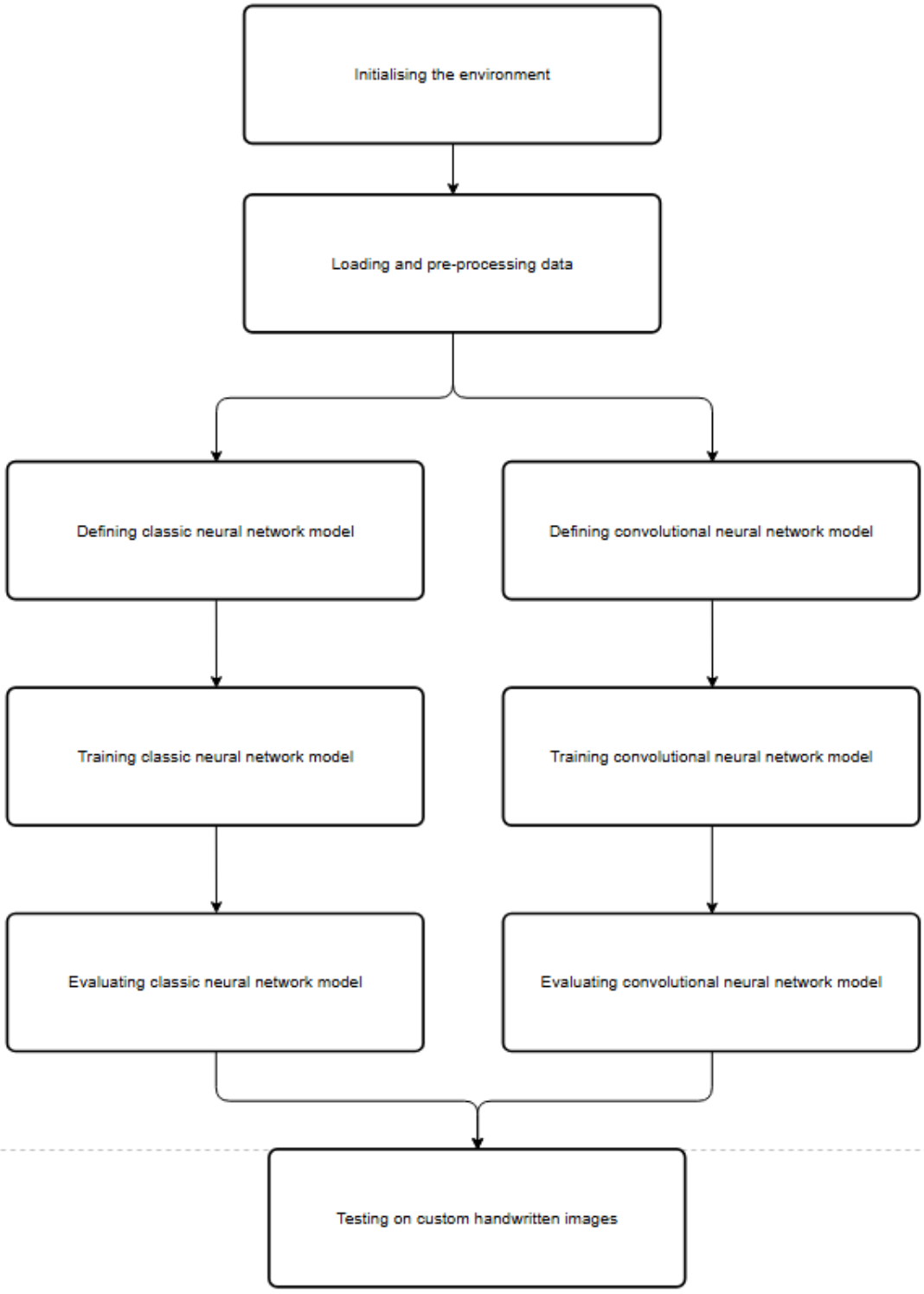

Figure 6 - flowchart

For a more complex problem (Letter recognition), how can you expand the current solution to solve problems?

To adapt the current number recognition program and amend it for letter recognition key factors need to be kept consistent and some other adapted for a successful implementation. Firstly, it needs to be ensured that the dataset resolution is adapted to fit the new dataset, this is because letters might require a larger resolution inputs depending on how complex the training dataset will be.

The output layer will also need to be adjusted, this is because there is more possible letters than numbers. For numbers only 10 classifications were needed, for 0 to 9. Meanwhile to properly implement letter detection 52 classification are needed, this is as there are 26 letters in the alphabet and the system should work on both uppercase and lowercase letters.

Furthermore, with a new dataset different normalisation and optimisation techniques would need to be used to ensure the most optimal solution. The data could be adjusted on many different factors to be consistent such as rotation, brightness and noise to ensure the training set is as close to real handwritten documents.

Overall, a multitude of adjustments would need to be made to ensure the smooth transformation between a number recognition system and a letter recognitions system.

Conclusion

This report explored the application of classic neural network and convolutional neural network using google colab for handwritten number recognition. The solution used the MNIST dataset to train different models and then used original handwritten numbers to test the accuracy of the models. Overall, the models achieved 97% accuracy and 99% accuracy respectively. Finally, the model could be further expanded for implementation of a handwritten letter recognition system by adjusting different features of the code.