Introduction

This project will provide a demonstration of the capabilities of machine learning techniques on object recognition in image data with the MNIST dataset. The MNIST dataset is set of 70,000 (60,000 for training and 10,000 for testing) 28 x 28 pixels grayscale images of handwritten digits, which are pre-labeled. The project will require us to train a classifier to recognise the digits that are in the testing portion of the dataset using the training portion to train the classifier. It is seen as the 'hello world' problem of the object recognition for machine learning as the MNIST dataset allows developers to evaluate models easily as there is very little pre-processing required of the data.

Specifically, we will be designing a Convolutional Neural Network classifier to solve our digit recognition problem as it is the most powerful supervised machine learning technique for solving image recognition problems. A Convolutional Neural Network (CNN) is a biologically-inspired model which consists of a multi-layered perceptron with a lot of layers that is trained using back-propagation. The layers in the model differ from normal neural network layers as they are Convolutional Layers, not the usual fully-connected layers. The Keras library allows us to easily implement the Convolutional neural network for required for the project using Python.

Aim

The aim of this project is to develop a program in Python, using the Keras neural network library, to implement a classifier for the MNIST handwritten digits database. The classifier will use a Convolutional neural network to achieve a classification accuracy greater than 99% on the MNIST test set.

Description of the model

Convolutional neural network architecture:

1. Convolutional layer with 32 feature maps of size 5x5
2. Convolutional layer with 32 feature maps of size 5x5
3. Pooling layer taking the max over 2*2 patches
4. Dropout layer with a probability of 25%
5. Convolutional layer with 32 feature maps of size 5x5
6. Convolutional layer with 32 feature maps of size 5x5
7. Pooling layer taking the max over 2*2 patches
8. Dropout layer with a probability of 25%
9. Flatten layer
10. Fully connected layer with 256 neurons and rectifier activation
11. Dropout layer with a probability of 50%
12. Fully connected layer with 10 neurons and softmax activation
13. Output layer

The first two layers of our model are convolution layers with 32 nodes in each layer and a 5x5 kernel, with the first layer also taking the input image. These layers will be applied with a Rectified Linear Activation function which has been proved to work well in neural networks. The pooling layer was then added to the CNN to reduces the spatial dimensions by 2 x 2 of the input image to reduce the computational load of the program. Next, a dropout layer is added which forces the classifier to learn from new data by randomly turning off every 0.25 neurons which are important for preventing overfitting. The four layers are then repeated in the network to improve the accuracy of the model.

A flatten layer is added to convert the 2d data from the images into a 1D feature vector. Each node in the Fully connected layer receives this flattened output as input. Another dropout layer is added, this time with a rate of 50%. The final layer consists of connections for our 10 classes and the softmax activation which is normal for this type of CNN.

To then compile the model, we will need an optimizer, loss and metric parameters. The optimizer of choice was ‘adam’ which adjusts the learning rate throughout training. The

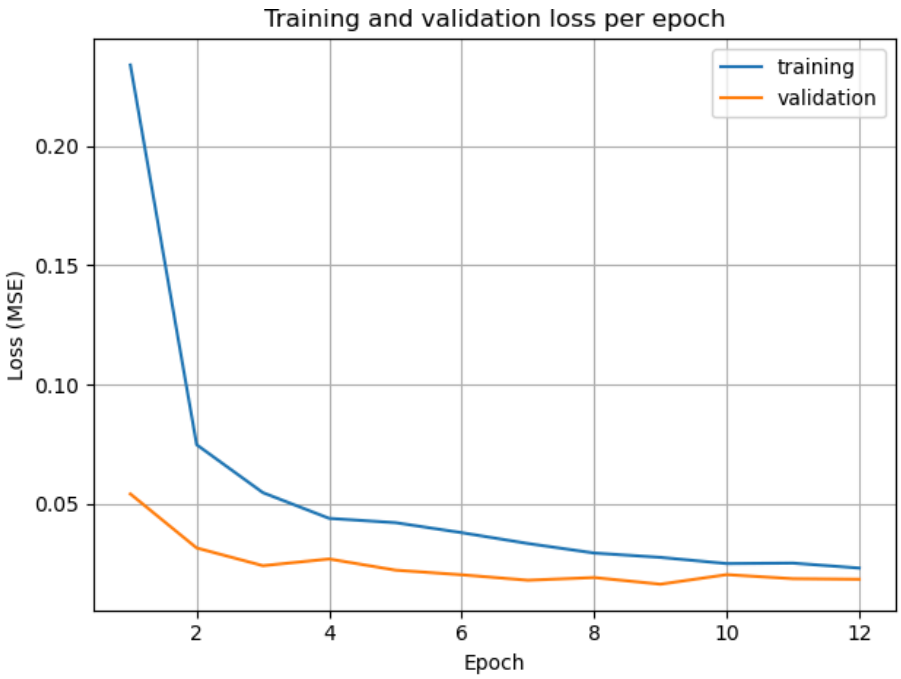
‘categorical_crossentropy’ seemed like a popular choice for the loss function and ‘accuracy’ metric will allow us to see the accuracy score on the validation set as we train the model.

Building the model and evaluating its performance

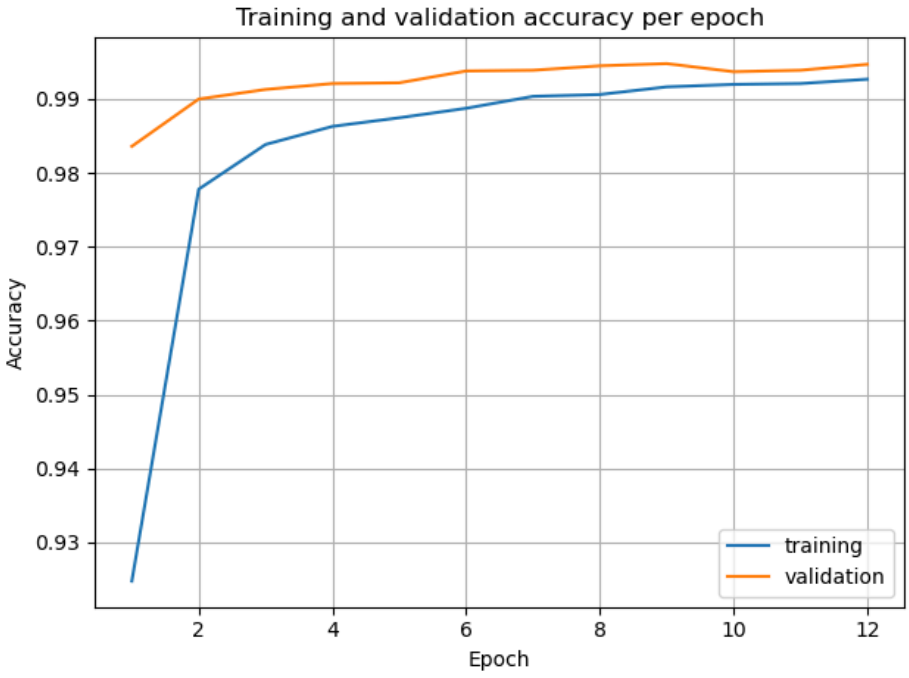
Building the model proved to be relatively straightforward with the Keras library, as there were functions provided for building the convolutional neural network.

```
Epoch 1/12
938/938 - 14s - 14ms/step - accuracy: 0.9247 - loss: 0.2340 - val_accuracy: 0.9836 - val_loss: 0.0541
Epoch 2/12
938/938 - 12s - 13ms/step - accuracy: 0.9778 - loss: 0.0747 - val_accuracy: 0.9900 - val_loss: 0.0314
Epoch 3/12
938/938 - 12s - 13ms/step - accuracy: 0.9839 - loss: 0.0546 - val_accuracy: 0.9913 - val_loss: 0.0240
Epoch 4/12
938/938 - 12s - 13ms/step - accuracy: 0.9863 - loss: 0.0438 - val_accuracy: 0.9921 - val_loss: 0.0268
Epoch 5/12
938/938 - 12s - 13ms/step - accuracy: 0.9875 - loss: 0.0420 - val_accuracy: 0.9922 - val_loss: 0.0221
Epoch 6/12
938/938 - 12s - 13ms/step - accuracy: 0.9887 - loss: 0.0378 - val_accuracy: 0.9938 - val_loss: 0.0202
Epoch 7/12
938/938 - 12s - 13ms/step - accuracy: 0.9904 - loss: 0.0333 - val_accuracy: 0.9939 - val_loss: 0.0179
Epoch 8/12
938/938 - 12s - 13ms/step - accuracy: 0.9906 - loss: 0.0293 - val_accuracy: 0.9945 - val_loss: 0.0190
Epoch 9/12
938/938 - 12s - 13ms/step - accuracy: 0.9916 - loss: 0.0275 - val_accuracy: 0.9948 - val_loss: 0.0162
Epoch 10/12
938/938 - 12s - 13ms/step - accuracy: 0.9920 - loss: 0.0249 - val_accuracy: 0.9937 - val_loss: 0.0202
Epoch 11/12
938/938 - 12s - 13ms/step - accuracy: 0.9921 - loss: 0.0251 - val_accuracy: 0.9939 - val_loss: 0.0185
Epoch 12/12
938/938 - 12s - 13ms/step - accuracy: 0.9927 - loss: 0.0230 - val_accuracy: 0.9947 - val_loss: 0.0183
```

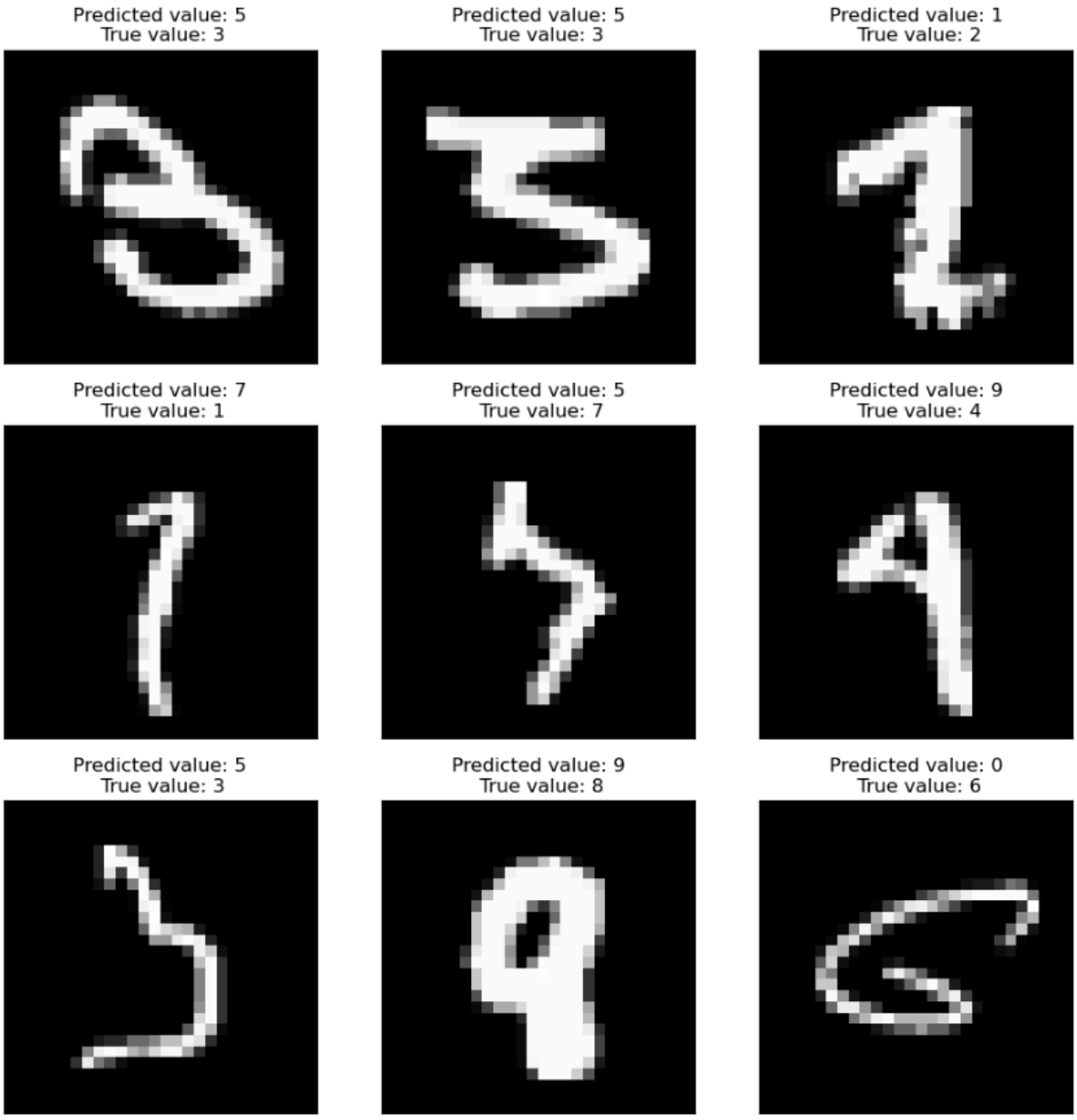
This shows the output from the Convolution neural network on the MNIST dataset. The CNN misclassifies 62 images out of 10,000, resulting in 99.38% accuracy after 12 training cycles. This model performs well on both the training and testing datasets which is a good indication that it would also perform well on unseen data.



We can see that the accuracy on the training and testing datasets increase rapidly after the first two epochs. This shows that the network can classify the digits quickly.



We can see that the loss on the training and testing dataset decreases rapidly after the first two epochs which is very desirable for this system.



From the above figure, we can see 9 examples that the classifier incorrectly predicted. A lot of these examples would even be difficult for a human to recognize so it is understandable that our classifier incorrectly classified them. In conclusion, the Convolution neural network is an especially useful method of image recognition and it performed excellently on our digit recognition problem. It achieved the overall goal of the project with an accuracy of approximately 99.3%.