CSC 761 Final Project

Dr. Christer Karlsson

CSC 761 - Advanced AI

April 29, 2022

Analysis of Implementations of Artificial Intelligence

This project aims at implementing two different Classifiers or Machine Learning Algorithms to identify handwritten numbers provided to it, correctly. Both the models will be evaluated on their performance as well as the training efficiency.

The data used for training and testing the model was obtained from the **MNIST dataset**. The testing and training datasets contain the following properties/features.

- Training Dataset has **60,000** instances
- Test Dataset has **10,000** instances
- There are 10 unique Evaluation Label Values (0 to 9)
- Each instance is a grayscale image of:
  - Size = **28x28** pixels
  - Pixel Value Range = **0 to 255**
- The images are centered in w.r.t the centre of mass of all the pixels

Model Evaluation Parameters

The models will be trained in batches of **15 epochs** with a learning rate of **0.001**.

Models will be evaluated **primarily on accuracy**. In the case of both models having fairly similar accuracies (~2.5% of each other), batch training time followed by model complexity (gotta give Occam some props) will be used as tiebreakers.

Artificial Neural Network Model

The first implemented model we are going to talk about is the Artificial Neural Network (ANN) . This model is made up of multiple layers of single units called **neurons or nodes or perceptrons** (that's why its sometimes called a Multi-Layer Perceptron model) (Figure 1).

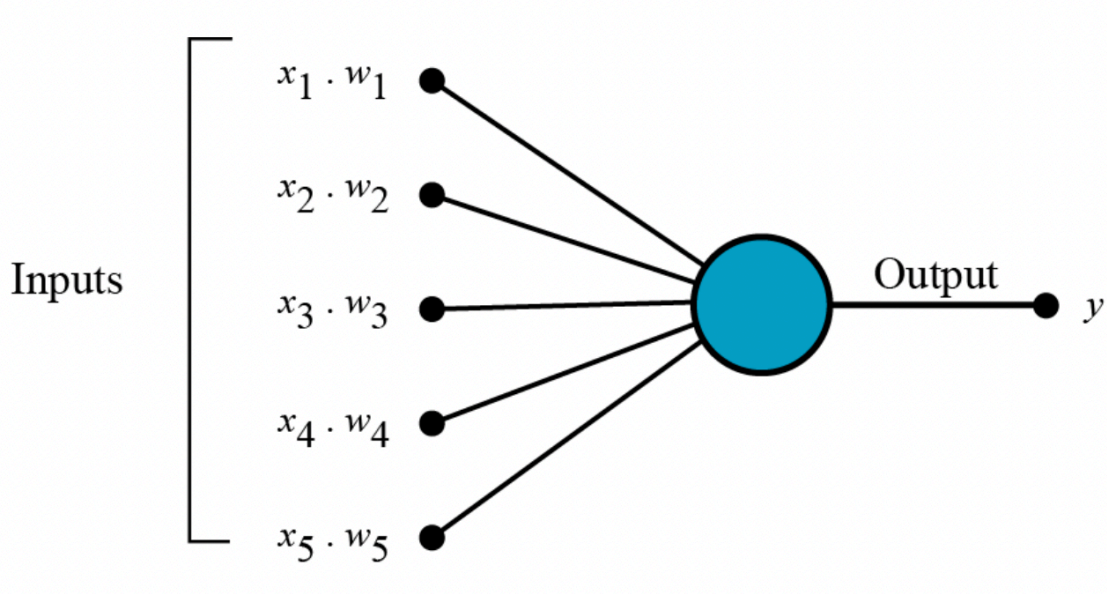

Figure 1. Mathematical Neuron

A neuron can be thought of as a function:

$$y = f(W * X)$$

where X is a vector of inputs $[x_1, x_2, \ldots . x_n]$ and W is a vector of corresponding weights $[w_1, w_2, \ldots . w_n]$ and f is a function that constraints the output between 0 and 1 (called an **activation function**). The values in the vector W are figured out through the process called **training** and **backpropogation.**

These basic units are then arranged in layers as shown below (Figure 2).

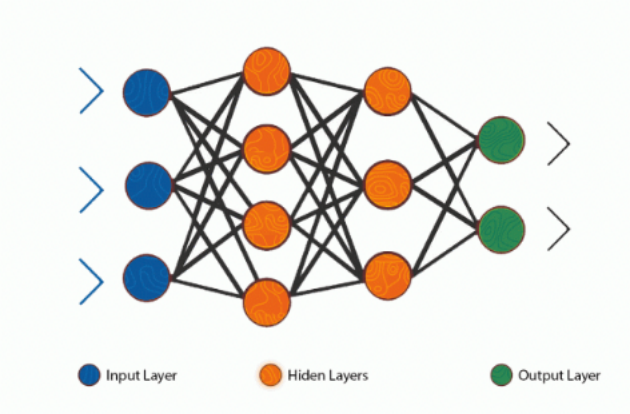

Figure 2. General Model of an Artificial Neural Network

What's evident from the figure above is that there are 2 layers exposed (not fully connected on both sides). These exposed layers are called the Input and Output layers while all the other layers in between are called the Hidden Layers (the specifics of this layer are only known to the implementer).

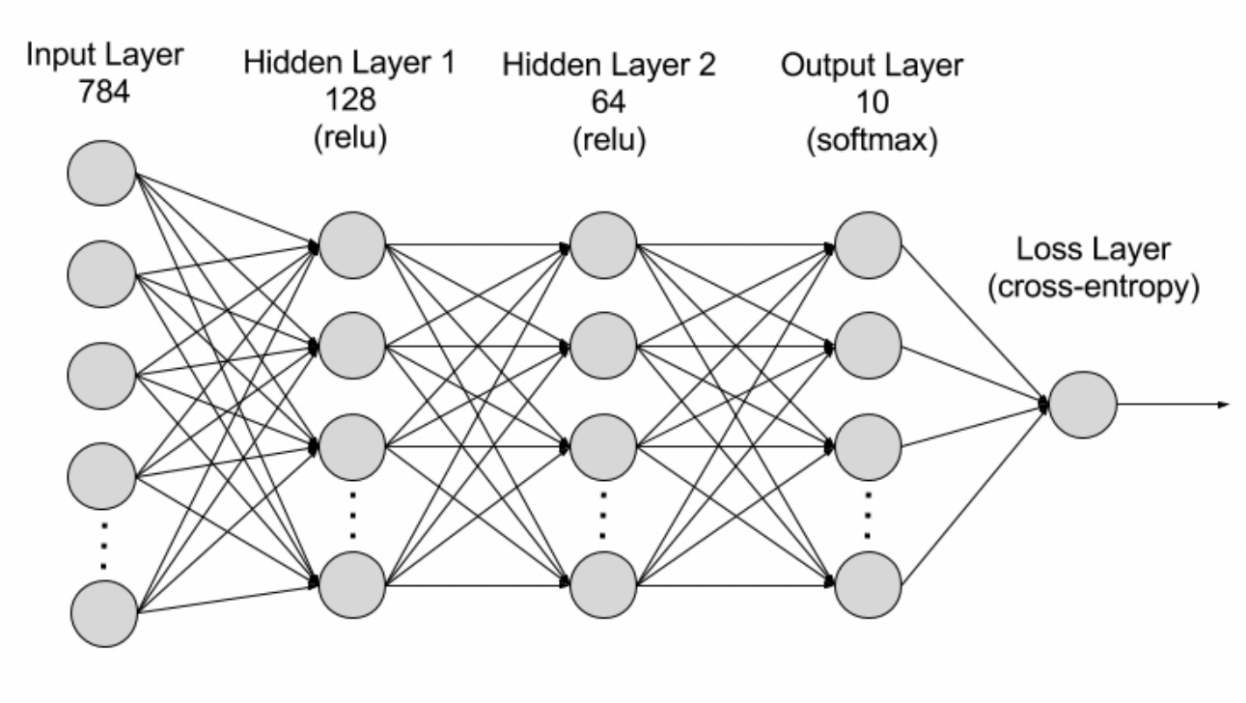

Figure 3. Implemented ANN

Now that we have a general idea of what a Multi Layer Perceptron Artificial Neural Network looks like, we can move on to a more complex, tuned model implemented in this project (Figure 3).

Every image instance contains 28x28 = 784 pixels, hence our input layer has 784 neurons and directly takes in the a pixel value [0-255] and passes it on to first Hidden Layer.

The number of nodes in the First Hidden Layer and Second Layer were arbitrarily chosen. The key feature here is the the activation function, which is a ReLU (Rectified Linear Unit) function. The input-output graph of a ReLU unit is shown below (Figure 4).

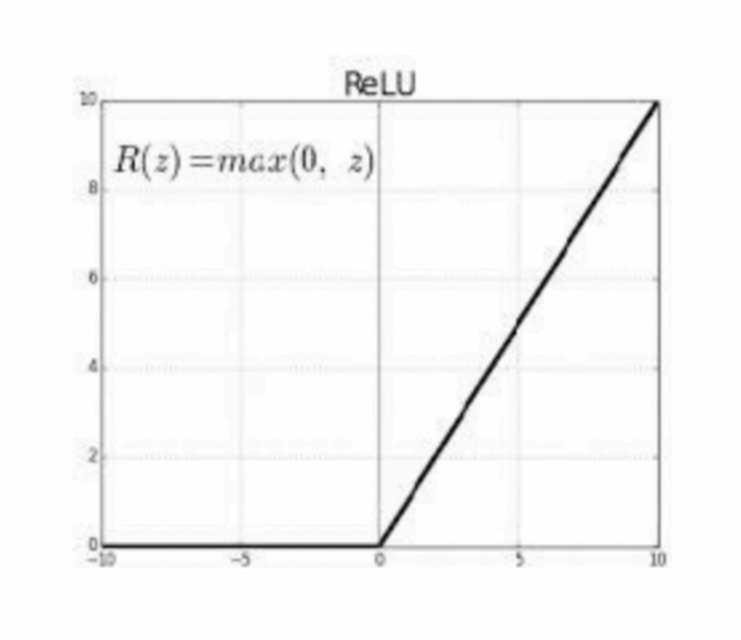

Fig 4. Input-Output of a ReLU

Each input image corresponds to a number between 0 and 9; that's the reason why the output layer has 10 nodes, as the ANN returns its prediction as a probability distribution between equally distributed among the 10 outputs. This is achieved using another activation function called a sigmoid function that spreads the probability of the prediction among the 10 outputs in such a way that the total probability is 1. The output neuron with the highest probability is considered as the prediction of the ANN and is evaluated against the actual answer. If the predicted value is correct, it gets a carrot otherwise it gets the stick. This is the process of training the network. The network initially starts out with all weights = 1, but then changes that when it gets penalized. This method is called backpropogation.

Now that we have a general idea of the implementation, we can get down to the specifics of the parameters and hyperparameters:

- 4 layers, 2 hidden
- 784 input nodes -> 128 nodes (HL 1) -> 64 nodes (HL2) -> 10 output nodes
- Hidden Layer 1 Activation Function = ReLU
- Hidden Layer 2 Activation Function = ReLU
- Output Activation Function = Softmax(10)
- Learner - Adam Function
- Learning Rate = 0.001
- Number of Epochs per Batch = 15

You can find the code to it here. The code is commented to depict which section implements which part of the neural network. The results that the model achieved along will be discussed and compared with the CNN model in a later Section.

One of the major shortcomings of this model is the loss of spatial orientation to some degree. Apart from, that if the image specifications were to change slightly, this would drastically change the number of nodes and hidden layers. It also struggles to accommodate high dimensional data.

Convolutional Neural Network Model

Convolutional Neural Networks implement 3 main features that improve Machine Learning: Sparse Interactions, Parameter Sharing and Equivariant Representations. The key defining feature of Convolutional Neural Networks is the implementation of Tensors and Matrix Mathematics, mainly the Convolution Operation.

On a broad scale, a Convolution Neural Network is made up of 3 major components, the input layer, convolutional layer and output layer. The convolutional layer is where most of the magic happens and is subdivided into further stages as shown in Figure 5. A CNN model can have multiple Convolution Layers (these are similar to the Hidden Layers in an MLP ANN)

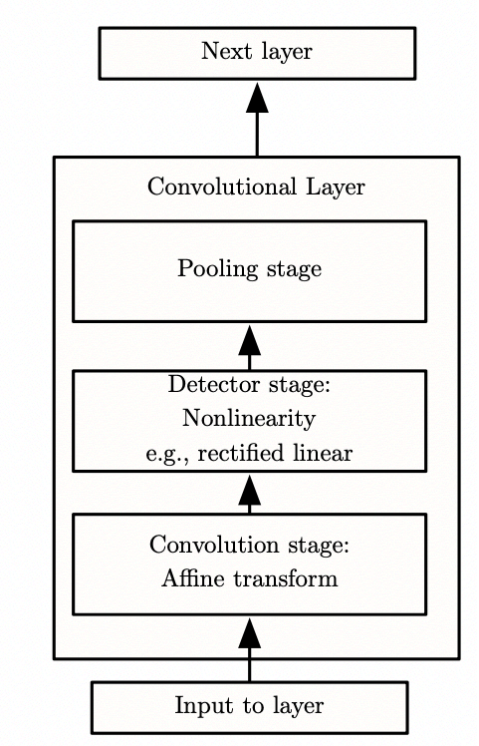

Figure 5. Layers of a CNN

The magic that happens as in the convolution layer is as follows:

- The data matrix that comes from the input is convoluted with a filter matrix or “kernel” of size (x,x). The filter then moves along the elements specified by the stride parameter.
- The convoluted results are then passed through an activation function to normalize it
- These results are then pooled together in the pooling layer and spread out among the output nodes to be sent to the next layer.

Now that we a rough idea of what a CNN is we can move on to the specifics of the implemented model.

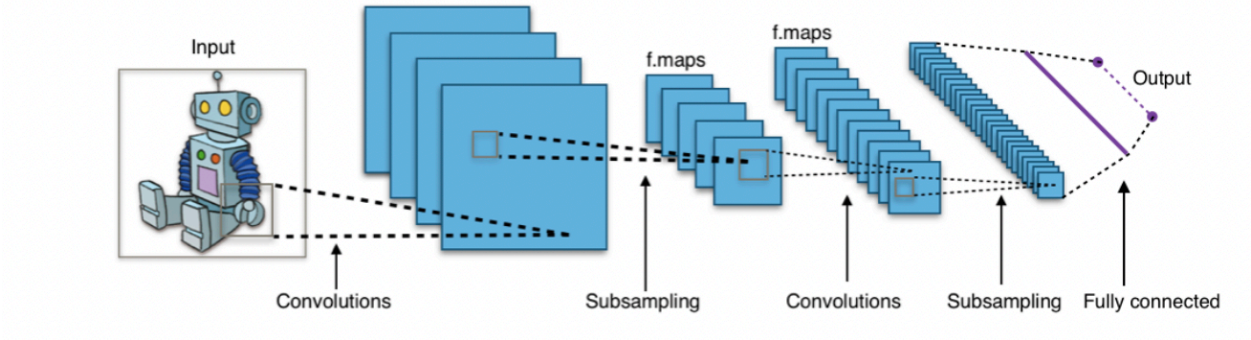

Figure 6. Implemented CNN Model

This model is made up of:

- 4 layers, 2 Convolutional Layers
- Input Layer that takes the whole picture as a tensor
- Convolutional Layer 1 with one input channel, 16 output channels, kernel size = (5,5) and stride = (1,1). Detector Function = ReLU with Pooling kernel size = (2,2)

- Convolutional Layer 2 with 16 input channel, 32 output channels, kernel size = (5,5) and stride = (1,1). Detector Function = ReLU with Pooling kernel size = (2,2)
- Fully Connected Layer that connects the 32 * 7 * 7 outputs from the previous layer to 10 outputs. (Equivalent to implementing a softmax)

The code that implements this model can be found here. The code is commented to depict which section implements which part of the neural network. The results that the model achieved along will be discussed below.

One of the major shortcomings of this model is that it is computationally expensive. Without the right hardware (CUDA cores), this implementation takes a significantly longer time to run.

Evaluation Results

Each model was run in batches of 15 epochs, 10 times; resulting in 150 total epochs. The model was tested after every 15 epochs or at the end of each epoch and the accuracy and training time were noted. Figure 7 shows the cross entropy loss after each epoch.

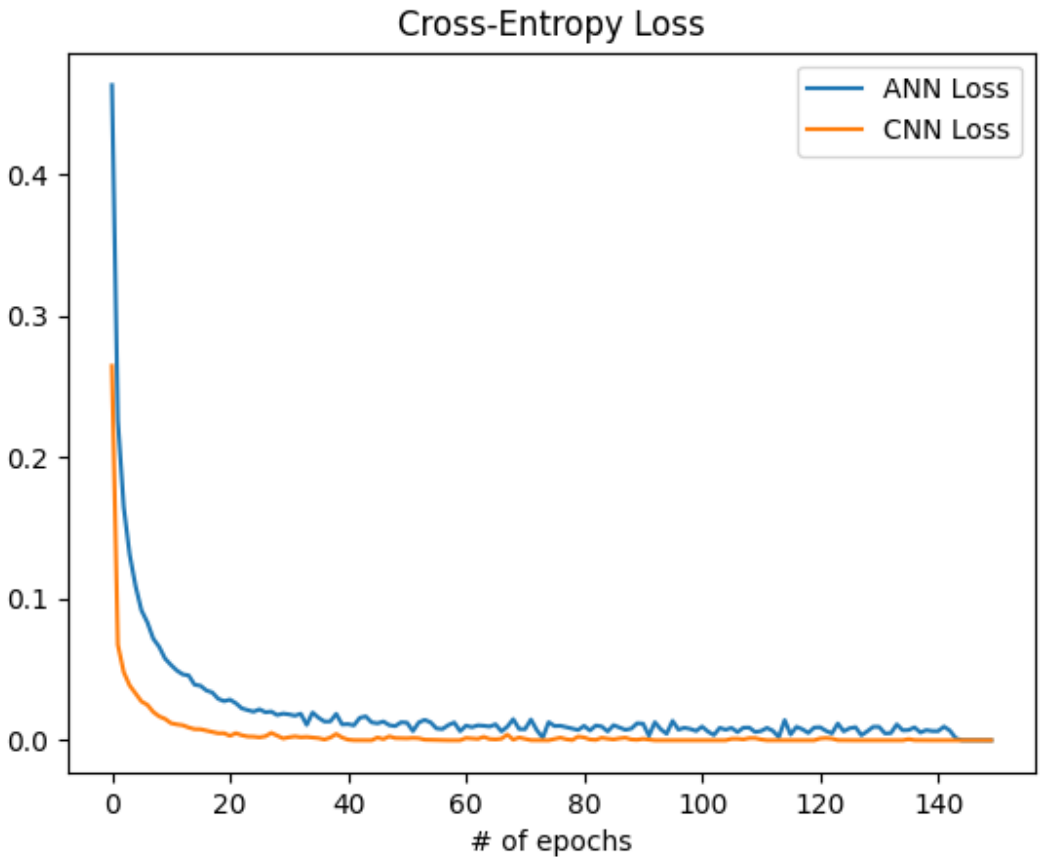

Figure 7. Cross Entropy Loss Comparison

From the above figure, we can see that the CNN model trains in a lesser number of epochs (this does not mean faster!) and reaches a stable-ish state by 20 epochs. After around 140 epochs, both models seem to display similar cross entropy loss. A ques-

tion you may have is can we straight up compare Cross-Entropy Losses like that? Yes, yes we can. Both models implemented the same simple cross entropy information gain (based on the KL Divergence formula) and in reality is the negative log of the expected value of the correct predictions to the wrong predictions (or the Maximum Likelihood value of P(Event)).

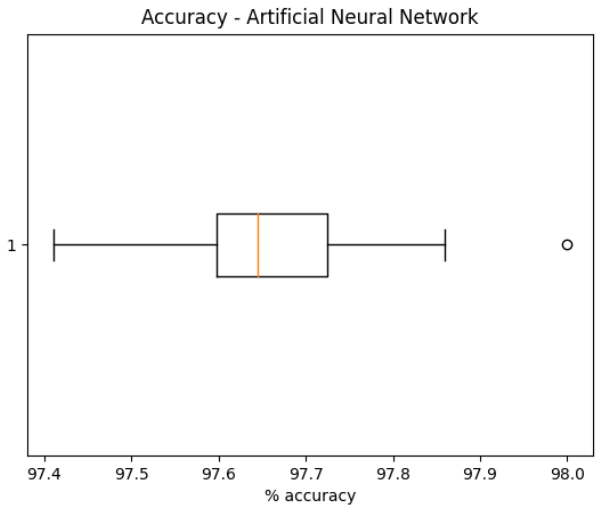

Figure 8. ANN Accuracy

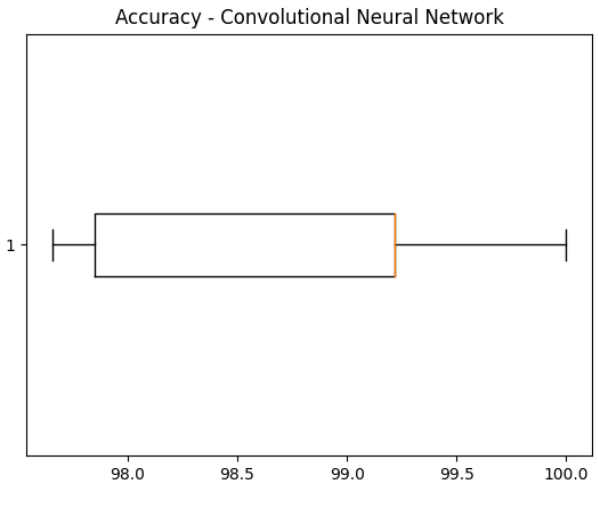

Figure 9. CNN Accuracy

From the above (Figure 8 and 9) we can see that the CNN did significantly better than the ANN. By our metric, the **CNN** is a better model with it having a median accuracy of about 99.25% and a maximum of 100%. The highest the ANN got in terms of accuracy was 98% with the median being 97.63%.

Even though we have established which model is better, let's take a look at another parameter - the training time each model took to train in 15 epochs.

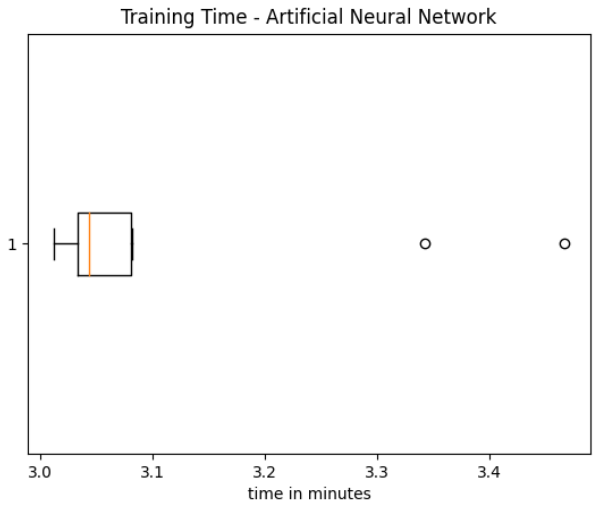

Figure 10. Training Time ANN

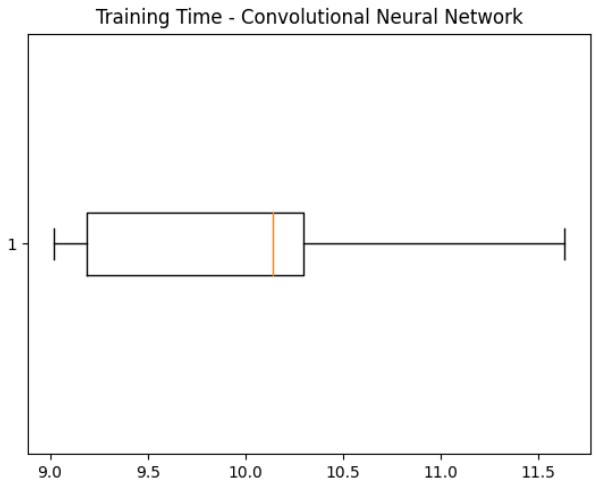

Figure 11. Training Time CNN

From the plots in Figure 10 and 11, we can see that the CNN took significantly more time than the ANN (more than 7 minutes on average). Even though the CNN reached a local minimum cross entropy loss in lesser epochs, the ANN reached the lower cross entropy loss way faster (1 hour and 8 minutes faster than the former). This longer training time though, can be accounted to the CNN being trained on a 8 core CPU rather than a GPU which would have greatly reduced the training time.

The link to the repository containing the files can be found here: csc761-final-project

Supplemental Questions and Answers:

1) What is your problem statement, your proposed solution?

A. The aim of this project is to model a deep learning network that is able to recognize and classify the number images provided in the MNIST dataset

2) How did you train, test and validate your model, i.e. what did you do to find the optimal solution for your generalized error?

A. I batched the model trainings into 15 epochs per batch. I trained them till I achieved a stable accuracy above 97%.

3) What model did you implement, what parameters and hyper parameters did you use. What are the expected shortcomings of this model?

A. For the Multilayer Perceptron ANN Model, the following parameters were used:

- 4 layers, 2 hidden
- 784 input nodes -> 128 nodes (HL 1) -> 64 nodes (HL2) -> 10 output nodes
- Hidden Layer 1 Activation Function = ReLU
- Hidden Layer 2 Activation Function = ReLU
- Output Activation Function = Softmax(10)
- Learner - Adam Function
- Learning Rate = 0.001
- Number of Epochs per Batch = 15

One of the major shortcomings of this model is the loss of spatial orientation to some degree. Apart from, that if the image specifications were to change slightly,

this would drastically change the number of nodes and hidden layers. It also struggles to accommodate high dimensional models.

For the Convoluted Neural Network Model, the following parameters were used:

- 4 layers, 2 Convolutional Layers
- Input Layer that takes the whole picture as a tensor
- Convolutional Layer 1 with one input channel, 16 output channels, kernel size = (5,5) and stride = (1,1). Detector Function = ReLU with Pooling kernel size = (2,2)
- Convolutional Layer 2 with 16 input channel, 32 output channels, kernel size = (5,5) and stride = (1,1). Detector Function = ReLU with Pooling kernel size = (2,2)
- Fully Connected Layer that connects the 32 * 7 * 7 outputs from the previous layer to 10 outputs. (Equivalent to implementing a softmax)

One of the major shortcomings of this model is that it is computationally expensive. Without the right hardware (CUDA cores), this implementation takes a significantly longer time to run. Works best with images as its able to extract key features from the image similar to perception by the human eye, but not as well with others like tabular data.

4) Detailed description what the code and each function does. Yes, this means that if you use predefined functions, you need to figure out how they work.

A. Please refer to the code for the answer to this question. There's also supplementary information on pages 5, 6 and 7 which loosely answers this.

5) Report of the results, including but not limited to things like graphs that describes the learning curves (learning loss and classification accuracy), if you batch your training you should be able to create box and whisker plots for the accuracy.

A. Please refer to the section on page 10 for this.

6) From the validation and/or prediction what digits did you model struggle with?

A. There was no clear indicator/pattern the ANN or CNN failed on in terms of predictions. I printed out the results as well as wrongly predicted values and for most of the cases the digits were just squiggly lines or dashes that I even could not recognize.

7) How accurate is your model?

A. The CNN model achieved an average accuracy of 99.25% while the MLP ANN model achieved an accuracy of 97.63%

Sources:

2. Bose, Amitrajit. “A Must Read Intro to Neural Networks Using Pytorch-Handwritten Digit Recognition.” Handwritten Digit Recognition Using PyTorch — Intro To Neural Networks, Towards Data Science, 9 Mar. 2022, https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627.

3. “Convolutional Neural Network Pytorch: CNN Using Pytorch.” Analytics Vidhya, 10 May 2020, https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/.

4. Pai, Aravind. “CNN vs. RNN vs. ANN – Analyzing 3 Types of Neural Networks in Deep Learning.”, February 17, 2020, https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

4. Rosebrock, Adrian “LeNet – Convolutional Neural Network in Python.”, August 1, 2016 https://pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/