Final Project Report

On

"Digit Recognition using CNN with Optimized Model's Performance using Batch Normalization and Dropout Layer"

Group Name: Neurons

Group Members:

First name	Last Name	Student number
Anil	Dhakal	(C0898886)
Sanat Kumar	Upreti	(C0898884)
Syed Roman	Uddin	(C0906298)
Ujjwal	Khatri	(C0899849)

Submitted to:

Pouria Modaresi

Artificial Intelligence and Machine Learning Department

Lambton College, Toronto, Canada

Submission date: ***18th August 2023***

ABSTRACT

Using neural networks, this project delves into the challenge of identifying digits accurately. The project mainly focuses on performing digit recognition with CNN and model optimization using batch normalization and dropout techniques. The handwritten digit dataset is preprocessed through different techniques like normalization, one-hot encoding, etc. to make it suitable to feed into neural networks. Then, the different models such as the multi-layer neural network model, convolution neural network (CNN) model, and CNN model optimized with batch normalized layer and dropout layer are developed and trained. Different visualization plots such as confusion matrix, actual versus predicted graph, training and validation accuracy and loss are depicted for model evaluation. From overall results and analysis, the introduction of batch normalization and dropout technique in the CNN model shows improved performance such as higher accuracy, convergence speed, and stability.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION TO THE PROJECT

1.1 Overview

The digit recognition problem is a crucial component of computer vision and pattern recognition that involves distinguishing handwritten digits and assigning them their corresponding numerical values. This issue is of great significance with practical applications in optical character recognition, postal automation, and digitized document processing. Thus, in this study, we examine the feasibility of solving the digit recognition predicament using neural networks.

In this study, we investigate the viability of using neural networks to solve the digit recognition problem. To accomplish precise and efficient digit recognition, we created a comprehensive system that integrates cutting-edge tools, methodologies, and design concepts. To improve its recognition skills, the system makes use of modern neural network topologies, data preparation methods, and training methodologies.

1.2 Background

To understand the significance of our effort, it is necessary to first understand the history of Artificial Intelligence (AI) and Machine Learning (ML). The evolution of AI initially conceptualized in the mid-20th century, had remarkable progress over the decades. In the area of machine learning, where algorithms were developed to let computers learn from data and gradually improve their performance, this advancement was particularly noticeable. Image recognition, a subdomain of AI and ML, has undergone a transformational journey. From its early stages of basic pattern matching, it evolved into a complex process involving the extraction of intricate features from images. This transformation paved the way for applications such as facial recognition, object detection, and, importantly, digit classification.

This project stands on the shoulders of these earlier developments because our project is firmly founded on digit categorization. This research into the digit recognition issue makes use of this development to build a modern system that not only handles the problems presented by handwritten digits but also utilizes the power of neural networks to produce precise and effective results. The technological specifics of our system, its techniques, tests, and results will be covered in more detail in the next chapters of this paper.

1.3 Problem Statement

In this project, a neural network-based system is constructed to address the problem of digit recognition. The main goal is to guarantee handwriting digit recognition with high accuracy, which is essential for applications like optical character recognition and document processing. To balance accuracy and efficiency, concentration is made on attaining rapid model convergence and training speed, optimizing neural network designs, and choosing the right hyperparameters. Strategies like regularization and dropout layers are adopted to reduce overfitting. The digit classification accuracy is also optimized by fine-tuning Convolutional Neural Network (CNN) designs. The system's generalization capabilities are evaluated using methods like cross-validation, showcasing its capacity for precisely identifying novel and unseen digits, to demonstrate its practical applicability.

1.4 Objectives

The prime motive behind the execution of this project was to develop a high-performance CNN model for achieving accurate and reliable digit recognition along with the implementation of the Batch Normalization and Dropout Technique to enhance the model's performance and generalization capabilities. Along with that, the following were the sub-motives of this project:

- To achieve a substantial enhancement in digit classification accuracy compared to that of the traditional neural network models.
- To evaluate the impact of Batch Normalization and Dropout layers in CNN model optimization.
- To apply the optimized CNN model to real-world scenarios using images sourced from Google Images and self-generated handwritten digit images.

1.5 Scopes of the Project

This project has numerous real-world applications in a variety of industries. One such area is postal automation, where the efficient sorting of mail items based on handwritten addresses can be streamlined using our digit recognition system. This project can also help automate handwritten digit-based data entry in the financial services industry, lowering transaction errors and increasing data accuracy. Additionally, this solution for digit recognition in medical forms can help the healthcare industry by streamlining the management of patient records. Inventory management in retail can be made easier by accurately classifying handwritten numbers,

especially when recording product IDs and quantities. Finally, this system can help the education industry by automating the grading of handwritten tests. These practical applications highlight the extensive range and usefulness of our digit recognition project.

CHAPTER 2: TERMINOLOGIES

3.1 Artificial Neural Network (ANN) and Deep Neural Networks

ANNs are computational models inspired by the structure and function of the human brain. ANNs comprise interconnected nodes, or "neurons," that process and transmit data. An artificial neural network (ANN) with more than two layers between the input and output layers that help them learn complex patterns and features from data is called a deep neural network (DNN).

3.2 Convolution Neural Network (CNN)

A convolutional neural network is a feed-forward neural network typically used to analyze visual images by processing data in a grid-like topology. ConvNet is another name for it. A convolutional neural network quickly recognizes and categorizes objects.

3.3 Pixel and Color Channels

The smallest units in an image are called pixels, and each one represents a different color or shade. Red, green, and blue (RGB) are frequently used in color images as the three distinct color representational components known as color channels. The entire color spectrum that we see in an image is produced by the combination of these channels.

3.4 Grayscale and Color Images

Grayscale images contain only shades of gray, lacking color information. They are often used in scenarios where color is unnecessary, or computational simplicity is desired. The use of color channels in color images, on the other hand, enables them to represent a wider variety of visual data and makes them appropriate for tasks requiring in-depth color analysis.

3.5 Batch Normalization and Dropout Technique

Batch normalization is a method for accelerating and stabilizing neural network training. A batch of data's intermediate layer outputs are normalized, which speeds convergence by minimizing internal covariate shifts. Dropout, on the other hand, prevents overfitting by randomly deactivating a portion of neurons while the model is being trained, improving the model's capacity for generalization.

3.6 Numpy and Pandas

Numpy and Pandas are powerful Python libraries for data manipulation and analysis. While Pandas excels at handling structured data and providing data structures like DataFrames for effective data organization and manipulation, Numpy supports multi-dimensional arrays and mathematical operations.

3.7 TensorFlow

TensorFlow is an open-source deep learning framework developed by Google. It provides an intuitive interface for creating, training and deploying neural network models. TensorFlow is a popular choice among researchers and practitioners due to its flexibility and scalability.

3.8 OpenCV

OpenCV (Open-Source Computer Vision Library) is a flexible open-source library primarily used for computer vision tasks. It offers a wide range of features and resources for image and video analysis, processing, and manipulation.

3.9 Matplotlib

Matplotlib is a popular Python plotting library for creating static, interactive, and animated visualizations. It provides an easy-to-use interface for creating various types of plots and charts to aid in data exploration and presentation.

3.10 Jupyter Notebook

Jupyter Notebook is an interactive, web-based environment that makes it easier to create and share documents with live code, equations, visualizations, and explanatory text. It is a preferred platform for data analysis and machine learning prototyping because it encourages easy collaboration and experimentation.

CHAPTER 3: METHODOLOGY

The project is started with data extraction and some EDA is performed to identify preprocessing needs and have a thorough understanding of the data. Then, the project delves into data preprocessing such as normalization, reshaping, and converting labels into one-hot encoding suitable for neural network training. Then, three models of neural networks are developed and trained. The first model is a multi-layer fully connected dense neural network, the second is a convolutional neural network (CNN) and the third one is CNN with batch normalization and dropout regularization. After that, the project further proceeds for model performance visualizations such as accuracy, predictions of the labels, etc.

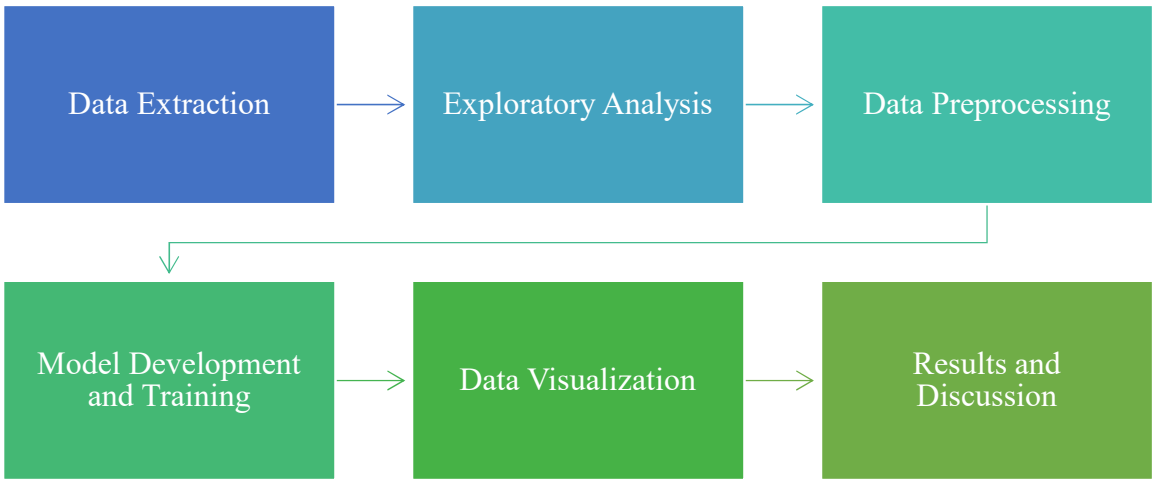


Figure 1 Project Methodology

3.1 Study Design

For data collection, this project utilizes a dataset consisting of handwritten digit images, where each image is a 28x28 grayscale representation. The dataset is preprocessed by normalizing pixel values and converting labels into one-hot encoded vectors. The project explores three different neural network architectures for digit recognition. The models are trained using the Adam optimizer and categorical cross-entropy loss. The 10% validation split is employed for monitoring

the training accuracy and loss. The model performance is evaluated using accuracy and visualizations of confusion matrices.

3.2 Data Extraction and Exploratory Analysis

The main study variables for this project are:

- Input: Grayscale images of size 28x28 pixels
- Output: Predicted digit labels (0-9)

From data extraction, the training and testing datasets are found to have 42K and 28K observations with 785 and 784 features respectively. The training dataset consists of one more feature than the testing set: "label" as the target variable. For exploratory analysis, the label distribution plot is generated that shows the class label "1" has a higher count of observations while label "5" has the lowest. However, there is not much difference in counts of all class labels, which means that the dataset has balanced data. The train dataset is grouped by label column and a pixel representation of the number 0 is formed. Also, the class labels from 0 to 9 are plotted as an image using the matplotlib. Furthermore, correlation analysis between pixel values of labels 0-9 is performed, aiding in understanding pixel relationships between numbers.

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pi›
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

5 rows × 785 columns

Figure 2 Training dataset

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	p
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

5 rows × 784 columns

Figure 3 Testing dataset

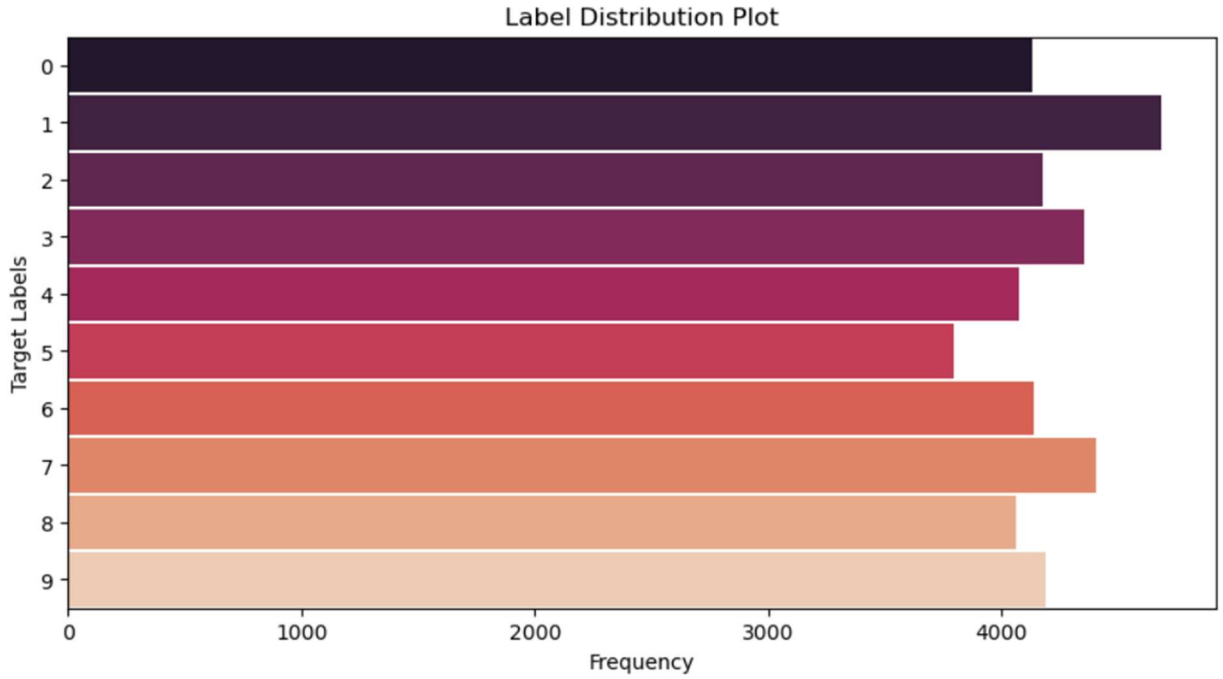


Figure 4 Label distribution plot

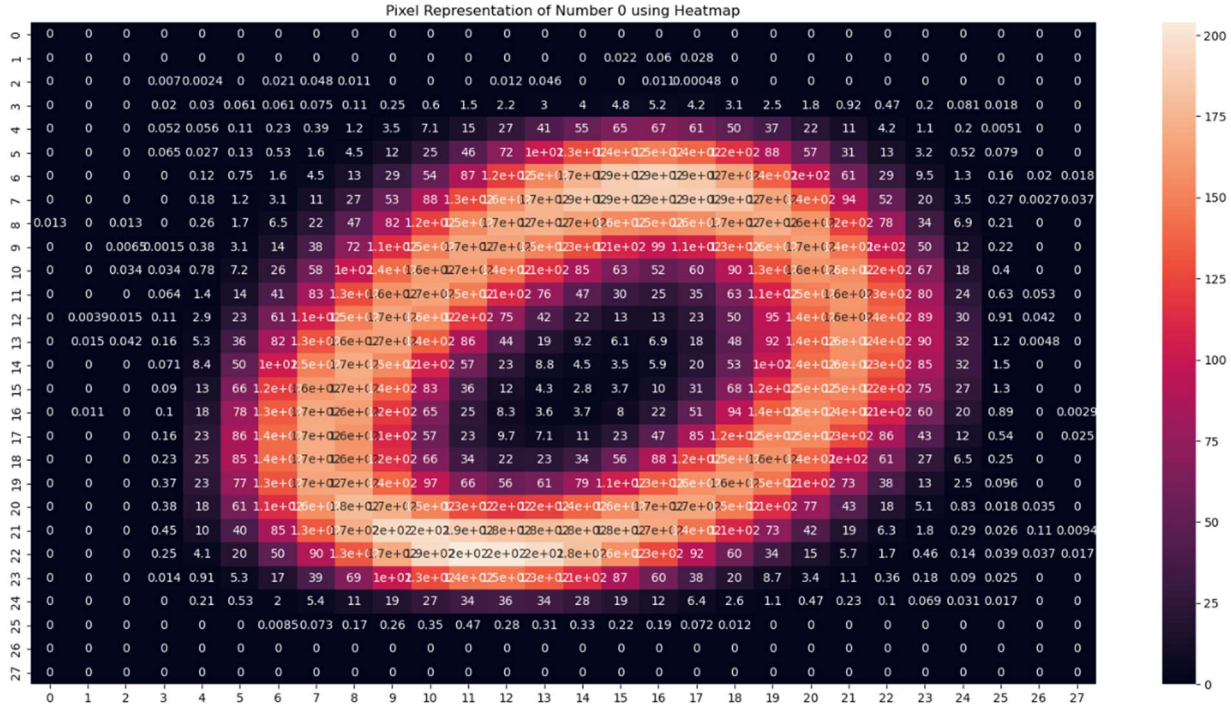


Figure 5 Pixel representation of number 0 using the heatmap

Figure 6 Labeled digit image

3.3 Data Preprocessing

3.3.1 Validation and Cleansing

In the first step of data preprocessing, the project is stepped through validation and cleansing of the dataset. The null value and duplicates are checked where no null records, as well as duplicates, are found in the data as expected because the input data is an image, and the pixel values are never null.

3.3.2 Normalization

In the next process, the pixel values are normalized. Before feeding the data into the neural network, we normalized the pixel values to a range between 0 and 1. This was achieved by dividing all pixel values by 255.0. This step is crucial because it scales the pixel values to a consistent range, which aids in the convergence of the training process.

```
1  X_train = train.drop(['label'], axis=1)
2  y_train = train['label']
```

```
1  X_train = X_train / 255
2  test = test / 255
```

Figure 7 Normalization of pixel values

3.3.3 Reshaping and One-Hot Encoding

The 784-pixel valued columns are reshaped into 28X28 images to provide input to the model. Also, the categorical target labels are one-hot encoded making it suitable for neural networks.

```
X_train = X_train.to_numpy('float32').reshape(42000,28,28)
y_train = to_categorical(y_train, 10)

X_train.shape, y_train.shape
```

```
((42000, 28, 28), (42000, 10))
```

Figure 8 Reshaping and one-hot encoding

3.4 Model Development and Training:

This project utilizes Python along with popular machine-learning libraries such as Pandas, NumPy, Matplotlib, Seaborn, OpenCV, and TensorFlow. In this project, three neural network models namely the multi-layer NN model, CNN, and CNN with batch normalized layer and dropout layer are formed. The first model consists of one input layer with an input dimension of (28, 28, 1) that represent the height, width, and color channel of an image, two hidden dense layers, and one output dense layer with 10 neurons representing 10 class labels in the dataset. The second is the CNN model which consists of two convolutional layers including input and output layers as in the previous model. In the third model, along with input, output, and convolution layers, numerous batch-normalized layers and one dropout layer are also added. Each model was defined with distinct architectures, including dense, and convolutional, and is compiled with appropriate loss functions and optimizers. Then, these models are trained on the preprocessed training data, using a batch size of 32 and running for 10 epochs. The training loop involved forward and backward propagation to update model parameters and minimize the loss. Additionally, the training process is monitored by visualizing training and validation loss as well as accuracy over epochs. This comprehensive approach to model development and training aimed to create an accurate and well-performing digit recognition model.

```
Model: "model_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_44 (InputLayer)       [(None, 28, 28, 1)]       0

 flatten_43 (Flatten)        (None, 784)               0

 dense_109 (Dense)           (None, 128)               100480

 dense_110 (Dense)           (None, 100)               12900

 dense_111 (Dense)           (None, 10)                1010

=================================================================
Total params: 114390 (446.84 KB)
Trainable params: 114390 (446.84 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 9 Model1 architecture - multi-layer neural network

```
Model: "model_41"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_45 (InputLayer)       [(None, 28, 28, 1)]       0

 conv2d_52 (Conv2D)          (None, 28, 28, 12)        204

 conv2d_53 (Conv2D)          (None, 14, 14, 15)        1635

 flatten_44 (Flatten)        (None, 2940)              0

 dense_112 (Dense)           (None, 10)                29410

=================================================================
Total params: 31249 (122.07 KB)
Trainable params: 31249 (122.07 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 10 Model2 architecture - convolution neural network (CNN)

```
Model: "model_42"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_46 (InputLayer)       [(None, 28, 28, 1)]       0

 conv2d_54 (Conv2D)          (None, 28, 28, 12)        204

 batch_normalization_30 (Ba  (None, 28, 28, 12)        48
 tchNormalization)

 leaky_re_lu_20 (LeakyReLU)  (None, 28, 28, 12)        0

 conv2d_55 (Conv2D)          (None, 14, 14, 10)        1930

 batch_normalization_31 (Ba  (None, 14, 14, 10)        40
 tchNormalization)

 leaky_re_lu_21 (LeakyReLU)  (None, 14, 14, 10)        0

 conv2d_56 (Conv2D)          (None, 14, 14, 12)        1092

 batch_normalization_32 (Ba  (None, 14, 14, 12)        48
 tchNormalization)

 leaky_re_lu_22 (LeakyReLU)  (None, 14, 14, 12)        0

 conv2d_57 (Conv2D)          (None, 7, 7, 10)          1930

 batch_normalization_33 (Ba  (None, 7, 7, 10)          40
 tchNormalization)

 leaky_re_lu_23 (LeakyReLU)  (None, 7, 7, 10)          0


 flatten_45 (Flatten)        (None, 490)               0

 dense_113 (Dense)           (None, 200)               98200

 batch_normalization_34 (Ba  (None, 200)               800
 tchNormalization)

 leaky_re_lu_24 (LeakyReLU)  (None, 200)               0

 dropout_6 (Dropout)         (None, 200)               0

 dense_114 (Dense)           (None, 10)                2010

=================================================================
Total params: 106342 (415.40 KB)
Trainable params: 105854 (413.49 KB)
Non-trainable params: 488 (1.91 KB)
_________________________________________________________________
```

Figure 11 Model3 architecture - CNN with batch normalization and dropout

B. Assigning Loss Function and Optimizer

```
opt = Adam(learning_rate=0.006)
model1.compile(loss="categorical_crossentropy", optimizer=opt, metrics=['accuracy'])
```

After developing model, the model is compiled with loss function as "categorical_crossentropy", optimizer function as "Adam", and metrics as "accuracy".

C. Model Training

```
history1 = model1.fit(X_train, y_train, batch_size=32, epochs=10, shuffle=True, validation_split=0.1)
```

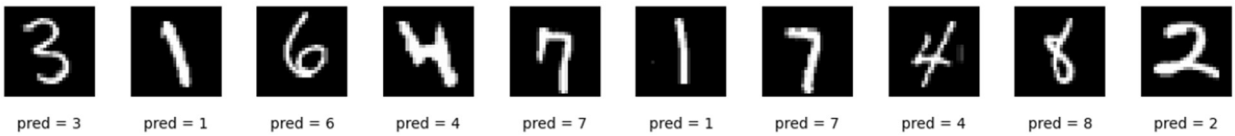
Figure 12 Model compilation and training

3.5 Data Visualization

Various data visualization steps have been employed to gain insights into the dataset and the model's performance. The performance measures such as training and validation loss, accuracy, confusion matrices, and examples of digit predictions are visualized over epochs, providing a comprehensive understanding of the model's training progress and performance. These visualization steps offer valuable insights into data and model behavior, assisting in model development, validation, and interpretation.

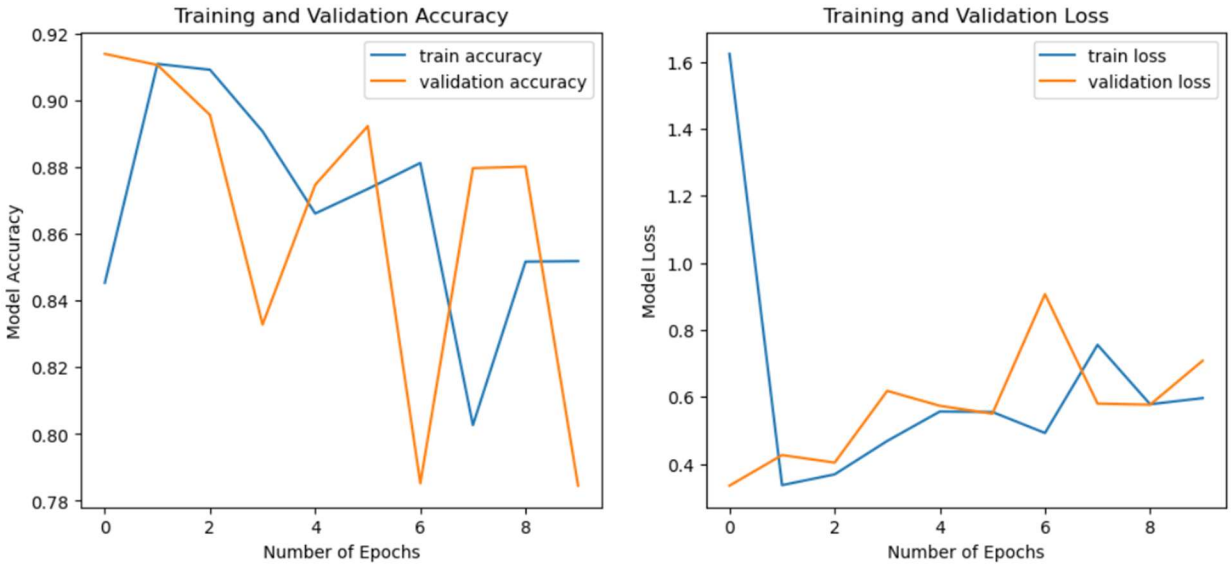
(a)

(b)

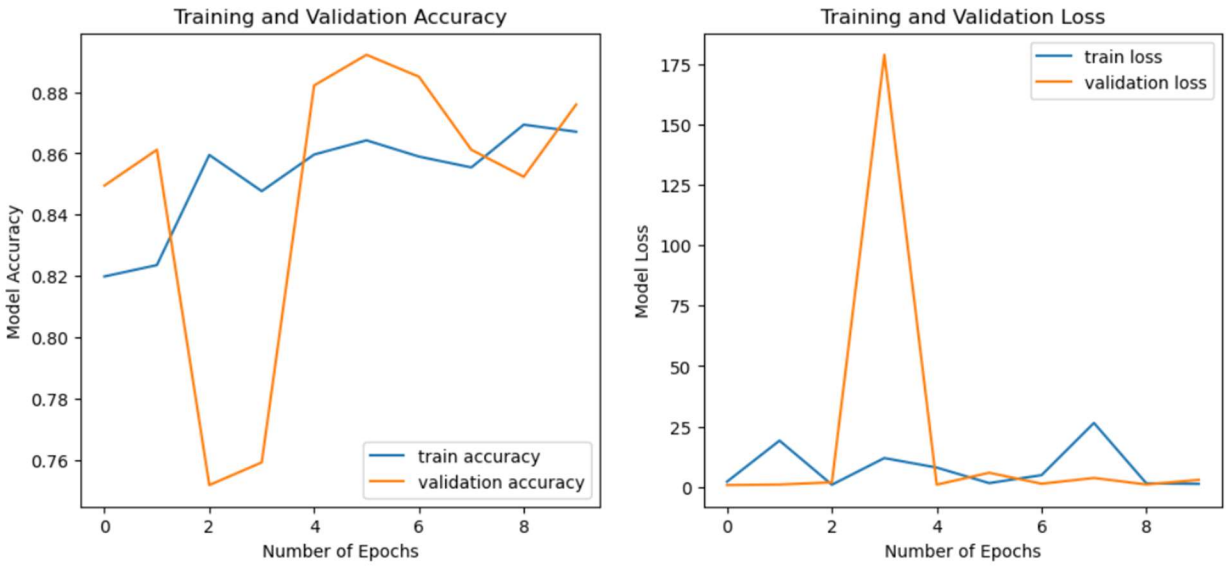


(c)

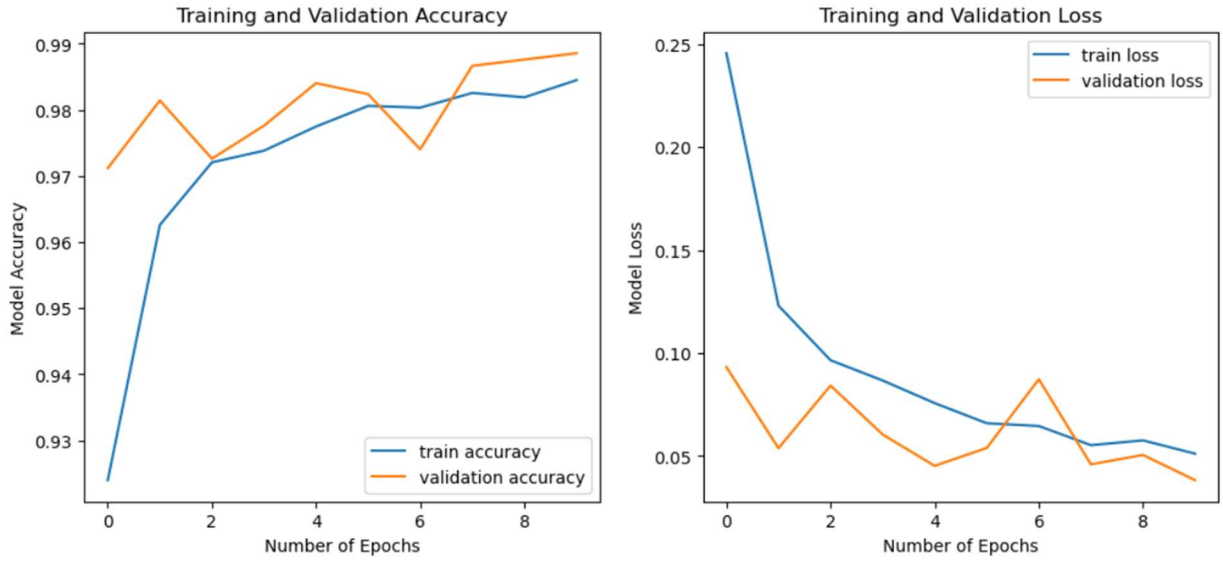
Figure 13 Prediction on the testing dataset (a) Model1 (b) Model2 (c) Model3



(a)

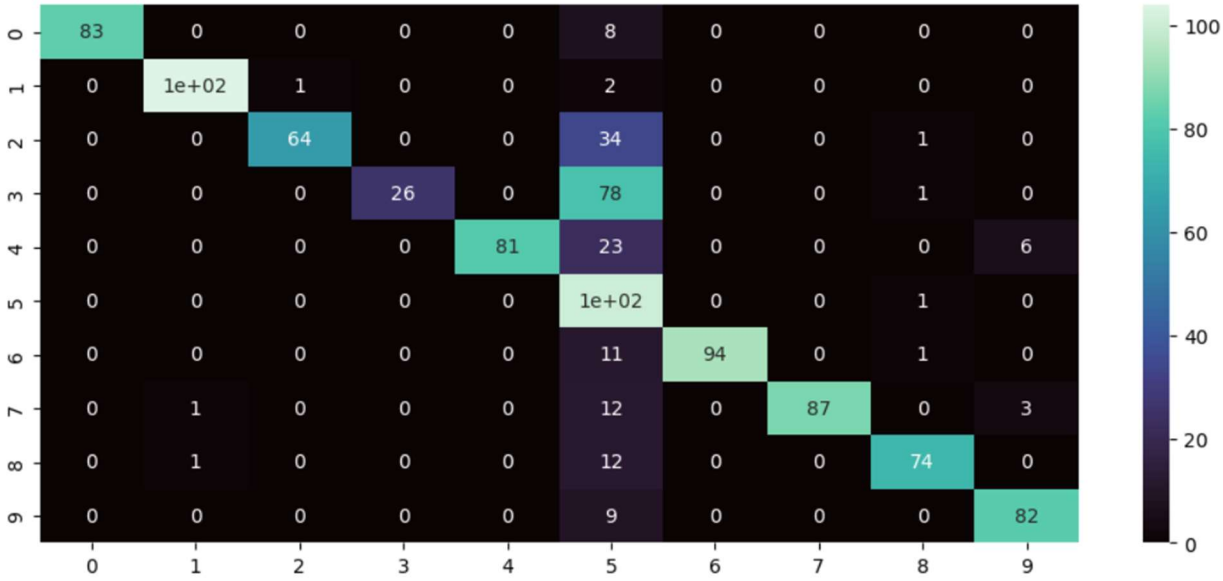


(b)

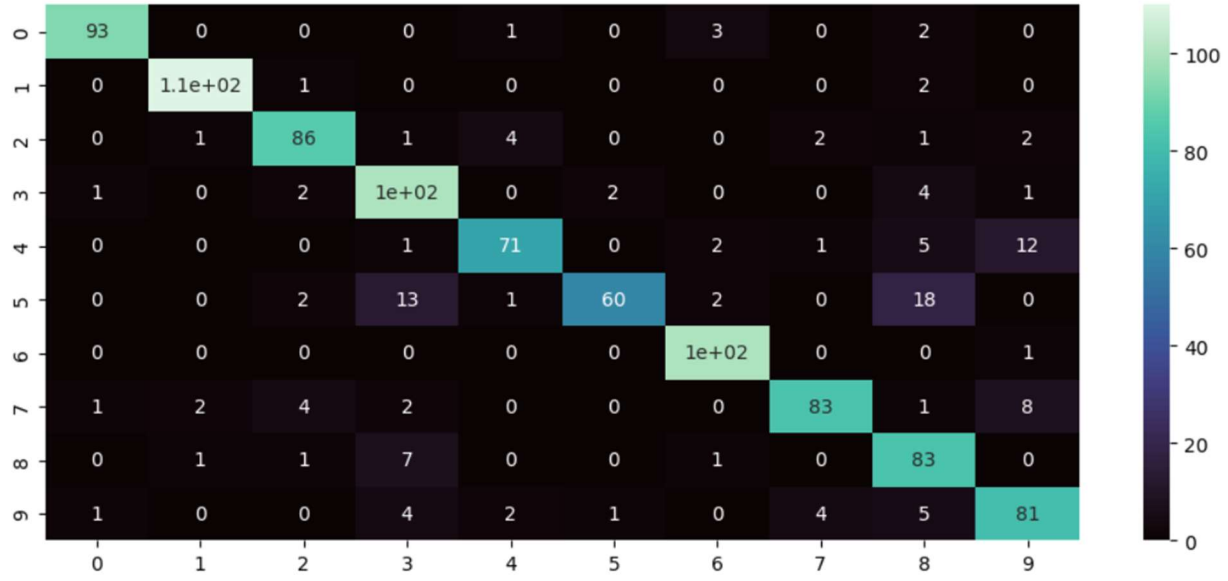


(c)

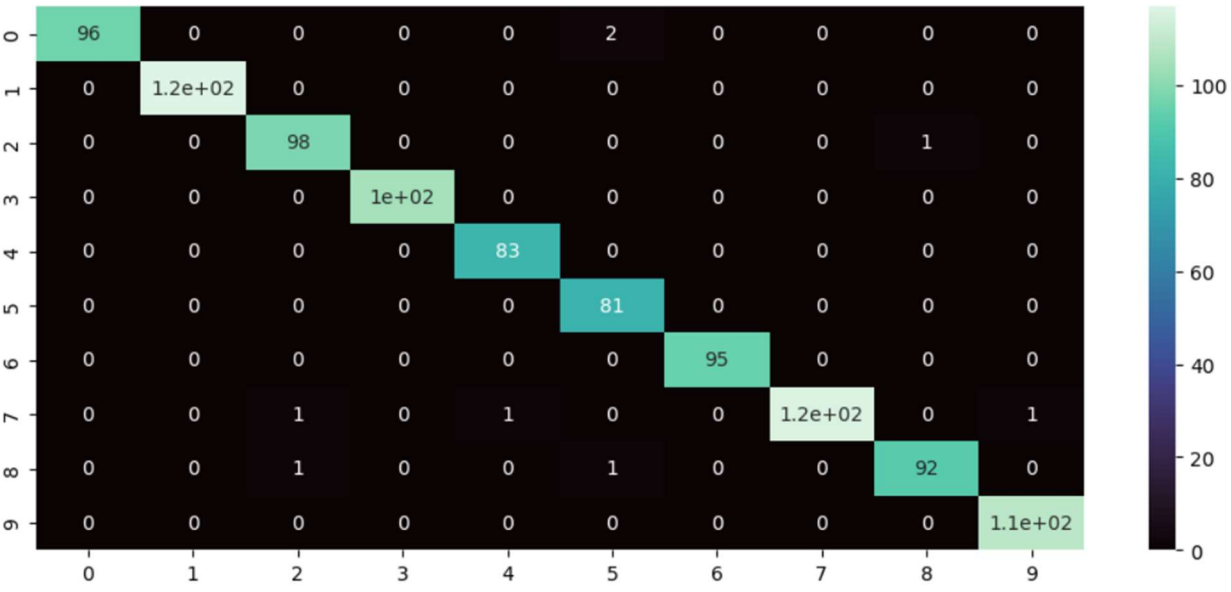
Figure 14 Training and validation accuracy and loss (a) Model1 (b) Model2 (c) Model3



(a)

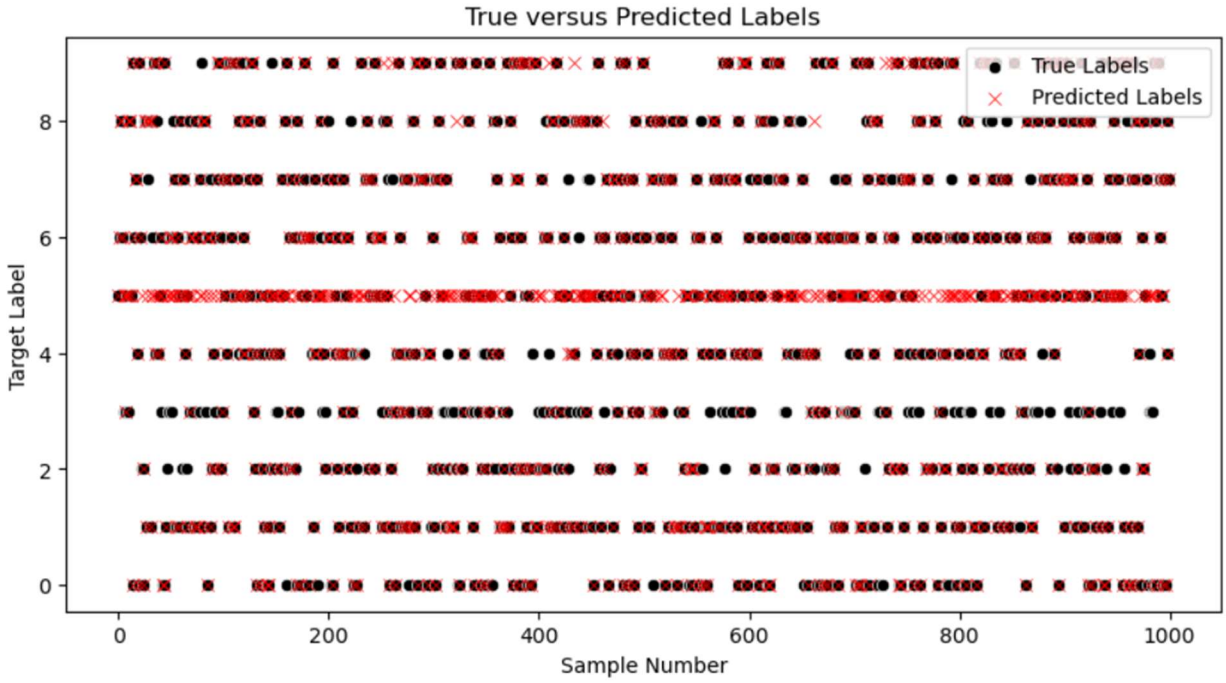


(b)

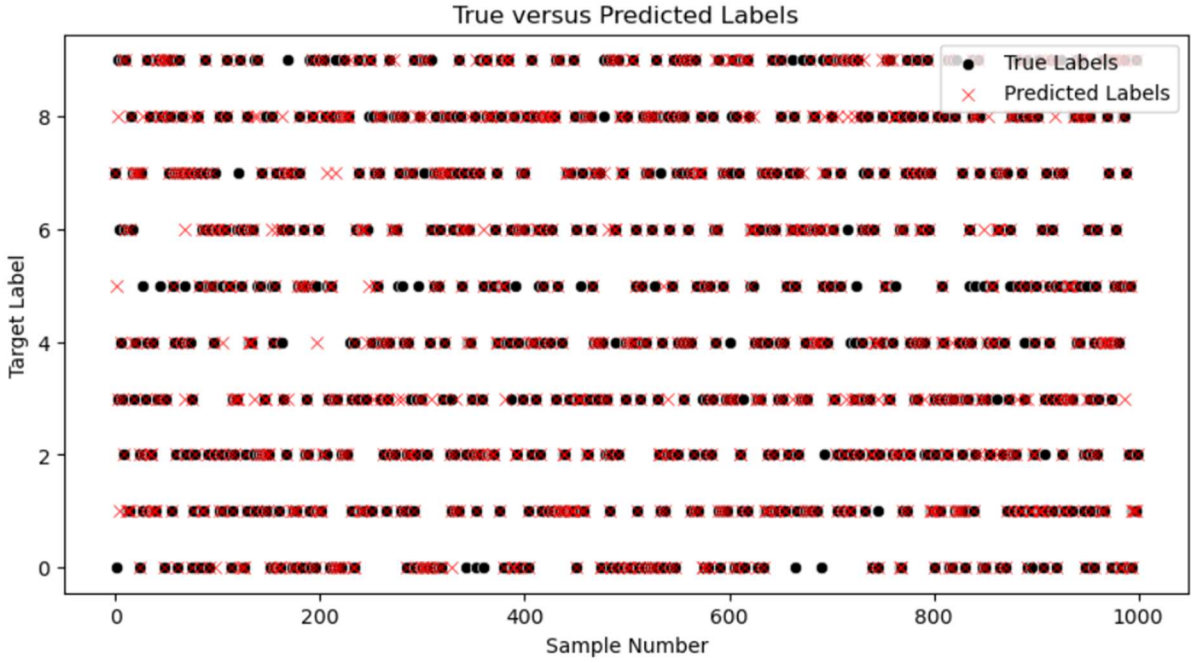


(c)

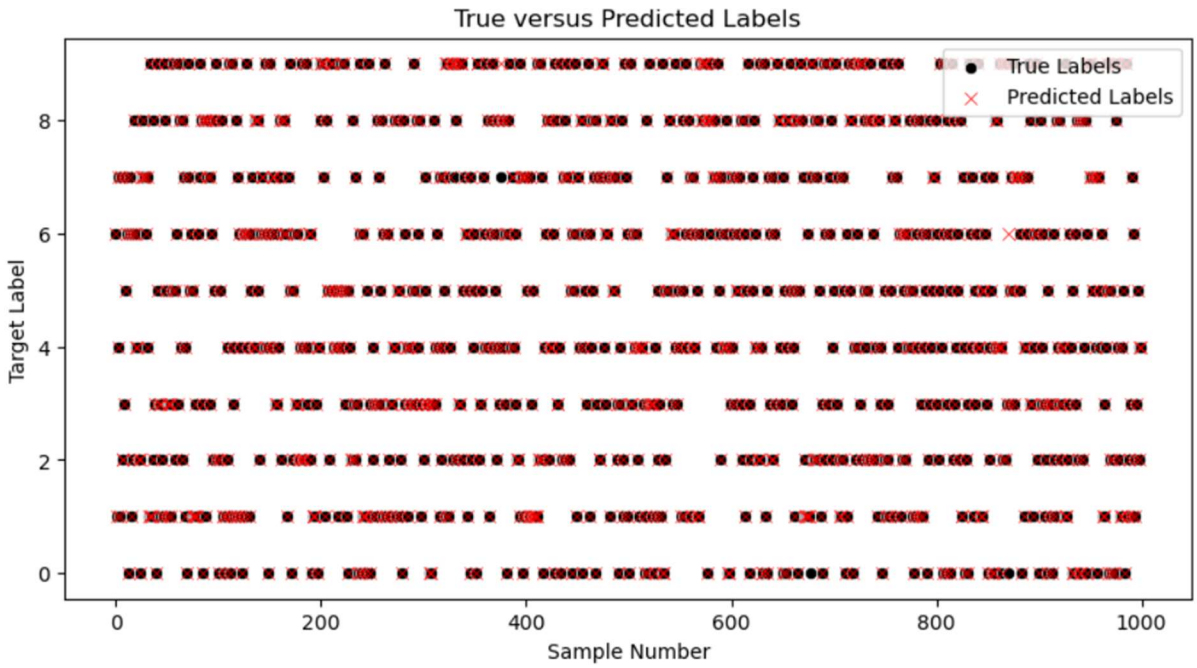
Figure 15 Confusion matrix heatmap (a) Model1 (b) Model2 (c) Model3


True versus Predicted Labels
True Labels
Predicted Labels
Target Label
Sample Number
0
2
4
6
8
0
200
400
600
800
1000

(a)


True versus Predicted Labels
True Labels
Predicted Labels
Target Label
Sample Number
0
2
4
6
8
0
200
400
600
800
1000

(b)



(c)

Figure 16 Actual versus predicted (a) Model1 (b) Model2 (c) Model3

CHAPTER 4: RESULTS AND DISCUSSION

4.1 Results

The results of the digit recognition experiments are presented for the different neural network architectures.

4.1.1 Model 1 – Multi-Layers Neural Network

For model1 i.e., multi-layers neural network, the training logs of the neural network model over 10 epochs are shown below. Each epoch corresponds to a complete pass through the training dataset. The "loss" and "accuracy" metrics are reported for both the training and validation sets. In the initial epoch (Epoch 1), the training loss is relatively high (1.6238), and the training accuracy is 84.54%. The validation loss and accuracy are more favorable (0.3353 and 91.40%, respectively). As training progresses, the model's training loss decreases, indicating that it is learning to fit the data better, but the validation loss begins to fluctuate and increase from the second epoch onwards, potentially indicating overfitting. The training accuracy remains high, while the validation accuracy varies, suggesting that the model might not generalize well to unseen data. These trends continue across subsequent epochs, with the model's performance gradually deteriorating, particularly in terms of validation accuracy. The final epoch shows a training loss of 0.5966, training accuracy of 85.19%, validation loss of 0.7084, and validation accuracy of 78.45%. Overall, the model appears to overfit the training data and struggles to perform well on unseen validation data, suggesting a need for further optimization or adjustments to prevent overfitting.

```
Epoch 1/10
1182/1182 [==============================] - 6s 4ms/step - loss: 1.6238 - accuracy: 0.8454 - val_loss: 0.3353 - val_accuracy:
0.9140
Epoch 2/10
1182/1182 [==============================] - 4s 4ms/step - loss: 0.3369 - accuracy: 0.9110 - val_loss: 0.4266 - val_accuracy:
0.9107
Epoch 3/10
1182/1182 [==============================] - 4s 4ms/step - loss: 0.3690 - accuracy: 0.9093 - val_loss: 0.4041 - val_accuracy:
0.8957
Epoch 4/10
1182/1182 [==============================] - 4s 4ms/step - loss: 0.4688 - accuracy: 0.8908 - val_loss: 0.6183 - val_accuracy:
0.8329
Epoch 5/10
1182/1182 [==============================] - 4s 3ms/step - loss: 0.5565 - accuracy: 0.8662 - val_loss: 0.5736 - val_accuracy:
0.8748
Epoch 6/10
1182/1182 [==============================] - 4s 4ms/step - loss: 0.5552 - accuracy: 0.8735 - val_loss: 0.5501 - val_accuracy:
0.8924
Epoch 7/10
1182/1182 [==============================] - 4s 4ms/step - loss: 0.4925 - accuracy: 0.8813 - val_loss: 0.9067 - val_accuracy:
0.7852
Epoch 8/10
1182/1182 [==============================] - 4s 3ms/step - loss: 0.7556 - accuracy: 0.8027 - val_loss: 0.5802 - val_accuracy:
0.8798
Epoch 9/10
1182/1182 [==============================] - 4s 4ms/step - loss: 0.5783 - accuracy: 0.8517 - val_loss: 0.5770 - val_accuracy:
0.8802
Epoch 10/10
1182/1182 [==============================] - 4s 3ms/step - loss: 0.5966 - accuracy: 0.8519 - val_loss: 0.7084 - val_accuracy:
0.7845
```

Figure 17 Training log for Model1

4.1.2 Model 2 - Convolutional Neural Network (CNN)

Model 2 with CNNs, shows significant improvement in accuracy and generalization. During the initial epoch (Epoch 1) in this model, the training loss is high (2.2308), and the training accuracy is 81.99%. The validation loss and accuracy are relatively better (0.8021 and 84.95%, respectively). However, as training progresses, the model's behavior becomes erratic. In some epochs, the training loss increases substantially (e.g., Epoch 2), and in other epochs, it reaches extremely high values (e.g., Epoch 4). Validation performance also exhibits fluctuations and inconsistencies. Compared to the previous model, this model's performance is less stable and likely to suffer from optimization issues or other problems that hinder its learning ability.

```
Epoch 1/10
1182/1182 [==============================] - 16s 13ms/step - loss: 2.2308 - accuracy: 0.8199 - val_loss: 0.8021 - val_accuracy:
0.8495
Epoch 2/10
1182/1182 [==============================] - 14s 12ms/step - loss: 19.1827 - accuracy: 0.8236 - val_loss: 1.0135 - val_accurac
y: 0.8612
Epoch 3/10
1182/1182 [==============================] - 15s 13ms/step - loss: 0.9425 - accuracy: 0.8595 - val_loss: 1.9316 - val_accuracy:
0.7519
Epoch 4/10
1182/1182 [==============================] - 16s 14ms/step - loss: 11.9680 - accuracy: 0.8477 - val_loss: 178.7915 - val_accura
cy: 0.7593
Epoch 5/10
1182/1182 [==============================] - 14s 12ms/step - loss: 8.0316 - accuracy: 0.8596 - val_loss: 0.9700 - val_accuracy:
0.8821
Epoch 6/10
1182/1182 [==============================] - 15s 13ms/step - loss: 1.6127 - accuracy: 0.8643 - val_loss: 5.8958 - val_accuracy:
0.8921
Epoch 7/10
1182/1182 [==============================] - 17s 14ms/step - loss: 4.9000 - accuracy: 0.8590 - val_loss: 1.3578 - val_accuracy:
0.8850
Epoch 8/10
1182/1182 [==============================] - 14s 12ms/step - loss: 26.4931 - accuracy: 0.8554 - val_loss: 3.7322 - val_accurac
y: 0.8612
Epoch 9/10
1182/1182 [==============================] - 14s 12ms/step - loss: 1.5006 - accuracy: 0.8694 - val_loss: 1.0377 - val_accuracy:
0.8524
Epoch 10/10
1182/1182 [==============================] - 14s 12ms/step - loss: 1.3001 - accuracy: 0.8671 - val_loss: 2.9696 - val_accuracy:
0.8760
```

Figure 18 Training log for Model2

4.1.3 Model 3 - CNN with Batch Normalization and Dropout Regularization

This model started with a training loss of 0.2456 and an accuracy of approximately 92.40% in the first epoch. As training progresses, the loss consistently decreases, reaching 0.0512 by the 10th epoch. The accuracy also improved gradually, achieving around 98.45% by the end. The validation metrics followed a similar trend, with validation loss decreasing from 0.0932 to 0.0383 and validation accuracy increasing from 97.12% to 98.86% over the 10 epochs. This indicates that the model effectively learned to classify handwritten digits and performed well on both training and validation data, demonstrating strong accuracy and good generalization to new data. Also, the convergence speed and stability of this model are very well. Overall, the CNN model with batch normalization and dropout regularization outperforms traditional neural networks and CNN models in terms of accuracy, stability, and convergence.

```
Epoch 1/10
1182/1182 [==============================] - 37s 29ms/step - loss: 0.2456 - accuracy: 0.9240 - val_loss: 0.0932 - val_accuracy:
0.9712
Epoch 2/10
1182/1182 [==============================] - 35s 30ms/step - loss: 0.1230 - accuracy: 0.9626 - val_loss: 0.0538 - val_accuracy:
0.9814
Epoch 3/10
1182/1182 [==============================] - 36s 30ms/step - loss: 0.0965 - accuracy: 0.9721 - val_loss: 0.0842 - val_accuracy:
0.9726
Epoch 4/10
1182/1182 [==============================] - 33s 28ms/step - loss: 0.0867 - accuracy: 0.9738 - val_loss: 0.0605 - val_accuracy:
0.9776
Epoch 5/10
1182/1182 [==============================] - 39s 33ms/step - loss: 0.0757 - accuracy: 0.9775 - val_loss: 0.0453 - val_accuracy:
0.9840
Epoch 6/10
1182/1182 [==============================] - 37s 31ms/step - loss: 0.0659 - accuracy: 0.9806 - val_loss: 0.0540 - val_accuracy:
0.9824
Epoch 7/10
1182/1182 [==============================] - 35s 30ms/step - loss: 0.0646 - accuracy: 0.9803 - val_loss: 0.0873 - val_accuracy:
0.9740
Epoch 8/10
1182/1182 [==============================] - 36s 30ms/step - loss: 0.0553 - accuracy: 0.9826 - val_loss: 0.0460 - val_accuracy:
0.9867
Epoch 9/10
1182/1182 [==============================] - 36s 31ms/step - loss: 0.0576 - accuracy: 0.9819 - val_loss: 0.0505 - val_accuracy:
0.9876
Epoch 10/10
1182/1182 [==============================] - 36s 30ms/step - loss: 0.0512 - accuracy: 0.9845 - val_loss: 0.0383 - val_accuracy:
0.9886
```

Figure 19 Training log for Model3

4.2 Discussion

This project showcases the process of building, training, evaluating, saving, and loading neural network models for digit recognition using different architectures. Additionally, various visualization techniques are employed to gain insights into model performance and predictions. The trained model has achieved very good accuracy and it also predicted the labels of unseen data correctly. The project has met expectations as the accuracy for the final model is almost ~99% which is ideal for any real-field applications and this model has improved generalization.

CHAPTER 5: CONCLUSIONS AND FUTURE WORKS

5.1 Conclusion

In this project, various neural network architectures are successfully explored for digit recognition. The results demonstrate the effectiveness of convolutional neural networks in improving accuracy and generalization. Batch normalization and regularization techniques further enhance the model's performance. A high-performing CNN model is developed in this project that recognizes the digits accurately. At first, the multi-layers neural network is developed where the predictions are not that accurate. Then, the CNN model is developed and trained where the performance is a little bit better than the first model, but it is not that good. Furthermore, the CNN model is implemented with batch normalization and dropout layers to optimize its performance where this model can predict the digits very accurately with an accuracy of almost 99%.

5.2 Future Works

The project results are astonishing as the final model accuracy is 98.5% ~ 99%, however, the project can be further improved through different ways. There is always room for improvement, and future works for the project are the following:

- Fine-tuning hyperparameters to achieve the optimal settings for the model.
- Exploring different augmentation techniques to enhance the dataset.
- Exploring different advanced algorithms such as ResNet, DenseNet, or EfficientNet, to further improve model accuracy.
- For making the model more robust, play with different ensemble learning techniques.
- Integrating this project into real-time applications such as real-time object detection, number plate detection, OCR, etc.

REFERENCES

1. AstroDave, & Will Cukierski. (2012). Digit Recognizer. Kaggle. https://kaggle.com/competitions/digit-recognizer
2. R. Kanniga Devi; G. Elizabeth Rani. (2019). A Comparative Study on Handwritten Digit Recognizer using Machine Learning Technique. 2019 IEEE International Conference on Clean Energy and Energy Efficient Electronics Circuit for Sustainable Development. Krishnankoil, India: IEEE. https://ieeexplore.ieee.org/abstract/document/9167748
3. Renata F. P. Neves; Alberto N. G. Lopes Filho; Carlos A.B. Mello; Cleber Zanchettin. (2011). An SVM-based off-line handwritten digit recognizer. 2011 IEEE International Conference on Systems, Man, and Cybernetics. Anchorage, AK, USA: IEEE. https://ieeexplore.ieee.org/abstract/document/6083734
4. Foster, D. (2019). Generative Deep Learning. 1005 Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc.
5. Keras. (n.d.). Getting started with the Keras functional API. https://keras.io/getting_started/
6. Matplotlib. (n.d.). Tutorials. https://matplotlib.org/stable/tutorials/index