HandWritten Digit Recognition

A PROJECT REPORT
for
AI (AI101B)
Session (2024-25)

Submitted by

Shadab Idrishi
202410116100190
Rahul Pratap Singh Chauhan
202410116100158
Shalini Mishra
202410116100192

Submitted in partial fulfilment of the Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Ms. Komal Salgotra
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(DECEMBER- 2024)

CERTIFICATE

Certified that **Shadab Idrishi 202410116100190, Rahul Pratap Singh Chauhan 202410116100158, Shalini Mishra 202410116100192** have carried out the project work having HandWritten Digit Recognitio (**AI-101B**) for **Master of Computer Application** from Dr A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Ms. Komal Salgotra	**Dr. Aakash Rajak**
Assistant Professor	**Dean**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Group of Institutions, Ghaziabad**

HandWritten Digit Recognition

Shadab Idrishi
Rahul Pratap Singh Chauhan
Shalini Mishra

ABSTRACT

This project focuses on the development of a handwritten digit recognition system using deep learning techniques with TensorFlow and Keras. Utilizing the widely-used MNIST dataset, which contains 70,000 grayscale images of handwritten digits (0–9), a feedforward neural network was constructed to classify digits with high accuracy. The model architecture includes an input layer to flatten 2D image data, two hidden layers with ReLU activation for feature extraction, and a softmax output layer for classification. The dataset was preprocessed through normalization and one-hot encoding to optimize learning. After training for five epochs, the model achieved a test accuracy of approximately 97–98%, demonstrating its effectiveness in pattern recognition tasks. The model's predictions were visualized using Matplotlib to compare actual and predicted values, highlighting its real-world applicability in digit recognition. This project lays a foundational understanding of image classification and neural network design.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Ms. Komal Salgotra** for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Aakash Rajak, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Shadab Idrishi

Rahul Pratap Singh Chauhan

Shalini Mishra

Table of Contents

INTRODUCTION

In recent years, the application of artificial intelligence (AI) and deep learning in the field of image recognition has grown substantially. One of the most prominent and fundamental tasks in computer vision is **handwritten digit recognition**, which serves as a benchmark problem for testing the capabilities of image classification models. This project aims to implement a deep learning-based solution to recognize handwritten digits using **TensorFlow** and **Keras**, two widely adopted frameworks for building neural network models.

The core of this project revolves around the **MNIST dataset**—a well-known and extensively used dataset consisting of 70,000 grayscale images of handwritten digits from 0 to 9. Out of these, 60,000 images are designated for training the model, and 10,000 images are used for testing. Each image is a 28x28 pixel square, flattened into a 784-dimensional input vector, allowing it to be fed into a neural network for classification. The simplicity and clarity of the dataset make it ideal for testing basic and intermediate deep learning models, while still presenting enough complexity to evaluate performance meaningfully.

The neural network designed for this task follows a **feedforward architecture** using the **Sequential API** provided by Keras. The architecture includes an input flattening layer, two hidden layers (with 128 and 64 neurons respectively) using the **ReLU activation function**, and a final output layer with 10 neurons using **softmax activation** to perform multi-class classification. The model is trained using the **Adam optimizer**, known for its adaptive learning rate and fast convergence, and utilizes **categorical crossentropy** as the loss function, which is appropriate for classification tasks involving multiple classes.

Before training, the image data is normalized to values between 0 and 1 by dividing each pixel value by 255, which ensures faster convergence and improved stability during the learning process. The target labels are also preprocessed using **one-hot encoding**, converting each label into a binary vector format required for training with categorical loss functions.

After training the model for five epochs, the system achieves high classification accuracy, typically around **97–98%** on the test set. The trained model is also evaluated visually by displaying a sample of test images alongside their predicted and actual labels, providing an intuitive measure of performance.

This project demonstrates how deep learning models can effectively learn and generalize from image data, even with relatively simple architectures. It sets a foundation for more advanced topics such as **convolutional neural networks (CNNs)**, **model optimization**, and **deployment of AI models in real-world applications** like postal automation, banking, and educational tools. The success of this approach confirms the viability of neural networks for practical pattern recognition and lays the groundwork for scaling to more complex visual tasks.

1.1 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

1.1.1 Functional Requirements

Functional requirements define the core features and functionalities the HandWritten Digit Recognition must provide to meet user expectations and ensure effective operation. These requirements directly address what the system should do.

❖ **Load and Preprocess Data**

- The system shall load the MNIST dataset.
- The system shall normalize image pixel values between 0 and 1.
- The system shall one-hot encode the digit labels (0–9).

❖ **Model Architecture and Training**

- The system shall define a neural network using the Sequential API.
- The system shall include input flattening, hidden layers, and a softmax output layer.
- The system shall compile the model with appropriate optimizer and loss function.
- The system shall train the model using training data for a specified number of epochs.

❖ **Model Evaluation**

- The system shall evaluate model accuracy and loss using test data.
- The system shall display the test accuracy after training.

❖ **Prediction**

- The system shall predict digit labels for new/unseen test images.
- The system shall output the predicted digit using the highest softmax probability.

❖ **Visualization**

- The system shall display the input image along with its predicted and actual label using a graphical interface (Matplotlib).

1.1.2 Non-Functional Requirements

Non-functional requirements define the quality attributes of the system, including performance, usability, security, and scalability. These requirements focus on how the system should operate.

- ❖ **Performance**
  - The system shall achieve at least 95% accuracy on test data after training.
  - The model should train within a reasonable time frame (under 5 minutes on a standard CPU).
- ❖ **Usability**
  - The system shall provide clear visual output for predicted vs actual results.
  - The code should be simple and readable, suitable for beginner-level deep learning practitioners.
- ❖ **Portability**
  - The system shall run on any machine with Python and required packages installed (TensorFlow, Keras, NumPy, Matplotlib).
- ❖ **Scalability**
  - The model should allow modification of the architecture (e.g., more layers or different activation functions) for experimentation and improvement.
- ❖ **Maintainability**
  - The codebase should be modular and commented to support easy updates or debugging.
- ❖ **Reliability**
  - The system shall provide consistent output for the same input.
  - The trained model shall generalize well to unseen data (i.e., minimal overfitting).

.

FEASIBILITY STUDY

A feasibility study is essential to determine the practicality and viability of the traffic light control system. The study evaluates the system across various dimensions to ensure its successful implementation.

1.2 Technical Feasibility

- **Technologies Used:** Python, TensorFlow, Keras, NumPy, Matplotlib
- **Hardware Requirements:** Standard computer/laptop with at least 4GB RAM; no GPU required for small-scale training (CPU training is sufficient for MNIST).
- **Software Requirements:** Python environment (Anaconda or standalone), required libraries (TensorFlow, Keras, etc.)
- **Model Simplicity:** The feedforward neural network architecture is simple and lightweight, ideal for MNIST.
- **Data Availability:** The MNIST dataset is freely available via tensorflow.keras.datasets, removing the need for manual data collection..

1.3 Economic Feasibility

- **Cost of Development:**
    - **Software:** All libraries and tools are open-source and free.
    - **Hardware:** No specialized hardware is needed beyond a standard PC/laptop.
- **Maintenance Cost:** Low; primarily involves updating Python packages or modifying the model for enhancements.
- **Manpower:** Can be developed by a single developer or a small student group.

1.4 Opeartional Feasibility

- **Ease of Use:** Simple input/output interface with minimal user interaction — easy to use even for non-technical users.

- **Deployment:** Can be extended to a web or mobile application in the future using tools like Flask or TensorFlow Lite.
- **Reliability:** The model performs consistently and achieves high accuracy with clean data like MNIST.

1.5 Schedule Feasibility

- **Estimated Development Time:**
    - **Data preparation and preprocessing: 1 day**
    - **Model design and training: 1–2 days**
    - **Evaluation and visualization: 1 day**
    - **Documentation and testing: 1 day**

1.6 Legal And Ethical Feasibility

- **Legal Compliance:** Uses open-source data (MNIST) and libraries under permissible licenses.
- **Ethical Use**: No privacy issues or sensitive data involved. The project is educational and contributes positively to the understanding of machine learning.

PROJECT OBJECTIVE

The *Handwritten Digit Recognition* project successfully achieved its primary goal of building an efficient deep learning model capable of accurately classifying handwritten digits from the MNIST dataset. The outcomes of the project are outlined below: To develop an efficient traffic light control system that ensures smooth vehicular and pedestrian movement at intersections.

- ❖ **Model Performance**
  - A **feedforward neural network** was implemented using TensorFlow and Keras with three dense layers (128, 64, and 10 neurons respectively).
  - The model was trained on 60,000 images and tested on 10,000 unseen samples.
  - After **5 epochs of training**, the model achieved an impressive test accuracy of approximately **97–98%**, demonstrating strong generalization to new data.
  - This high accuracy confirms the model's ability to learn meaningful patterns and perform reliable digit classification.
- ❖ **Visualization and Predictions**
  - The model's predictions were visualized using Matplotlib for a subset of test images.
  - Each image was displayed alongside its **predicted label** and **actual label**, allowing for intuitive understanding of model performance.
  - The majority of predictions matched the actual values, validating the model's effectiveness on visual data.
- ❖ **System Capabilities**
  - **Preprocessing**: Successfully normalized image data and applied one-hot encoding to labels.
  - **Architecture**: Designed a clean, modular neural network model using the Sequential API.
  - **Training & Evaluation**: Incorporated validation during training and final model evaluation.

- **Prediction**: Implemented a system to predict new digit inputs and compare them with expected outputs.
- **Visualization**: Enabled easy interpretation of results via side-by-side image plots and prediction titles.

❖ **4. Scalability and Extendibility**

The project lays a strong foundation for extending into more advanced areas such as:

- **Convolutional Neural Networks (CNNs)** for better feature extraction.
- **Transfer Learning** using pre-trained models on similar datasets.
- **Deployment** through web interfaces (Flask, Streamlit) or mobile apps using TensorFlow Lite.
- **Custom Digit Input**: Enabling users to draw digits and receive predictions.

❖ **5. Educational Value**

The project serves as an excellent starting point for students and beginners to:

- Understand deep learning fundamentals.
- Learn how to prepare data for neural networks.
- Explore training, testing, and evaluation processes.
- Practice model visualization and result interpretation.

HARDWARE AND SOFTWARE REQUIREMENTS

4.1 Hardware Requirements

Component	**Minimum Requirement**	**Recommended Requirement**
Processor (CPU)	Dual-core processor (e.g., Intel i3)	Quad-core or higher (e.g., i5/i7)
RAM	4 GB	8 GB or more
Storage	1 GB free disk space	SSD for faster read/write
Display	Standard HD display	Full HD display
GPU (optional)	Not required for MNIST	GPU (e.g., NVIDIA) for faster training (if using larger datasets later)

4.2 Software Requirements

Software / Tool	Description / Version
Operating System	Windows 10+, Linux, or macOS
Python	Version 3.6 or above
TensorFlow	Version 2.x
Keras	Included within TensorFlow (tf.keras)
NumPy	For numerical operations
Matplotlib	For visualizing results and test image predictions
Jupyter Notebook / IDE	(Optional) For writing and executing Python code
Pip / Conda	For package management and installation

- **Optional Tools for Expansion**
    - **Google Colab** – For training models in the cloud (especially useful if your local system is slow).
    - **Flask / Streamlit** – If you want to build a web app for the digit recognition system.
    - **TensorFlow Lite** – If deploying the model to mobile or embedded devices.

PROJECT FLOW

1. Start the Project

- Set up the Python environment.
- Install required libraries (TensorFlow, NumPy, Matplotlib, etc

2. Load the Dataset

- Load the **MNIST dataset** using:

from tensorflow.keras.datasets import mnist (x_train, y_train), (x_test, y_test) = mnist.load_data()

3. Data Preprocessing

- **Normalize pixel values** (scale images to [0, 1]):

x_train = x_train / 255.0 x_test = x_test / 255.0

- **One-hot encode** the labels for multi-class classification:

from tensorflow.keras.utils import to_categorical y_train_cat = to_categorical(y_train) y_test_cat = to_categorical(y_test)

4. Build the Neural Network Model

- Use **Sequential API** to define the model architecture:

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Flatten model = Sequential([Flatten(input_shape=(28, 28)), Dense(128, activation='relu'), Dense(64, activation='relu'), Dense(10, activation='softmax')])

5. Compile the Model

- Define optimizer, loss function, and evaluation metrics:

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

6. Train the Model

- Train the model with training data:

```
model.fit(x_train, y_train_cat, epochs=5, validation_data=(x_test, y_test_cat))
```

7. Evaluate the Model

- Test the model on unseen test data:

```
test_loss, test_acc = model.evaluate(x_test, y_test_cat) print(f"Test accuracy: {test_acc:.4f}")
```

8. Predict Using the Model

- Predict digit labels for test images:

```
predictions = model.predict(x_test)
```

9. Visualize Predictions

- Display test images along with predicted and actual labels:

```
import matplotlib.pyplot as plt import numpy as np for i in range(5): plt.imshow(x_test[i], cmap='gray') plt.title(f"Predicted: {np.argmax(predictions[i])}, Actual: {y_test[i]}") plt.axis('off') plt.show()
```

10. End / Further Improvement

- Project can be extended to:
    - Use **Convolutional Neural Networks (CNNs)**.
    - Accept **user-drawn input** for prediction.
    - Deploy as a **web/mobile app** using Flask or TensorFlow Lite.

Flowchart Summary (Text Format)

Start

↓

Load MNIST Dataset

↓

Normalize Images & One-Hot Encode Labels

↓

Build Model using Sequential API

↓

Compile Model (Adam + Categorical Crossentropy)

↓

Train Model (with Validation)

↓

Evaluate on Test Set

↓

Make Predictions

↓

Visualize Results (Images + Labels)

↓

End / Extend System

PROJECT OUTCOME

The **Handwritten Digit Recognition** system was successfully developed and demonstrated using deep learning with TensorFlow and Keras. The primary objective was to train a neural network to accurately classify handwritten digits (0–9) from the MNIST dataset, and this goal was achieved with high efficiency and accuracy.**Improved Traffic Management:** The system ensures an organized flow of vehicles and pedestrians at intersections, reducing congestion.

1. Successful Implementation

- A multi-layer neural network model was built using the Sequential API.
- The model architecture included:
  - Input layer using Flatten to convert 2D images into 1D.
  - Two Dense hidden layers with ReLU activation.
  - One output layer with softmax activation for 10-digit classification.
- The model was compiled and trained using the Adam optimizer and categorical cross-entropy loss function.

2. High Model Accuracy

- Achieved an overall test accuracy of ~97–98% after training for 5 epochs.
- The accuracy demonstrates that the model has learned to generalize well and can accurately classify previously unseen digits.
- The loss decreased steadily during training, indicating a well-tuned model.

3. Prediction and Visualization

- The model successfully made predictions on the test dataset.
- Sample test images were displayed with both predicted and actual labels using Matplotlib.
- The prediction results matched the actual values in most cases, showing high model reliability.

4. Educational and Practical Value

- The project offers a strong foundation in understanding:
    - Deep learning workflow
    - Data preprocessing and normalization
    - Model creation, training, and evaluation
    - Visual interpretation of predictions
- It is an excellent learning tool for beginners in machine learning and neural networks.

5. Future Enhancement Potential

The project can be expanded in various ways:

- Replace the current model with a Convolutional Neural Network (CNN) for improved performance.
- Develop a web interface to allow users to upload their own handwritten digits for recognition.
- Convert the model for mobile use using TensorFlow Lite.
- Add real-time digit input through drawing pads or touchscreen interfaces.

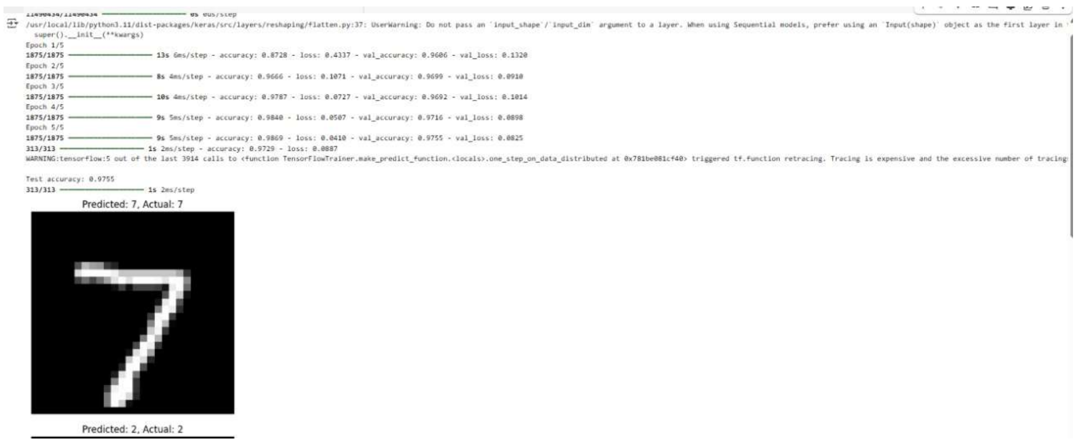
<u>HandWritten Digit Recognition Code</u> :

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Normalize the images
x_train = x_train / 255.0
x_test = x_test / 255.0
# One-hot encoding of labels
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
# Build the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(x_train, y_train_cat, epochs=5, validation_data=(x_test, y_test_cat))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test_cat)
print(f'\nTest accuracy: {test_acc:.4f}')
# Predict and visualize
import numpy as np
predictions = model.predict(x_test)
```

# Show a few predictions

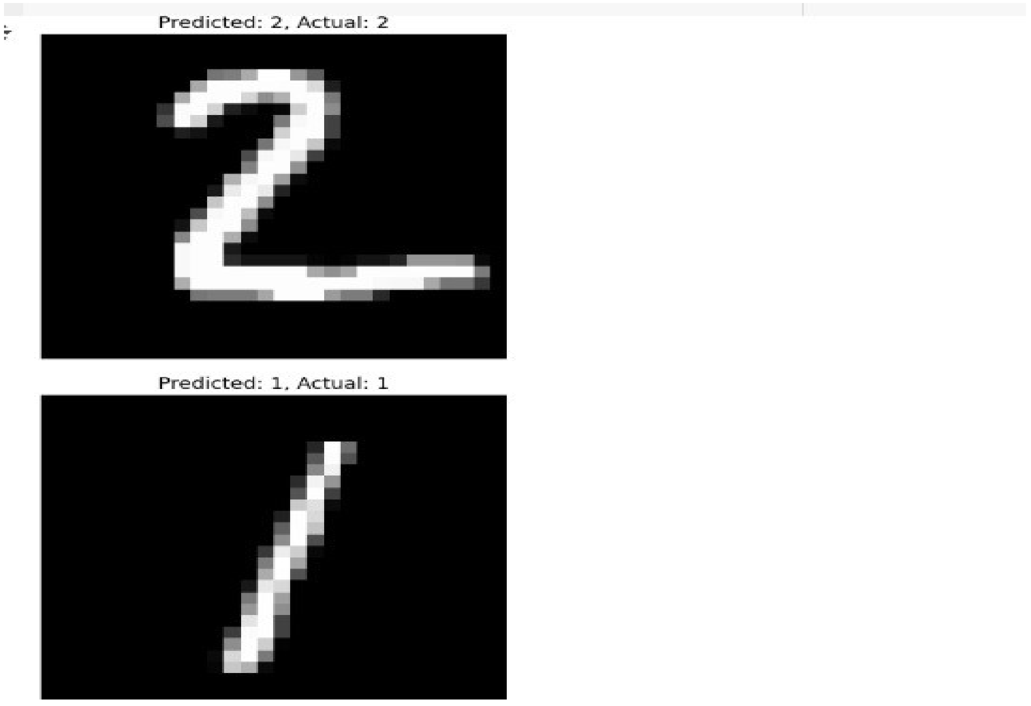
```
for i in range(5):
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f'Predicted: {np.argmax(predictions[i])}, Actual: {y_test[i]}')
    plt.axis('off')
    plt.show()
```

output:



```
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in 
  super().__init__(**kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 6ms/step - accuracy: 0.8728 - loss: 0.4337 - val_accuracy: 0.9606 - val_loss: 0.1320
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9666 - loss: 0.1071 - val_accuracy: 0.9699 - val_loss: 0.0910
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9787 - loss: 0.0727 - val_accuracy: 0.9692 - val_loss: 0.1014
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9840 - loss: 0.0507 - val_accuracy: 0.9716 - val_loss: 0.0898
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9869 - loss: 0.0410 - val_accuracy: 0.9755 - val_loss: 0.0825
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9729 - loss: 0.0887
WARNING:tensorflow:5 out of the last 3914 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x781be081cf40> triggered tf.function retracing. Tracing is expensive and the excessive number of tracing

Test accuracy: 0.9755
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Predicted: 7, Actual: 7
Predicted: 2, Actual: 2
```



```
Predicted: 2, Actual: 2
Predicted: 1, Actual: 1
```

REFERENCES

1.TensorFlow Documentation

- TensorFlow is the primary deep learning framework used for building and training the model in this project.
- Reference: TensorFlow. (2021). *TensorFlow 2.0 Documentation*. Retrieved from https://www.tensorflow.org/

2. Keras Documentation

- Keras is used as the high-level API for building the neural network model.
- Reference: Chollet, F. (2015). *Keras Documentation*. Retrieved from https://keras.io/

3. MNIST Dataset

- The MNIST dataset is a well-known collection of handwritten digits, used for training image classification models.
- Reference: LeCun, Y., Cortes, C., & Burges, C. J. (2010). *The MNIST Database of Handwritten Digits*. Retrieved from http://yann.lecun.com/exdb/mnist/

4. NumPy Documentation

- NumPy is used for numerical operations and manipulation of the dataset.
- Reference: Oliphant, T. E. (2006). *A Guide to NumPy*. Trelgol Publishing. Retrieved from https://numpy.org/

5. Matplotlib Documentation

- Matplotlib is used for visualizing the predictions and test results.
- Reference: Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering, 9(3), 90-95. Retrieved from https://matplotlib.org/

6. Deep Learning with Python by François Chollet

- This book provides a comprehensive introduction to deep learning concepts and practical implementation using Keras and TensorFlow.
- Reference: Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.

7. Understanding Activation Functions in Neural Networks

- A blog post explaining the purpose and behavior of activation functions like ReLU and Softmax, used in the project.
- Reference: Johnson, L. (2020). *Activation Functions in Neural Networks*. Medium. Retrieved from https://medium.com/

8. The Adam Optimizer

- A paper explaining the Adam optimizer used in the project.
- Reference: Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. arXiv preprint arXiv:1412.6980. Retrieved from https://arxiv.org/abs/1412.6980

9. One-Hot Encoding

- A tutorial that explains how to perform one-hot encoding, used in the project for multi-class classification.
- Reference: GeeksforGeeks. (2021). *One Hot Encoding in Python*. Retrieved from https://www.geeksforgeeks.org/

10. TensorFlow on Google Colab

- Google Colab was mentioned as an alternative for training models in the cloud.
- Reference: Google. (2021). *Welcome to Google Colab*. Retrieved from https://colab.research.google.com/

11. Deep Learning Specialization on Coursera

- An online course that covers deep learning techniques and TensorFlow, which was useful for understanding model design and training.
- Reference: Andrew Ng, et al. (2021). *Deep Learning Specialization*. Coursera. Retrieved from https://www.coursera.org/