Comparative Analysis of Neural Networks trained on the MNIST Dataset: Performance and Generalization

Name: Aaditya Sahoo

Course: Mechatronics (IMC)

Semester: 2

Elective: Machine Learning

Summer Semester 2024

Professor: Prof. Boris Bittner

Comparative Analysis of Neural Networks Trained on the MNIST Dataset: Performance and Generalization

Aaditya Sahoo

Abstract

A neural network is a type of artificial intelligence that processes information through structures called neurons (just like the human brain). Each neuron sits in a particular layer, and each neuron in any layer is connected to every other neuron in the layers in front of it and behind it. Using neural networks, complicated problems can be solved with limited human interaction, as they are very good at understanding complex non-linear relationships between input and output. Three neural networks have been used to analyse the performance of each in doing a not so basic task **- The identification of a hand-drawn digit**. In this research paper a comparison has been made between 3 different types of neural networks.

How a neural network works

A neural network is composed of an input layer, one or more hidden layers, and an output layer. Each neuron is connected to every other neuron in the layer in front and behind it, and each connection has its own associated weight and bias. This bias is also called the threshold value of the node, as whenever the output of any node is less than the threshold, then no output is produced. Only if the value of the output is more than the threshold value, is the output passed forward to the next neurons.

For this paper 3 different neural networks have been tested-

- **(a) Single layer neural network -** This network has only one input layer and one output layer, which are connected with each other
- **(b) Double layer neural network -** In this network, there is a second layer after the first input layer. Each layer can have an arbitrary number of neurons, however in my case, this second layer contains 800 neurons. Since they are not directly part of the input or the output layer, this is called the hidden layer, and the neurons are called "hidden units"
- **(c) Convolutional Neural Network, with augmented elastic distortion -** In this neural network, 8 **layers** have been taken **- 2 convolutional layers, 2 pooling layers, 3 dense layers and 1 regular output layer.** This makes the CNN very powerful and much more accurate as compared to the other neural networks that were tested.

1. Introduction

1.1 Background on handwritten digit recognition:

Handwritten digit recognition is a very important application in the field of machine learning and pattern recognition. This task involves classifying images of handwritten digits into one of ten categories (0-9). It serves as a benchmark problem for evaluating the performance of various machine learning algorithms because of its challenging nature and real-world applicability. Effective digit recognition systems have significant implications for numerous applications, including automated postal mail sorting, bank check processing, and digitizing handwritten notes.

1.2 The MNIST Dataset

The Modified National Institute of Standards and Technology (MNIST) dataset is a widely used benchmark dataset for handwritten digit recognition. It consists of 60,000 training images and 10,000 testing images, each of 28x28 pixels in grayscale, representing digits from 0 to 9. The MNIST dataset is particularly valuable for the machine learning community because it is large, well-studied, and contains a diverse set of samples, making it an ideal starting point for developing and testing new algorithms and models.

1.3 Overview of Neural Network Architectures and Their Evolution:

Neural networks have undergone significant evolution over the past few decades. Early neural networks were simple, consisting of only a few layers and neurons, limiting their ability to capture complex patterns in data. The introduction of multi-layer perceptrons (MLPs) marked a significant advancement, but it was the development of convolutional neural networks (CNNs) that revolutionized the field of image recognition. CNNs are specifically designed to process grid-like data, such as images, through their use of convolutional layers that capture spatial hierarchies in the data. Over time, more sophisticated architectures like deep CNNs, residual networks (ResNets), and various regularization techniques have been developed, leading to remarkable improvements in performance and generalization

1.4 Objective of the study:

The objective of this study is to compare the performance of three different neural network models on the MNIST dataset and evaluate their effectiveness in identifying hand-drawn digits. By analyzing the accuracy, computational complexity, and training efficiency of each model, this study aims to provide insights into the strengths and limitations of various neural network architectures for the task of handwritten digit recognition.

2. Methodology

2.1 Dataset:

I have split the MNIST dataset into 2 parts - the training part consists of 60000 samples, and the test part consists of 10000 samples. Each sample is essentially a number scribbled onto a canvas that is a grid of 28 x 28 pixels. Then the sample gets converted to a column vector of 784 elements (28x28) as it contains the row-wise cell value for each pixel, which in turn has its own value of black and white, from 0 to 255
For the CNN, the input is not a 784-element column vector, but rather the direct 28x28 grid. Before the CNN was trained, the samples were augmented by a process called elastic distortion. This is called preprocessing. A convolutional NN is considered to be the best for image recognition and analysis. The difference between a normal NN and CNN is that instead of normal weight multiplication like in other NNs, the hidden layers of CNNs (called convolution layers) get an operation called Convolution done on them. The convolution is done by something called a filter, which can be thought of as a matrix. Depending on what filter is used and the number of filters used, this can be used to greatly increase the accuracy of the prediction. Elastic distortion is essentially shifting the x and y position of each pixel using some random value between -1 and 1. Then the fields (i.e. function which give the x and y displacement of every pixel) are convolved with the Gaussian of the standard deviation. This creates many more images which are different from the original, but still have the same value. This allows us to have a larger range of training data, so that our neural network is more accurate.

2.2 Drawing board:

Draw and Recognize Digit
Models
Clear Board
Go

I have written a code that creates a drawing board on the screen. The user has to choose out of one of 3 models available to recognize the digit. Once the model has been selected, the user has to draw their digit on the board. Once they are done, they can press “GO” and the program will try to interpret the digit drawn by the user, using the model they have selected.

Fig 1 (left): drawing board

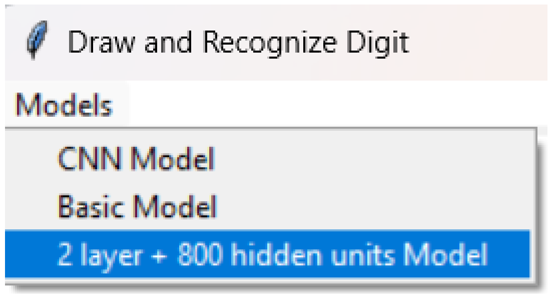
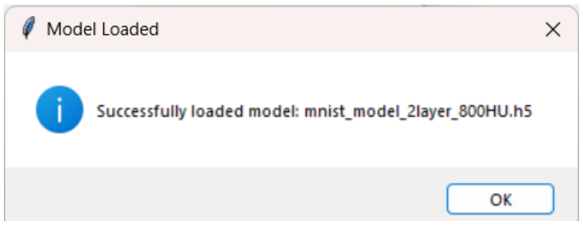


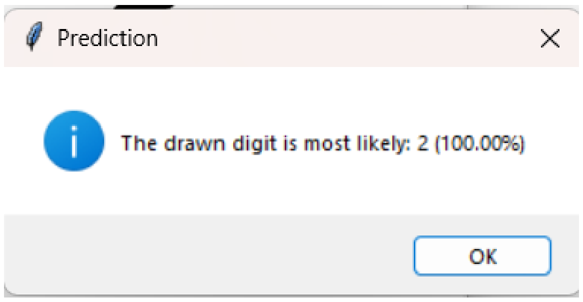
Fig. 2 (above): choosing the model


Model Loaded

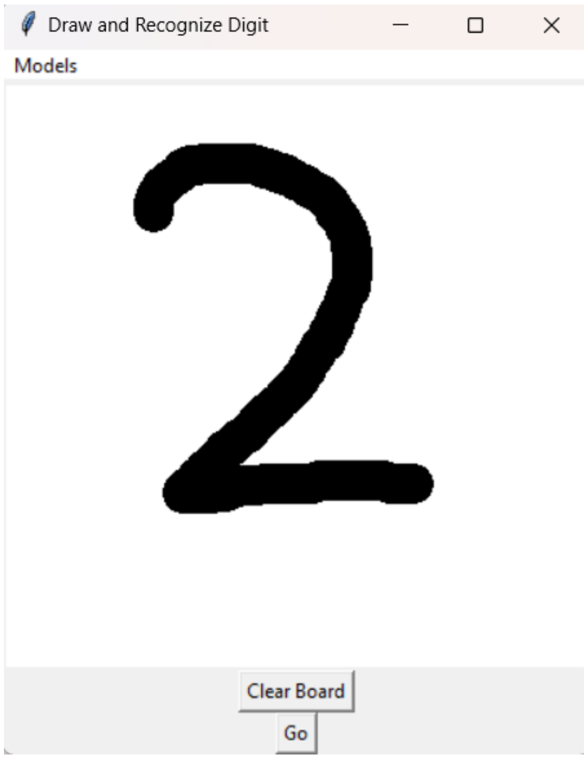
Successfully loaded model: mnist_model_2layer_800HU.h5

Fig 3. (left): Confirmation on choosing model
Fig.4 (bottom): Drawing the image on the board

Fig 5 (bottom): the recognized digit!


Prediction

The drawn digit is most likely: 2 (100.00%)



3. Models

I have used 3 models, and each of them were created using the help of the library ‘TensorFlow’. Once these models were trained, they were saved onto the disk. Later on in the main program, these models were loaded again to predict the digit.

3.1 Single layer neural network

This program begins by importing the libraries which are TensorFlow (to create and load the neural network) and NumPy to do array manipulations. The MNIST dataset, which consists of handwritten digit images, is loaded using keras.datasets.mnist.load_data(). This function returns training and testing datasets. After that, the training and testing images are reshaped from 28x28 pixels to 784-element vectors using the reshape method, preparing them for input into the neural network. The pixel values of the images are then normalized by dividing by 255.0, scaling the values to a range of 0 to 1. This helps in faster convergence during training.
A simple neural network model is defined using keras.Sequential(). It has one Dense layer with 10 units and a sigmoid activation function, this makes it suitable for multi-class classification. After this,

the model is compiled with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the evaluation metric.

The **Adam optimizer** is an adaptive learning rate optimization algorithm designed for training deep learning models. It combines the advantages of two other popular optimizers: AdaGrad, which works well with sparse gradients, and RMSProp, which works well in online and non-stationary settings. Adam maintains per-parameter learning rates that are adjusted based on the first and second moments of the gradients, making it efficient and suitable for large datasets and high-dimensional parameter spaces.

Sparse categorical cross-entropy is a loss function used for multi-class classification problems where the target variable is an integer. The loss function measures the performance of a model, whose output probability is between 0 and 1.

After compiling, the model is trained on the training dataset for 15 epochs using the fit method. During training, the model learns to recognize digit patterns from the input data. Then the model is evaluated on the test dataset using the evaluate method to determine its accuracy on unseen data.

The trained model is then saved to a file named model.h5 using the save method. This allows the model to be loaded and used later without retraining.

Purpose: The overall purpose of this code is to train a basic neural network to classify handwritten digits, demonstrating the process of data preparation, model training, evaluation, and saving in a concise script.

```python
import tensorflow as tf #imports
from tensorflow import keras
import numpy as np

(xtrain, ytrain), (xtest, ytest) = keras.datasets.mnist.load_data()
flat_xtrain = xtrain.reshape(len(xtrain), 784)
flat_xtest = xtest.reshape(len(xtest), 784)
flat_xtrain = flat_xtrain/255.0 #normalising dataset
flat_xtest = flat_xtest/255

model = keras.Sequential([   #creating the NN
    keras.layers.Dense(10, input_shape = (784, ), activation='sigmoid')
])

model.compile(  #compiling and optimising
    optimizer='adam',
    loss = 'sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.fit(flat_xtrain, ytrain, epochs= 15)
model.evaluate(flat_xtest, ytest)
model.save('model.h5')  #saving the model for later use
```

3.2 Double layer neural network (with 800 hidden Units):

```
model = keras.Sequential([
    keras.layers.Dense(800, input_shape=(784,), activation='relu'),
    keras.layers.Dense(800, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

The rest of the code is almost the same. In this code 2 hidden layers with 800 hidden units each are created. Then the output is returned to the final layer containing 10 neurons, each corresponding to the percentage of certainty per digit.

Activation function: An activation function in a neural network defines the output of a neuron given an input or set of inputs, determining whether a neuron should be activated or not. It introduces non-linearity into the network, enabling it to learn and perform complex tasks. Activation functions allow the network to capture intricate patterns in the data by transforming the summed weighted input from the node into the activation output.

ReLU is an activation function that returns 0 if the input is less than 0 and x if it is greater than 0. The softmax function is mostly used in the output layers. It is used to convert logits to probabilities. It is defined by the formula

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

3.3 Convolutional Neural network

Convolutional Neural Networks consist of several layers, including convolutional layers, pooling layers, and fully connected layers, which help to automatically and adaptively learn spatial hierarchies of features from input images.

The convolutional layer is arguably the most important part of a CNN. It applies a set of filters to the input image, creating **feature maps** that detect various patterns such as edges, textures, and more complex structures. Each filter convolves around the image, producing a two-dimensional activation map that represents the presence of specific features at different spatial locations. This process significantly reduces the number of parameters and computation required, making CNNs highly efficient for image processing.

Pooling layers, typically following convolutional layers, perform down-sampling operations that reduce the spatial dimensions of the feature maps. The most common type of pooling is max-pooling, which takes the maximum value from a set of neighboring pixels. This not only reduces the computational load but also helps to make the detected features invariant to small translations in the input image, enhancing the model’s robustness.

In the final stages of a CNN, fully connected layers are used to make predictions. These layers take the high-level filtered and pooled features and use them to output class probabilities or other predictions. By the time the data reaches these layers, the CNN has effectively learned to recognize and classify images based on the features extracted in the earlier layers.

In our CNN model, elastic image augmentation is applied to enhance the robustness and generalization capabilities of the network. Elastic distortions mimic real-world variations by slightly warping the images in a non-linear way, simulating the kind of transformations that might occur in natural settings. This technique helps the network to learn more invariant features and prevents overfitting by providing a diverse set of training examples.

The input layer accepts 28x28 pixel grayscale images. The first convolutional layer with 32 filters of size 3x3 detects low-level features like edges, followed by a max-pooling layer to reduce spatial dimensions and computational load. The second convolutional layer with 64 filters further captures detailed patterns, and another max-pooling layer consolidates the learned features. The flatten layer converts 2D feature maps into a 1D vector for the fully connected layers. The first fully connected layer with 128 neurons learns higher-level representations, and the output layer with 10 neurons uses softmax activation to provide class probabilities. This layer structure enables the network to progressively extract and integrate complex features for accurate digit classification.

```
model = Sequential([  #main CNN
        Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(800, activation='relu'),
        Dense(800, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

def augment_images(images): #implements the elastic distortion
    seq = iaa.Sequential([
        iaa.ElasticTransformation(alpha=34.0, sigma=4.0)
    ])
    return seq(images=images)

model.save('cnn_model_with_distortion.h5')
```

In the end the mode is saved, so that it can be called later without having to retrain it.

4. Training

The general training process of a neural network involves feeding input data into the network, performing **forward propagation** to compute the output, and then using a loss function to measure the discrepancy between the predicted and actual values. **Backpropagation** is then employed to adjust the weights in the network by minimizing the loss function through an optimization algorithm like stochastic gradient descent (SGD) or Adam. This process is iterated over multiple **epochs** until the model converges to an optimal or near-optimal state, ensuring it can generalize well on unseen data.

Before the training process, some terms called **hyperparameters** need to be defined. Hyperparameters are external configurations set before the training process of a neural network, and they play a crucial role in determining the model's performance and efficiency. Unlike model parameters, which are learned during training, hyperparameters are predefined and include elements such as learning rate, batch size, number of epochs, optimizer type, and the architecture of the network (e.g., number of layers and units per layer).

4.1 Single Layer NN

Hyperparameters:

- **Optimizer:** Adam
- **Loss Function:** Sparse Categorical Cross-Entropy
- **Metrics:** Accuracy
- **Batch Size:** Default (typically 32 in Keras)
- **Epochs:** 15

Training Process: The model was trained for 15 epochs on the flattened MNIST images, with each image represented as a 784-dimensional vector. The Adam optimizer was used to adjust the learning rate dynamically during training, and the sparse categorical cross-entropy loss function was minimized using backpropagation. The accuracy metric was tracked to monitor performance.

4.2 Neural Network with 2 hidden layers

Hyperparameters:

- **Optimizer:** Adam
- **Loss Function:** Sparse Categorical Cross-Entropy
- **Metrics:** Accuracy
- **Batch Size:** Default (typically 32 in Keras)
- **Epochs:** 15
- **Activation Function (Hidden Layers):** ReLU
- **Activation Function (Output Layer):** Softmax

Training Process: Similar to the single-layer network, this model was trained for 15 epochs. The Adam optimizer and sparse categorical cross-entropy loss function were used, with the ReLU activation function employed in the hidden layers to improve learning efficiency and the softmax activation in the output layer for classification.

4.3 Convolutional Neural Network

Hyperparameters:

- **Optimizer:** Adam
- **Loss Function:** Sparse Categorical Cross-Entropy
- **Metrics:** Accuracy
- **Batch Size:** Default (typically 32 in Keras)
- **Epochs:** 15
- **Activation Function (Conv & Dense Layers):** ReLU
- **Activation Function (Output Layer):** Softmax

Training Process: The training involved data augmentation using elastic distortions to enhance the robustness of the model. The images were augmented before being fed into the CNN. The Adam optimizer facilitated dynamic learning rate adjustments, and the sparse categorical cross-entropy loss was minimized. The training process was conducted for 15 epochs, similar to the previous models, with accuracy tracked as the primary performance metric.

4.4 Comparison of training times

Below in table 1 is a comparison between the various training times taken for the different neural networks.

Networks	Time for first Epoch vs Time for last Epoch		Total time
Single Layer NN	2.660 s	1.656s	19.25 seconds
Double NN + 800 HU	19s	19s	4min 52.18 seconds
CNN	38s	42s	10min 32.01 seconds

Table 1: comparison of times taken for the first and last epoch, and the total time taken

4.5 Comparison of loss and accuracy metrics

Networks	Loss of first layer vs Loss of last layer		Accuracy of first layer vs Accuracy of last layer	
Single Layer NN	0.46	0.24	87.78%	93.26%
Double NN + 800 HU	0.98	0.07	67.10%	97.92%
CNN	0.547	0.03	82.17%	99.07%

Table 2: Comparison of loss and accuracy on the first and last layer of each neural network

Networks	Loss of test data	Accuracy of test data
Single Layer NN	0.266	92.51%
Double NN + 800 HU	1.564	78.25%
CNN	0.600	91.78%

Table 3: The comparison of loss and accuracy for the test data (after training the model)

4.6 Comparison of actual accuracy

For this comparison each digit was tested on each neural network 10 times, the ratio for number of right guesses to the number of total attempts (i.e., 100) was taken as the metric for accuracy.

Networks	Actual accuracy
Single Layer NN	49%
Double NN + 800 HU	87%
CNN	97%

Table 4: Showing the actual experimental accuracy for each neural network.

For Single layer NN, it had difficulty identifying 0, 6, 8 and 1, 7 and 4, 9
For Double NN, it still has difficulty to understand difference between 8, 6, 0
For the CNN there is a much better improvement, as there is no more confusion between digits, and the errors occur due to random human writing errors.

4.7 Comparison with other established benchmarks

The MNIST dataset has been tested many times by different algorithms. Some common benchmarks have been provided to see how they test with recognising handwritten digits.

a. **Support vector machines (SVMs):** this is a supervised machine learning model that tries to find the line/plane of best fit to evaluate and classify input data. It is heavily dependent on the kernel that is used for the data. On the linear MNIST dataset, a linear kernel gives an accuracy of 91.82%
b. **k-Nearest Neighbours (kNNs):** it is one of the simplest algorithms for supervised machine learning. It uses all of its training data to classify each new point/instance. While testing at k = 1 and k = 5, it gives the highest accuracy of 95.4% and 95.1% respectfully
c. **Random Forest Classifier:** it is an ensemble learning method that uses multiple decision trees which are then merged into one on the basis of the majority result. Random forests correct for the trees' habit of overfitting to the training data. The accuracy comes to about 96%

5. Analysis

The tables provided give a comprehensive comparison of the performance, efficiency, and accuracy of three different neural network models: a Single Layer Neural Network (NN), a Double Layer Neural Network with 800 hidden units (HU), and an 8-layer Convolutional Neural Network (CNN). Below is a detailed analysis of these results.

5.1 Training time:

- The single layer takes the least time to train
- The double layer takes longer time, which reflects its increased complexity
- The CNN has the longest training time, consistent with its more complex architecture and the additional computational cost of convolutional operations.

5.2 Accuracy and Loss during training:

- The Single Layer NN shows a reduction in loss and an increase in accuracy, indicating effective learning, although the final accuracy is lower compared to more complex models.
- The Double neural network with 800 HU shows a substantial decrease in loss and a dramatic increase in accuracy, reflecting its capacity to learn and generalize better due to its deeper architecture.
- The CNN demonstrates the most significant improvement in accuracy and reduction in loss, achieving the highest final accuracy, highlighting the strength of convolutional layers in analysing image data.

5.3 Test Data performance

- The Single Layer NN performs reasonably well on the test data, with a loss of 0.266 and an accuracy of 92.51%.
- The Double NN with 800 HU performs worse on test data, with a higher loss and lower accuracy, suggesting overfitting despite high training accuracy.
- The CNN performs well on the test data, with a lower loss compared to the Double NN but slightly lower accuracy than the Single Layer NN. However, considering the highest final training accuracy, the CNN likely has the best generalization.

5.4 Actual experiment data

- The Single Layer NN has a much lower actual accuracy compared to its test accuracy, indicating that it may not generalize well to real-world handwritten digits.
- The Double NN with 800 HU shows significantly better actual accuracy, aligning more closely with its high training accuracy.
- The CNN has the highest actual accuracy at 97%, reinforcing its superiority in handling real-world variations in handwritten digits due to its deep and complex architecture.

It is also possible that because all the actual data was taken on a very small subset (100 tests) it might not be very accurate, and also scaling up the testing data would probably change the entire result.

5.5 CONCLUSION

The analysis reveals the following key points:

1. **Training Efficiency:** The Single Layer NN is the fastest to train but has the lowest overall performance.

2. **Model Complexity vs. Performance:** The Double NN with 800 HU shows good learning capacity but suffers from overfitting, indicated by its poor test performance despite high training accuracy.

3. **CNN Superiority**: The CNN, while the most computationally expensive, achieves the best performance in terms of both training accuracy and real-world generalization, making it the most effective model for handwritten digit recognition among the three.

LINKS and REFERENCES:

Links:

- MNIST handwritten digit database: http://yann.lecun.com/exdb/mnist/
- Amazon Web Services: What is a neural network? https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a,that%20resembles%20the%20human%20brain
- IBM: Convolutional Neural Networks https://www.ibm.com/topics/convolutional-neural-networks
- Towards Data Science: Elastic deformation on images https://towardsdatascience.com/elastic-deformation-on-images-b00c21327372
- Data for benchmark comparisons
  https://dmkothari.github.io/Machine-Learning-Projects/SVM_with_MNIST.html
  https://www.kaggle.com/code/walidmourou/k-nearest-neighbors-on-mnist-dataset
  https://www.kaggle.com/code/ashwani07/mnist-classification-using-random-forest

Papers:

- Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003, August). Best practices for convolutional neural networks applied to visual document analysis. In Icdar (Vol. 3, No. 2003).

- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).