MINI PROJECT REPORT
ON
HANDWRITTEN DIGITRECOGNIZER

A report submitted in partial fulfillment of the requirements of

BACHELOR OF TECHNOLOGY
in
APPLIED COMPUTATIONAL SCIENCE AND ENGINEERING
by

Name: Arushi Tariyal

Roll No: 2101921540014

Under the Supervision of:

Ms. Vipasha Abrol

(Duration: From 2023 to 2024)

G.L. BAJAJ INSTITUTE OF TECHNOLOGY & MANAGEMENT,
GREATER NOIDA
Affiliated to

DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY,
LUCKNOW

2023-2024

Declaration

I hereby declare that the internship work presented in this report entitled "Handwritten Digit Recognizer", in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Applied Computational Science & Engineering, submitted to A.P.J. Abdul Kalam Technical University, Lucknow, is based on my own work carried out at Department of Applied Computational Science & Engineering, G.L. Bajaj Institute of Technology & Management, Greater Noida. The work contained in the report is true and original to the best of my knowledge and project work reported in this report has not been submitted by me for award of any other degree or diploma.

Signature

Name: Arushi Tariyal

Roll No: 2101921540014

G.L. BAJAJ INSTITUTE OF TECHNOLOGY & MANAGEMENT, GREATER NOIDA

DEPARTMENT OF APPLIED COMPUTATIONAL SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the **"Mini Project Report"** entitled **"Handwritten Digit Recognizer"** is being done by Arushi Tariyal (2101921540014) in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY in Computer Science & Engineering** for the academic session 2023-2024, under my guidance. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Date:

Ms. Vipasha Abrol
Mini Project Coordinator

ACKNOWLEDGEMENT

I humbly pay my thanks to my project guide **Ms. Vipasha Abrol** who has given guidance and light to us during this project. Her versatile knowledge has caused me in the critical times during the span of this project.

I pay special thanks to our Head of Department **Dr. Sansar Singh Chauhan** who has been always present as a support and help me in all possible way during this project.

I pay special thanks to the Dean of the department **Prof.(Dr.) Naresh Kumar** for his constructive criticism throughout my project.

I also take this opportunity to express our gratitude to all those people who have been directly and indirectly with us during the completion of the project.

I want to thank my friends who have always encouraged me during this project. At the last but not least thanks to all the faculty of CSE department who provided valuable suggestions during the period of project.

Name: Arushi Tariyal
Roll No.: 2101921540014

GL BAJAJ
Institute of Technology & Management
Greater Noida

[Approved by AICTE, Govt. of India & Affiliated to Dr. APJ Abdul Kalam Technical University, Lucknow, U.P., India]
Department of Applied Computational Science & Engineering

Mini Project Feedback Form

Student Name:_____ Company Name:_____

Contact Number:_____ Title:_____

Duration of Project:_____ Email:_____

Summary of student's activities and responsibilities:

_____

_____

Performance Areas	Excellent	Above Average	Average	Below Average
Basic Engineering Knowledge				
Problem Analysis				
Design/Development Skills				
Solving complex problems using research-based techniques				
Familiarity with Modern Tools				
Engineer and Society				
Awareness on Environment and Sustainability				
Professional Ethics for Engineering Practice				
Individual and Teamwork				
Communication on complex engineering activities.				
Project Management in multidisciplinary environments				
Life-Long Learning				
Use Problem solving skill to develop efficient algorithmic solution				
Develop a solution in the area for AI, Data Analytics, Computer Vision and IoT etc.				

Based on the above observations, we put forward few suggestions regarding this Mini Project program:

_____

_____

Supervisor's Signature:

Name:

Designation:

Official Email_Id…

Revolutionizing Pattern Recognition:
A Comprehensive Report to Machine Learning for Handwritten Digit Recognition

Abstract:

Unleashing the Power of Machine Learning in Handwritten Digit Recognition

In the vast landscape of artificial intelligence, one of the most intriguing and impactful applications is the recognition of handwritten digits through machine learning algorithms. This article delves into the fascinating realm of handwritten digit recognition, exploring the evolution of the field, the underlying machine learning techniques, and the applications that have reshaped industries. From the fundamentals of image processing to the intricacies of convolutional neural networks, we embark on a journey to understand how machines learn to decipher the complex patterns of handwritten characters.

In an era dominated by digital transformation, the ability to decipher and understand the intricacies of human handwriting has taken center stage in the realm of artificial intelligence. Handwritten digit recognition, a seemingly simple task for humans, poses a formidable challenge for machines due to the inherent variability in writing styles. However, with the advent of machine learning (ML), particularly advanced techniques like convolutional neural networks (CNNs), the landscape of handwritten digit recognition has undergone a revolutionary metamorphosis.

In this project, we aim to correctly identify digits from a dataset of variety of handwritten images. Kaggle has curated a set of tutorial-style kernels which cover everything from regression to neural networks. They hope to encourage us to experiment with different algorithms to learn first-hand what works well and how techniques compare

TABLE OF CONTENT

REFERENCES:

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction
The Power of Handwritten Digit Recognition

1.1 Background and Significance

Handwritten digit recognition has emerged as a cornerstone in the field of machine learning, offering versatile applications ranging from digitizing historical documents to facilitating automated data entry.

The historical journey of handwritten digit recognition mirrors the evolution of computing and artificial intelligence. In earlier decades, the recognition of handwritten characters relied heavily on manual methods and rule-based systems. These approaches struggled to adapt to the diverse and nuanced nature of human handwriting. The breakthrough came with the advent of machine learning, marking a paradigm shift from explicit programming to systems that could learn and adapt autonomously.

1.2 Evolution of Handwritten Digit Recognition

Tracing the evolution of handwritten digit recognition, we explore the key milestones that have shaped its development. Handwritten digit recognition, a fascinating intersection of artificial intelligence and pattern recognition, has undergone a remarkable evolution over the years. This section delves into the key milestones and technological shifts that have shaped the development of handwritten digit recognition, transforming it from early manual methods to sophisticated machine learning models.

In the rapidly evolving landscape of the digital age, the ability to recognize handwritten digits holds profound significance, touching upon various aspects of technology, communication, and efficiency. As we transition from traditional pen-and-paper methods to digital interfaces, the recognition of handwritten digits emerges as a pivotal enabler in numerous domains

1.3 Role of MNIST

MNIST ("Modified National Institute of Standards and Technology") is the de facto “Hello World” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

The MNIST dataset, a cornerstone in the field of machine learning, serves as a benchmark for developing and evaluating models designed for handwritten digit recognition. Its significance lies not only in its simplicity but also in its ability to represent real-world challenges encountered in various applications.

1.4 Overview of the model

1.4.1 Keras

Keras is an open-source deep learning framework written in Python. It serves as a high-level neural networks API, enabling easy and efficient development of deep learning models. Keras is built on top of other popular deep learning libraries such as TensorFlow and Theano, allowing users to take advantage of their capabilities while providing a simplified interface for constructing and training neural networks.

We will be using Keras (with TensorFlow as our backend) as the main package to create a simple neural network to predict, as accurately as we can, digits from handwritten images. In particular, we will be calling the Functional Model API of Keras, and creating a 4-layered and 5-layered neural network.

1.4.2 Optimizers

In machine learning, optimizers are algorithms or methods used to minimize (or maximize) an objective function, also known as a loss function. The objective function represents the error or the difference between the predicted output and the actual output in a supervised learning problem. The optimization process involves adjusting the model's parameters to minimize this error.

We will be experimenting with various optimizers: the plain vanilla Stochastic Gradient Descent optimizer and the Adam optimizer. However, there are many other parameters, such as training epochs which will we will not be experimenting with.

1.4.3 Regularization

Regularization is a technique used in machine learning to prevent overfitting and improve the generalization of a model. Overfitting occurs when a model learns the training data too well, capturing noise or random fluctuations in the data rather than the underlying patterns. Regularization introduces a penalty term to the objective function, discouraging the model from becoming too complex or fitting the noise in the training data.

We introduce dropout, a form of regularisation, in our neural networks to prevent overfitting.

Chapter 2: Problem Statement
The Need of Developing Handwritten Digit Recognizer

2.1 Sectors of Requirement

The development of handwritten digit recognition models serves several important purposes and has various applications. Here are some key reasons for developing such models:

1. Automation in Data Entry:

Handwritten digit recognition models can be employed to automate the process of digitizing handwritten documents. This is particularly useful in scenarios where large volumes of data need to be converted from paper to digital format.

2. Postal Services:

Handwritten digit recognition is crucial in postal services for automatically reading addresses and zip codes on mail envelopes. This helps in sorting and routing the mail efficiently.

3. Banking and Finance:

In banking and finance, handwritten digit recognition can be applied to recognize handwritten checks, forms, or other financial documents, facilitating faster and more accurate processing.

4. Educational Technology:

Handwritten digit recognition models are valuable in educational technology, such as grading handwritten exams or automatically converting handwritten notes into digital format.

5. Accessibility:

Handwritten digit recognition technology can contribute to making technology more accessible for individuals with disabilities. For

example, it can be used in devices that allow people with limited mobility to input information using handwritten gestures.

6. Digital Signature Verification:

Handwritten digit recognition can be employed in systems for verifying digital signatures, ensuring the authenticity of handwritten signatures in digital documents.

7. Healthcare:

In healthcare, handwritten digit recognition can be used for reading and interpreting handwritten medical forms, prescriptions, or patient records, contributing to more accurate and efficient healthcare management.

8. Human-Computer Interaction:

Handwritten digit recognition can enhance human-computer interaction, enabling users to input information using handwriting in devices like tablets and touchscreens.

9. Document Analysis:

Handwritten digit recognition is part of a broader field of document analysis, which involves extracting meaningful information from handwritten documents, forms, or historical records.

10. Pattern Recognition Research:

Handwritten digit recognition serves as a fundamental problem in pattern recognition research. Developing accurate models for this task contributes to advancements in machine learning and artificial intelligence.

11. Security:

Handwritten digit recognition can be employed in security applications, such as signature verification for access control or authorization purposes

2.2 The Existing System

Manual methods of handwriting recognition involve the use of human effort and expertise to interpret and recognize handwritten text or characters. These methods are often employed in situations where automated or computer-based recognition systems may not be suitable or available. Here are some manual methods of handwriting recognition:

1. Human Transcription:

One of the most straightforward methods is to have human transcribers read and manually transcribe handwritten text. This is a labor-intensive process but can be accurate, especially when dealing with complex or historical scripts.

2. Expert Analysis:

Handwriting experts, such as forensic document examiners, are trained to analyze and compare handwritten samples. They can provide insights into the characteristics of a person's handwriting and make determinations about the authorship or authenticity of a document.

3. Crowdsourcing:

Leveraging the power of crowds through platforms like Amazon Mechanical Turk, where multiple individuals can contribute to the transcription or recognition of handwritten text. This method is often used for large-scale data annotation tasks.

4.Template Matching:

Creating templates of known characters or symbols and manually matching them to handwritten text. This method is more suitable for recognizing specific symbols or characters rather than entire words or sentences.

5. Transcription by Multiple Experts:

Having multiple human transcribers independently transcribe the same piece of handwritten text and then comparing their results can enhance accuracy and reliability.

6. Transcription Guidelines:

Providing guidelines and training materials to human transcribers can help standardize the recognition process, ensuring consistency in how different individuals interpret and transcribe handwriting.

7. Interactive Transcription Tools:

Using interactive tools that allow users to manually transcribe and correct the recognition results. These tools may provide suggestions or predictions based on partial input, aiding the human transcriber in the process.

8. Educational Programs:

Teaching individuals to recognize and transcribe specific scripts or languages through education and training programs. This is often done in the context of historical document analysis or the study of ancient languages.

While manual methods have their merits, they are generally more time-consuming and resource-intensive compared to automated handwriting recognition systems. Automated systems, especially those based on machine learning and artificial intelligence, have made significant advancements in recent years, offering faster and more scalable solutions for handwriting recognition tasks. However, manual methods remain valuable in certain contexts, particularly when dealing with unique or challenging handwriting styles.

Chapter 3: Plan of Work
Prerequisites for Structuring the Model

3.1 Tools and Technologies Required

- **Python**

  Python, a versatile and powerful programming language, has emerged as a cornerstone in the world of software development. Renowned for its clean and readable syntax, Python caters to a diverse audience, from novice programmers to seasoned developers. Introduced by Guido van Rossum in 1991, Python prioritizes simplicity, making it an ideal language for rapid prototyping and efficient coding.

  The language's success is underscored by a robust community and a rich ecosystem of third-party libraries and frameworks. Its applications span various domains, including web development with frameworks like Django and Flask, data science using NumPy and Pandas, and machine learning through TensorFlow and PyTorch.

- **Keras**

  Keras, an influential high-level neural networks API written in Python, has significantly shaped the landscape of deep learning development. Designed for ease of use and rapid prototyping, Keras provides a user-friendly interface that abstracts complex deep learning concepts, making it accessible to both beginners and experienced researchers. Originally developed as an independent library with compatibility for multiple backend frameworks like TensorFlow.

  One of Keras' defining features is its modularity, allowing users to construct neural network models as sequences of customizable layers. This design promotes flexibility and ease in assembling intricate architectures for tasks such as image classification, natural language processing, and more. Keras' compatibility with various data formats and its support for convolutional and recurrent neural networks make it a versatile choice for a wide range of machine learning applications.

- **TensorFlow**

  TensorFlow is an open-source machine learning library that facilitates the development and deployment of machine learning models. Developed by the Google Brain team, TensorFlow provides a comprehensive ecosystem for building

and training various types of machine learning models, with a particular emphasis on deep learning. One of its key features is the ability to define, optimize, and deploy complex neural network architectures efficiently. TensorFlow supports a range of platforms and devices, making it versatile for tasks such as image and speech recognition, natural language processing, and more. Its flexibility and scalability have contributed to its widespread adoption in both research and industry, enabling developers to implement cutting-edge machine learning solutions. TensorFlow offers high-level APIs for quick model prototyping, as well as lower-level APIs for fine-grained control over model architecture and training processes. Overall, TensorFlow has become a cornerstone in the field of machine learning, empowering developers to explore and implement innovative solutions for various real-world challenges.

One of TensorFlow's defining features is its accessibility. With high-level APIs like Keras, developers can construct and train intricate neural network architectures with minimal code, making it approachable for beginners and efficient for experts. The framework's compatibility with Python, a popular programming language in the machine learning community, further contributes to its ease of use.

- **MNIST Dataset**

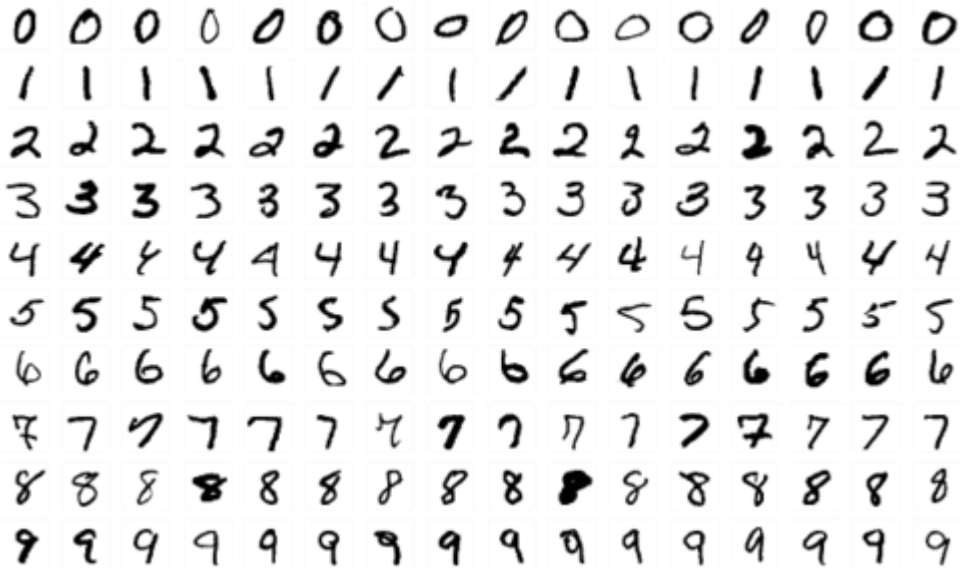

Fig 3.1

The MNIST dataset, short for Modified National Institute of Standards and Technology, has emerged as a cornerstone in the realm of machine learning, particularly in the domain of image classification. This dataset has played a pivotal role in shaping the landscape of artificial intelligence research and development. Comprising 28x28 pixel grayscale images of handwritten digits ranging from 0 to

9, MNIST has become a benchmark for testing the efficacy of various machine learning algorithms.

Originally created by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges, the MNIST dataset was introduced as a modified version of the NIST dataset to provide a standardized benchmark for handwritten digit recognition. Since its inception, MNIST has become a touchstone for researchers, serving as a common ground for comparing and evaluating different models and techniques.

MNIST boasts a collection of 70,000 images, with 60,000 designated for training and 10,000 for testing. Each image is a small square of 28x28 pixels, representing a single digit written by various individuals. The simplicity of the dataset makes it an ideal starting point for researchers and practitioners delving into the intricacies of image classification.

- **Jupyter Notebook**

Jupyter Notebook, an open-source web application, has become an indispensable tool in the world of interactive computing and data science. Originally developed as part of the Jupyter Project, the notebook format has gained widespread adoption due to its flexibility, ease of use, and support for various programming languages.

Jupyter Notebook supports multiple programming languages, including Python, R, Julia, and others. This versatility allows researchers and data scientists to choose the language that best suits their needs while seamlessly integrating different languages within the same document. The ability to mix code from various languages fosters collaboration and encourages interdisciplinary research.

3.2 Concepts Assimilated

- **Neural Network**

Neural networks, a fundamental concept in artificial intelligence, have revolutionized the field with their ability to mimic the human brain's learning process. These computational models play a crucial role in various applications, ranging from image and speech recognition to natural language processing and autonomous vehicles. In this essay, we will explore the intricacies of neural networks, their architecture, training process, and diverse applications.

At its core, a neural network is a computational model inspired by the structure and

functioning of the human brain. It is composed of interconnected nodes, commonly referred to as neurons or artificial neurons. These nodes are organized into layers – an input layer, one or more hidden layers, and an output layer. The connections between neurons, known as weights, determine the strength of the signal between them.

Architecture of a Neural Network:

Input Layer: This layer receives the initial data or input features, which serve as the information for the network.

Hidden Layers: These layers process the input data through weighted connections and activation functions, extracting complex patterns and representations.

Output Layer: The final layer produces the network's output, which could be a classification, regression, or some other type of prediction.

Types of Neural Networks:

Feedforward Neural Networks: Information flows in one direction, from the input layer to the output layer, without cycles.

Recurrent Neural Networks (RNNs): These networks have connections that form cycles, enabling them to capture temporal dependencies in data, making them suitable for sequential data like time series.

Convolutional Neural Networks (CNNs): Primarily designed for image processing, CNNs use convolutional layers to extract hierarchical features.

Applications:

Image and Speech Recognition: Neural networks power image recognition systems, enabling applications like facial recognition and object detection. Similarly, they excel in speech recognition, enhancing virtual assistants and voice-operated devices.

Natural Language Processing: In NLP, neural networks process and understand human language, facilitating tasks like sentiment analysis, language translation, and chatbot interactions.

Autonomous Vehicles: Neural networks contribute to the development of self-driving cars by processing sensor data, identifying objects, and making real-time

decisions.

Healthcare: In medical applications, neural networks aid in disease diagnosis, predicting patient outcomes, and analyzing medical images.

- **Optimizers**

Optimizers play a pivotal role in the realm of machine learning, acting as the driving force behind the refinement of models. The field of machine learning is inherently a quest for optimization—finding the optimal set of parameters that minimizes the error and maximizes the model's predictive power. In this essay, we will delve into the significance of optimizers, exploring their types, functions, and impact on the training process.

The Optimization Objective:
At the core of machine learning lies the optimization objective: to minimize the discrepancy between predicted outputs and actual outcomes. This objective is encapsulated in an objective function or a loss function, representing the quantifiable measure of the model's performance. Optimizers are algorithms designed to traverse the vast parameter space of a model, iteratively adjusting weights and biases to minimize this loss function.

We will be experimenting with various optimizers: the plain vanilla Stochastic Gradient Descent optimizer and the Adam optimizer.

1. Plain Vanilla Stochastic Gradient Descent Optimizer:
Stochastic Gradient Descent (SGD) is a foundational optimization algorithm in the field of machine learning, often referred to as "Plain Vanilla" SGD to emphasize its simplicity and effectiveness.

At its core, Plain Vanilla SGD is a gradient-based optimization algorithm used to minimize the error, or loss, of a model. It operates by iteratively adjusting the model's parameters based on the computed gradients of the loss function with respect to those parameters.

2. Adam Optimizer:
The Adam optimizer, an acronym for Adaptive Moment Estimation, represents a paradigm shift in the landscape of machine learning optimization algorithms. Introduced by Kingma and Ba in 2014, Adam seamlessly blends the advantages of both momentum-based optimization and adaptive learning rates, offering a robust and efficient approach to model parameter optimization.

At its core, Adam incorporates the concept of momentum by maintaining a moving average of the first-order moments, or gradients. This allows the optimization process to retain information about the historical gradients, aiding in the convergence of the model. In addition, Adam introduces the concept of adaptive learning rates by considering the second-order moments, or the uncentered variance, of the gradients. This adaptive mechanism enables the algorithm to dynamically adjust the learning rates for individual parameters, providing a powerful tool to navigate the complexities of high-dimensional and sparse gradient landscapes.

- **Dropout: Form of Regularization**

  Regularization:

  Regularization is a set of techniques used in machine learning to prevent a model from overfitting the training data. Overfitting occurs when a model learns not only the underlying patterns in the data but also the noise and outliers, leading to poor generalization on unseen data.

  Purpose of Regularization:

  The primary purpose of regularization is to strike a balance between fitting the training data well and preventing the model from becoming too complex. Regularization techniques discourage overly complex models by adding a penalty term to the objective function, which discourages extreme parameter values.

  Dropout:

  Dropout is a regularization technique where, during training, randomly selected neurons are ignored or "dropped out" with a probability p. This means that their contribution to the forward pass and backward pass is temporarily removed. Dropout is only applied during training, and all neurons are active during inference.

  Purpose of Dropout:

  The primary purpose of dropout is to prevent overfitting by reducing the reliance of neurons on specific features. By dropping out random neurons during each training iteration, the network becomes more robust and less likely to memorize the training data, leading to better generalization to unseen data.

Chapter 4: Methodology
Building a Handwritten Digit Recognizer

4.1 Importing key libraries, and reading data

```
import pandas as pd
import numpy as np

np.random.seed(1212)

import keras
from keras.models import Model
from keras.layers import *
from keras import optimizers

df_train = pd.read_csv('../input/train.csv')
df_test = pd.read_csv('../input/test.csv')

df_train.head() # 784 features, 1 label
```

Fig 4.1

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel7
0	1	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	
2	1	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	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 785 columns

Table 4.1

4.2 Splitting into training and validation dataset

```
df_features = df_train.iloc[:, 1:785]
df_label = df_train.iloc[:, 0]

X_test = df_test.iloc[:, 0:784]

print(X_test.shape)
```

Fig 4.2

```
from sklearn.model_selection import train_test_split
X_train, X_cv, y_train, y_cv = train_test_split(df_features, df_label,
                                                test_size = 0.2,
                                                random_state = 1212)

X_train = X_train.values.reshape(33600, 784) #(33600, 784)
X_cv = X_cv.values.reshape(8400, 784) #(8400, 784)

X_test = X_test.values.reshape(28000, 784)
```

Fig 4.3

4.3 Data cleaning, normalization and selection

```
# Feature Normalization
X_train = X_train.astype('float32'); X_cv= X_cv.astype('float32'); X_test = X_te
st.astype('float32')
X_train /= 255; X_cv /= 255; X_test /= 255

# Convert labels to One Hot Encoded
num_digits = 10
y_train = keras.utils.to_categorical(y_train, num_digits)
y_cv = keras.utils.to_categorical(y_cv, num_digits)
```

Fig 4.4

4.4 Model Fitting

We proceed by fitting several simple neural network models using Keras (with TensorFlow as our backend) and collect their accuracy. The model that performs the best on the validation set will be used as the model of choice for the competition.

Model: Simple Neural Network with 4 layers (300, 100, 100, 200)

Using a 4 layer neural network with:

1. 20 training epochs
2. A training batch size of 100
3. Hidden layers set as (300, 100, 100, 200)
4. Learning rate of 0.1

```
Inp = Input(shape=(784,))
x = Dense(n_hidden_1, activation='relu', name = "Hidden_Layer_1")(Inp)
x = Dense(n_hidden_2, activation='relu', name = "Hidden_Layer_2")(x)
x = Dense(n_hidden_3, activation='relu', name = "Hidden_Layer_3")(x)
x = Dense(n_hidden_4, activation='relu', name = "Hidden_Layer_4")(x)
output = Dense(num_digits, activation='softmax', name = "Output_Layer")(x)
```

Fig 4.5

```
# Insert Hyperparameters
learning_rate = 0.1
training_epochs = 20
batch_size = 100
```

Fig 4.6

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 Hidden_Layer_1 (Dense)      (None, 300)               235500

 Hidden_Layer_2 (Dense)      (None, 100)               30100

 Hidden_Layer_3 (Dense)      (None, 100)               10100

 Hidden_Layer_4 (Dense)      (None, 200)               20200

 Output_Layer (Dense)        (None, 10)                2010

=================================================================
Total params: 297910 (1.14 MB)
Trainable params: 297910 (1.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Table 4.2

- **Testing Plain Vanilla Stochatic Gradient Descent Optimizer**

```
# We rely on the plain vanilla Stochastic Gradient Descent as our optimizing methodology
sgd = optimizers.SGD(learning_rate=learning_rate)
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Fig 4.7

```
history1 = model.fit(X_train, y_train,
                     batch_size = batch_size,
                     epochs = training_epochs,
                     verbose = 2,
                     validation_data=(X_cv, y_cv))
```

Fig 4.8

- **Testing ADAM Optimzer**

```
# We rely on ADAM as our optimizing methodology
adam = keras.optimizers.Adam(learning_rate=learning_rate)
model2 = Model(Inp, output)

model2.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])
```

Fig 4.9

```
history2 = model2.fit(X_train, y_train,
                      batch_size = batch_size,
                      epochs = training_epochs,
                      verbose = 2,
                      validation_data=(X_cv, y_cv))
```

Fig 4.10

It was observed, the model which relies on 'Adam' as its optimizer tend to perform 1.5 - 2.5% better on average. Going forward, we will use 'Adam' as our optimizer of choice.

4.5 Including dropout (dropout rate of 0.3) in our second model to prevent overfitting.

```
Inp = Input(shape=(784,))
x = Dense(n_hidden_1, activation='relu', name = "Hidden_Layer_1")(Inp)
x = Dropout(0.3)(x)
x = Dense(n_hidden_2, activation='relu', name = "Hidden_Layer_2")(x)
x = Dropout(0.3)(x)
x = Dense(n_hidden_3, activation='relu', name = "Hidden_Layer_3")(x)
x = Dropout(0.3)(x)
x = Dense(n_hidden_4, activation='relu', name = "Hidden_Layer_4")(x)
output = Dense(num_digits, activation='softmax', name = "Output_Layer")(x)
```

Fig 4.11

```
model4.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])
```

Fig 4.12

4.6 Using model to predict for the test set

```
test_pred = pd.DataFrame(model4.predict(X_test, batch_size=200))
test_pred = pd.DataFrame(test_pred.idxmax(axis = 1))
test_pred.index.name = 'ImageId'
test_pred = test_pred.rename(columns = {0: 'Label'}).reset_index()
test_pred['ImageId'] = test_pred['ImageId'] + 1

test_pred.head()
```

Fig 4.13

```
test_pred.to_csv('model_result.csv', index = False)
```

Fig 4.14

Chapter 5: Result and Discussion

5.1 Results from different Optimizers

- **Plain Vanilla Stochastic Gradient Descent Optimizer**

  While using a 4 layer neural network with:

  1. 20 training epochs
  2. A training batch size of 100
  3. Hidden layers set as (300, 100, 100, 200)
  4. Learning rate of 0.1

  Achieved a training score of around 96-98% and a test score of around 95 - 97%.

- **ADAM Optimizer**

  While using a 4 layer neural network with:

  1. 20 training epochs
  2. A training batch size of 100
  3. Hidden layers set as (300, 100, 100, 200)
  4. Learning rate of 0.1

  Achieved a training score of around 99-99.7% and a test score of around 97 - 98%.

Optimizer	Epoch Limit	Loss	Accuracy	Val_Loss	Val_Accuracy
Plain Vanilla SGD	20/20	0.117	0.9663	0.1455	0.9587
ADAM	20/20	0.0096	0.9968	0.1175	0.9776

Table 5.1

5.2 Accuracy from different learning rates

Learning Rate	Epoch Limit	Loss	Accuracy	Val_Loss	Val_Accuracy
0.01	20/20	0.0117	0.9966	0.119	0.9776
0.1	20/20	0.0096	0.9968	0.1175	0.9776
0.5	20/20	0.0091	0.9971	0.1241	0.9752

Table 5.2

The accuracy, as measured by the 3 different learning rates 0.01, 0.1 and 0.5 are around 98%, 97%, and 98% respectively. As there are no considerable gains from changing the learning rates, we stick with the default learning rate of 0.01.

5.3 Effect of Epoch Limit

Epoch Limit	Loss	Accuracy	Val_Loss	Val_Accuracy
5/5.	0.0453	0.9862	0.0943	0.9746
20/20	0.0096	0.9968	0.1175	0.9776
100/100	0.0591	0.9771	0.1041	0.9732

Table 5.3

It was observed that on reducing the Epoch Limit down to 5, there was a downfall in the accuracy of the model. While on increasing it to 100, the accuracy was still negatively affected.

To avoid both Underfitting and Overfitting, a value of 20 was selected as the Epoch Limit, which gave very nearly accurate classifications.

5.4 Summarizing Discussion

1. It is found that a 4-layered neural network, using 'Adam' as the optimizer along with a learning rate of 0.01, performs best.

2. A dropout, which is a technique of regularization, was introduced in the model and used the model to classify the test set.

3. The Epoch Limit was set to 20 to reach considerable accuracy.

4. The test predictions generated by the model are predicted with an accuracy score of 97.600%.

5. The table below showcases a part of the final labels generated by the model.

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	0
4	5	3
5	6	7
6	7	0
7	8	3
8	9	0
9	10	3
10	11	5
11	12	7
12	13	4
13	14	0

Table 5.4

Chapter 6: Conclusion, Limitation and Future Scope

6.1 Conclusion

The following are the key conclusions about the handwritten digit recognizer:

1. Accuracy and Precision:
The model can reliably distinguish between different handwritten digits with high precision and has achieved considerable levels of accuracy.

2. Versatility:
The Handwritten digit recognizer is a versatile technology applicable in diverse contexts.

3. Real-time Applications:
The model can be used in real-time scenarios, such as automatic form processing, mobile check deposit systems, and digital handwriting input devices. Its speed and accuracy make it suitable for applications requiring quick and reliable recognition.

4. Effectiveness of Adam Optimizer:
It uses the momentum term to accelerate the optimization process and the adaptive learning rates to adjust the step size for each parameter individually. This combination often results in faster convergence and better generalization.

To conclude, the development and deployment of handwritten digit recognizers represent a successful application of machine learning, particularly deep learning, in solving real-world problems related to handwritten character recognition. As technology continues to advance, we can expect further improvements in accuracy, efficiency, and adaptability for these recognition systems.

6.2 Limitations

1. Noise and Distortions:
Poor image quality, smudges, or distortions in handwritten digits can impact recognition accuracy.

2. Scaling and Orientation:
Variations in digit size and orientation may pose challenges, as the model needs to handle different scales and rotations.

3. Limited Training Data:
If the model is trained on a small dataset, it may struggle with generalization to unseen

variations in handwriting.

4. Adaptation to New Writing Styles:
Adapting to new writing styles not encountered during training can be challenging for the model.

5. Overfitting:
The model may overfit to specific characteristics of the training data, reducing its ability to generalize to diverse handwritten digits.

6.3 Future Scope

The future scope of a handwritten digit recognizer model is promising and can extend into various areas. Here are some potential directions and applications for the development and deployment of the model:

1. Multilingual Support:
Extending the model to recognize handwritten digits in multiple languages can be a valuable direction. This involves training the model on datasets that include digits from different scripts, such as Devanagari, Chinese, or Arabic numerals.

2. Real-time Recognition:
Implementing real-time recognition capabilities can open up applications in various fields, including finance (cheque processing), retail (price tag recognition), and logistics (package labeling).

3. Integration with Other Technologies:
Combining digit recognition with other technologies such as augmented reality (AR) or virtual reality (VR) can lead to innovative applications. For example, recognizing handwritten digits in real-time through a smart AR headset.

4. Medical Applications:
Handwritten digit recognition can find applications in the healthcare sector, such as reading handwritten prescriptions or patient identification.

5. Education and Learning Tools:
Developing educational tools that leverage handwritten digit recognition can aid in teaching mathematics and improve the learning experience for students.

6. Security Applications:
Handwritten digit recognition can be utilized in security systems, such as signature

verification for authentication purposes.

7. Collaboration with IoT Devices:
Integrating handwritten digit recognition with Internet of Things (IoT) devices can enable smart interactions, such as recognizing handwritten commands on smart surfaces or devices.

8. Improving Accuracy and Efficiency:
The further aim is to enhance the accuracy and efficiency of the model. This involves refining algorithms, optimizing neural network architectures, and exploring advanced training techniques.

References

1. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Geron Aurelien, 2017

2. MNIST dataset: https://www.kaggle.com/datasets/hojjatk/mnist-dataset

3. Keras: https://keras.io/

4. Keras API: https://keras.io/api/

5. Optimizers: https://www.geeksforgeeks.org/optimizers-in-tensorflow/

6. Optimizers in Keras: https://keras.io/api/optimizers/

7. ADAM Optimizer: https://keras.io/api/optimizers/adam/

8. Plain Vanilla SGD: https://www.datacamp.com/tutorial/tutorial-gradient-descent

9. TensorFlow: https://www.tensorflow.org/

10. Dropout: https://www.geeksforgeeks.org/dropout-in-neural-networks/