HANDWRITTEN DIGIT RECOGNITION USING NEURAL NETWORK(ANN)

by

Pritiranjan Biswal (STB03005)

Submitted to Scifor Technologies

Meta
Scifor Technologies

Script. Sculpt. Socialize

UNDER GUIDIANCE OF

Urooj Khan

TABLE OF CONTENTS

ABSTRACT

This thesis explores the development and training of a neural network model for handwritten digit recognition using the MNIST dataset. The process involves importing necessary libraries, loading and preprocessing the data, constructing a neural network architecture, training the model, and evaluating its performance. The MNIST dataset, consisting of 60,000 training images and 10,000 testing images of handwritten digits, serves as the basis for this study. Through the utilization of TensorFlow and Keras frameworks, a Sequential model architecture is built, comprising layers such as Flatten and Dense with appropriate activation functions. The model is trained using the Adam optimizer and sparse categorical cross-entropy loss function. Throughout the training process, performance metrics including accuracy and loss are monitored and visualized using matplotlib. The trained model achieves a high accuracy on the test dataset, demonstrating its effectiveness in accurately classifying handwritten digits. This research contributes to the understanding and application of deep learning techniques in the field of image classification, with implications for various real-world applications such as optical character recognition and digitized document processing.

INTRODUCTION

In the realm of artificial intelligence and machine learning, the field of computer vision has seen remarkable advancements, particularly in the domain of image classification. One of the pivotal datasets contributing to this progress is the MNIST dataset, a collection of handwritten digits widely used as a benchmark in the development of machine learning algorithms.

This thesis delves into the exploration and implementation of a deep learning model for the classification of handwritten digits using the MNIST dataset. The primary objective is to develop a robust neural network architecture capable of accurately identifying digits ranging from zero to nine. Through the utilization of TensorFlow and Keras, prominent frameworks in the deep learning community, this research endeavors to construct a model that not only achieves high accuracy but also demonstrates a comprehensive understanding of key concepts in deep learning.

The methodology begins with preprocessing the dataset, including data normalization to facilitate efficient training. Subsequently, a sequential neural network architecture is constructed, comprising layers of densely connected neurons. The model architecture incorporates features such as flattening layers to transform the input data from a two-dimensional array to a one-dimensional vector, as well as employing activation functions such as ReLU (Rectified Linear Unit) and softmax to introduce non-linearity and facilitate multi-class classification.

Training the model involves optimizing its parameters using the Adam optimizer and evaluating its performance using the sparse categorical cross-entropy loss function. The training process is monitored through multiple epochs, with validation data utilized to assess the model's generalization capabilities and prevent overfitting.

Furthermore, this thesis elucidates the evaluation metrics employed to quantify the model's performance, including accuracy scores and loss metrics. Additionally, visualizations such as loss curves and accuracy plots provide insights into the training process and model convergence.

Ultimately, the culmination of this research aims to contribute to the broader discourse surrounding deep learning methodologies for image classification tasks, with a specific focus on handwritten digit recognition. By elucidating the intricacies of model development, training, and evaluation, this thesis seeks to empower practitioners and researchers in the field of computer vision to leverage advanced techniques in pursuit of innovative solutions to real-world challenges.

TECHNOLOGY USED

The technology stack used in this projects can be summarized as follows:

1. Programming Language: Python
2. Deep Learning Framework: TensorFlow
3. Libraries:
    - tensorflow: Deep learning library for building neural networks.
    - matplotlib: Data visualization library for plotting graphs and images.
    - streamlit: Python library for building interactive web applications.
    - numpy: Library for numerical computations.
    - scikit-learn: Machine learning library for various algorithms and metrics.
4. Data Handling and Manipulation:
    - Loading and preprocessing the MNIST dataset.
    - Manipulating arrays and images using NumPy and TensorFlow libraries.
5. Model Building:
    - Utilization of Keras, a high-level neural networks API running on top of TensorFlow, for building and training neural network models.
    - Sequential model architecture with densely connected layers.
    - Activation functions such as ReLU and Softmax.
    - Compilation of the model with specified loss function, optimizer, and evaluation metrics.
6. Training and Evaluation:
    - Training the model using the `fit` method.
    - Evaluation of the model using accuracy metrics.
7. Web Application Development:
    - Development of an interactive web application for handwritten digit recognition using Streamlit.
    - Utilization of Streamlit's file uploader and image display functionalities.
    - Integration of the pre-trained neural network model for prediction within the web application.

8. Model Deployment:

- Saving and loading the trained model using Keras's `load_model` function.
- Pre-processing of uploaded images for prediction using Keras's image pre-processing utilities.

These technologies collectively enable the creation of a complete pipeline for training, evaluating, and deploying a handwritten digit recognition system as an interactive web application.

DATASET INFORMATION

The MNIST dataset consists of 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image is represented as a 28x28 grayscale matrix. The dataset is commonly used for benchmarking machine learning algorithms in handwritten digit recognition tasks.

METHODOLOGY

The methodology employed in this project involves several steps:

1. Importing Libraries:

- The code begins by importing necessary libraries like TensorFlow, Keras, Matplotlib, and Streamlit for building and deploying the model, loading and visualizing data, and creating the web application.

2. Loading and Preprocessing Data:

- The MNIST dataset, a collection of 28x28 grayscale images of handwritten digits along with their labels, is loaded using Keras.

- Data preprocessing involves normalizing pixel values to a range of 0 to 1, which helps in training the model efficiently.

3. Model Architecture:

- The neural network model is defined using Keras Sequential API.

- The input layer flattens the 28x28 image into a 1D array of 784 elements.

- Dense layers with ReLU activation functions are added to introduce non-linearity and learn complex patterns.

- The output layer consists of 10 nodes, each representing a digit from 0 to 9, and utilizes softmax activation for multiclass classification.

4. Model Compilation:

- The model is compiled with appropriate loss function ('sparse_categorical_crossentropy'), optimizer ('Adam'), and evaluation metric ('accuracy') using the `compile()` method.

5. Model Training:

- The compiled model is trained on the training data using the `fit()` method.

- The training process involves iterating through epochs (25 in this case), with a validation split of 20% to monitor the model's performance on unseen data.

6. Model Evaluation:

- After training, the model's performance is evaluated on the test set using accuracy as the metric.

- The accuracy score is computed using `accuracy_score` from scikit-learn library.

7. Visualization:

- Matplotlib is used to visualize training and validation loss, as well as training and validation accuracy, over epochs.

- Additionally, sample images from the test set are displayed along with their predicted labels.

8. Model Saving:

- The trained model is saved in the Hierarchical Data Format (HDF5) using Keras's `save()` method.

9.Streamlit Setup:

import streamlit as st: Imports the Streamlit library, which is used to create interactive web applications.

import numpy as np: Imports NumPy, which is used for numerical computations.

from tensorflow.keras.models import load_model: Imports the function load_model from Keras to load the pre-trained model.

from tensorflow.keras.preprocessing import image: Imports image preprocessing utilities from Keras.

Load Pre-trained Model:

The pre-trained neural network model for handwritten digit recognition, saved as 'mnist_classification.h5', is loaded using the load_model() function from Keras.

Image Preprocessing Function:

preprocess_image(img): This function takes an image as input, converts it to an array, expands its dimensions to match the input shape expected by the model (28x28), and normalizes pixel values to the range [0, 1].

Streamlit App Configuration:

st.set_page_config(): Configures the Streamlit app, setting the page title, icon, layout, and initial sidebar state.

Title and Description:

st.title(): Displays a title for the web application.

st.write(): Writes a description for the user, instructing them to upload an image of a handwritten digit for prediction.

File Uploader:

st.file_uploader(): Displays a file uploader widget where users can upload an image file (JPEG, JPG, or PNG) containing a handwritten digit.

Prediction:

If the user uploads an image (uploaded_file is not None), it is displayed using st.image() with the caption 'Uploaded Image.'.

The uploaded image is then processed: resized to 28x28 grayscale (to match the model input shape) and preprocessed using the preprocess_image() function.

The preprocessed image is passed to the model's predict() method to obtain a prediction, which is the digit represented by the image.

The predicted digit is displayed as a success message using st.success().

Exception Handling:

If any error occurs during image processing or prediction, it is caught, and an error message is displayed using st.error()

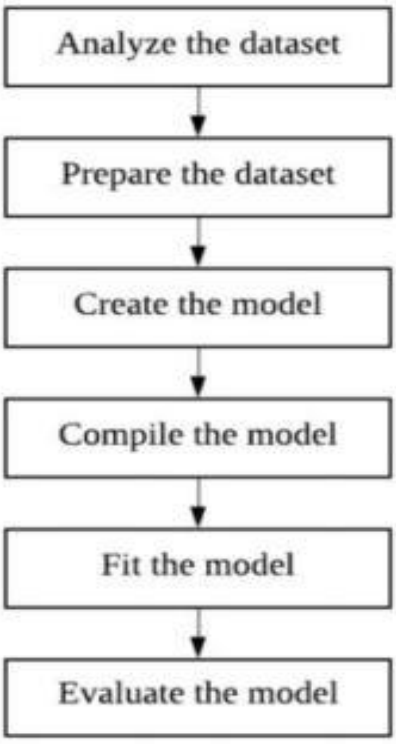


FIG-1: Workflow of the Model

CODE SNIPPET

+ Code + Text Connect Colab AI

```
#Import all libraries
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
```

```
#Load the MNIST datset
(X_train,y_train),(X_test,y_test) = keras.datasets.mnist.load_data()
```

```
X_train
```

```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
```

+ Code + Text Connect Colab AI

```
#here we get 60000 data and here the data present in 3D array
X_train.shape
```

```
(60000, 28, 28)
```

```
X_train[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
```

MNIST_dataset.ipynb
File Edit View Insert Runtime Tools Help Last edited on January 30
Comment Share
+ Code + Text Connect Colab AI

```
X_test.shape
```

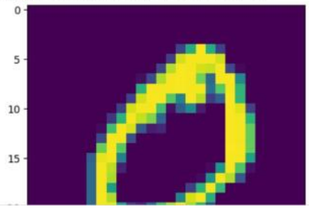
```
(10000, 28, 28)
```

```
#in y_train here it contains the label of the image what actual number it actual holded
y_train
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
import matplotlib.pyplot as plt
plt.imshow(X_train[1])
```

```
<matplotlib.image.AxesImage at 0x7fc0a070e350>
```



MNIST_dataset.ipynb

File Edit View Insert Runtime Tools Help Last edited on January 30

```
#Normalize the data from 0 to 1
X_train = X_train/255
X_test = X_test/255
```

```
x_train[0]
```

```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
```

MNIST_dataset.ipynb

File Edit View Insert Runtime Tools Help Last edited on January 30

```
#Train our model and add the layers into the model
model = Sequential()

model.add(Flatten(input_shape=(28,28)))  ##first we have to flatten the layer because we have to convert the data 2D to 1D from 28*28 to 784 as input shape
model.add(Dense(128,activation='relu'))  ##Add the first dense layer where that is 128
model.add(Dense(32,activation='relu'))
model.add(Dense(10,activation='softmax')) ##As it is a multiclass classification because we have 10 output here we are using 'softmax' because here we are getting the outpout in multi
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 32)                4128

 dense_2 (Dense)             (None, 10)                330

=================================================================
Total params: 104938 (409.91 KB)
Trainable params: 104938 (409.91 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

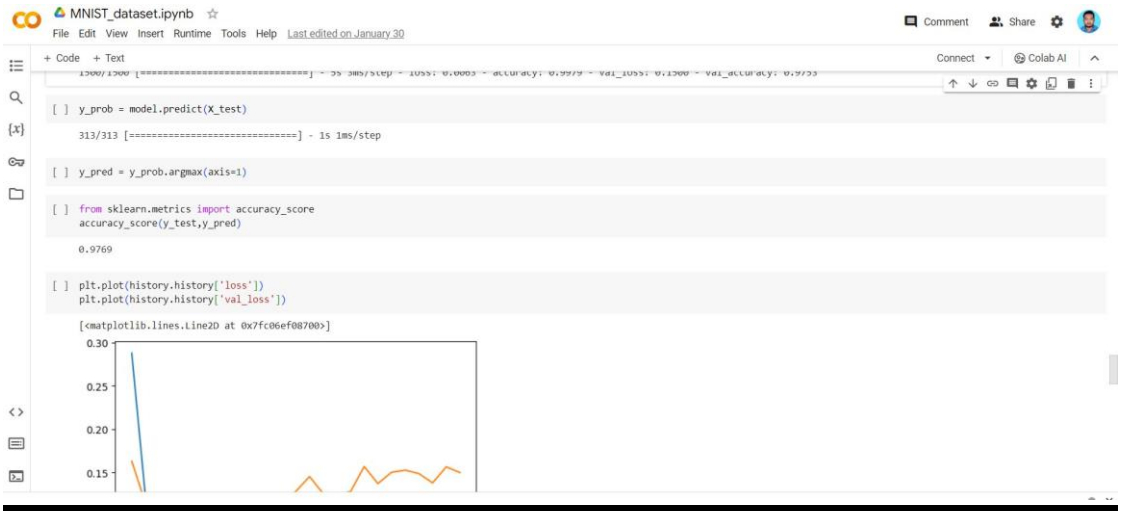
```
model.compile(loss='sparse_categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

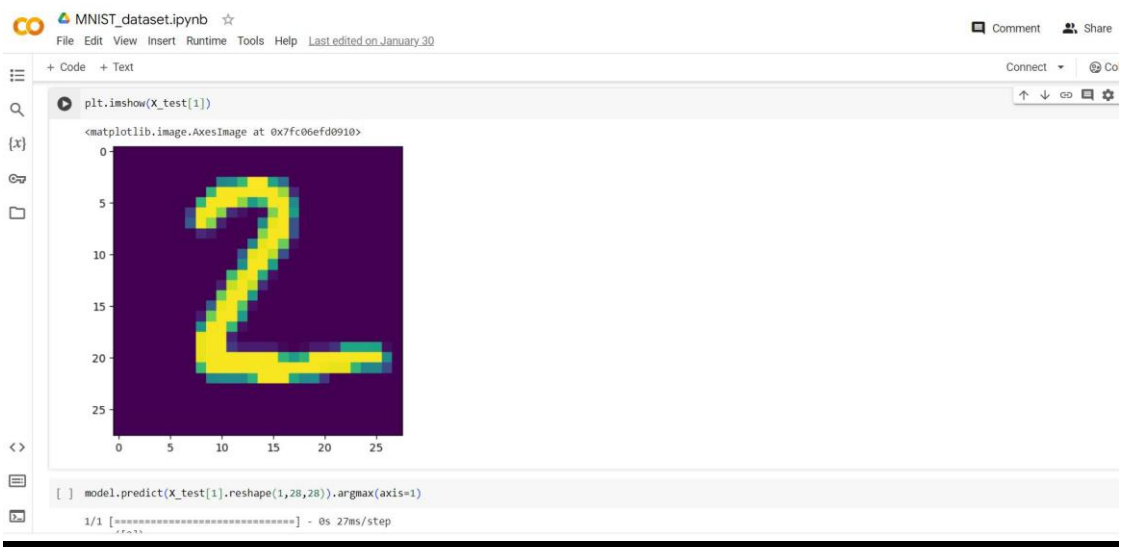
File Edit View Insert Runtime Tools Help Last edited on January 30

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

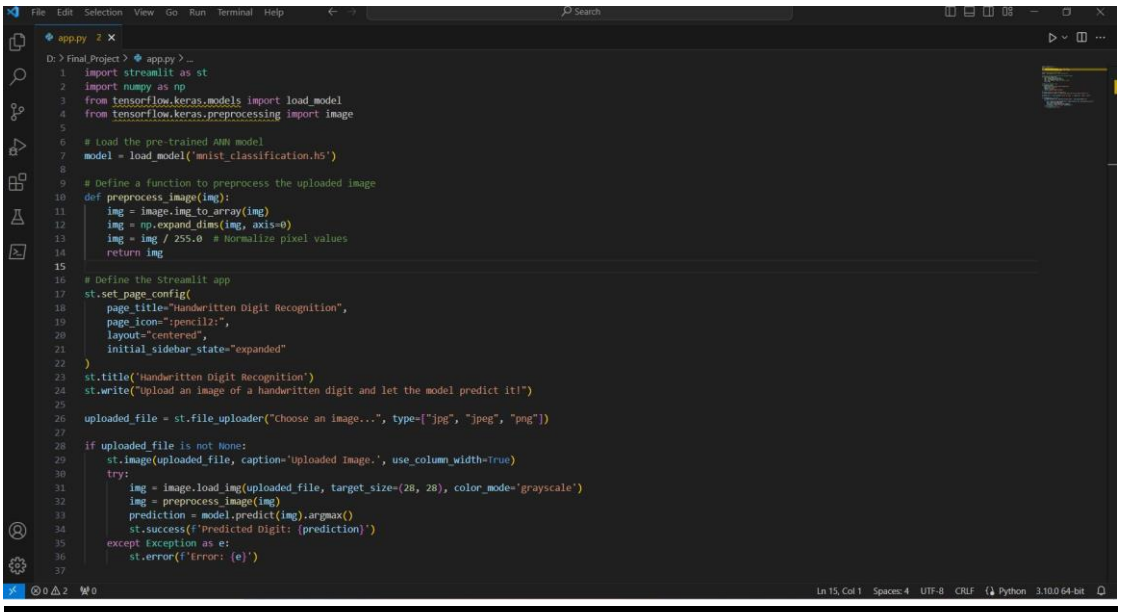
```
history = model.fit(X_train,y_train,epochs=25,validation_split=0.2)
```

```
Epoch 1/25
1500/1500 [==============================] - 7s 4ms/step - loss: 0.2878 - accuracy: 0.9176 - val_loss: 0.1632 - val_accuracy: 0.9521
Epoch 2/25
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1221 - accuracy: 0.9636 - val_loss: 0.1147 - val_accuracy: 0.9664
Epoch 3/25
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0838 - accuracy: 0.9750 - val_loss: 0.1043 - val_accuracy: 0.9682
Epoch 4/25
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0636 - accuracy: 0.9800 - val_loss: 0.0945 - val_accuracy: 0.9729
Epoch 5/25
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0490 - accuracy: 0.9842 - val_loss: 0.1053 - val_accuracy: 0.9692
Epoch 6/25
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0393 - accuracy: 0.9879 - val_loss: 0.1047 - val_accuracy: 0.9710
Epoch 7/25
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0326 - accuracy: 0.9893 - val_loss: 0.1140 - val_accuracy: 0.9707
Epoch 8/25
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0273 - accuracy: 0.9914 - val_loss: 0.0977 - val_accuracy: 0.9743
Epoch 9/25
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0226 - accuracy: 0.9926 - val_loss: 0.1076 - val_accuracy: 0.9761
Epoch 10/25
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0210 - accuracy: 0.9928 - val_loss: 0.1174 - val_accuracy: 0.9732
Epoch 11/25
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0167 - accuracy: 0.9942 - val_loss: 0.1135 - val_accuracy: 0.9744
Epoch 12/25
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0155 - accuracy: 0.9942 - val_loss: 0.1145 - val_accuracy: 0.9772
Epoch 13/25
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0147 - accuracy: 0.9948 - val_loss: 0.1283 - val_accuracy: 0.9733
```



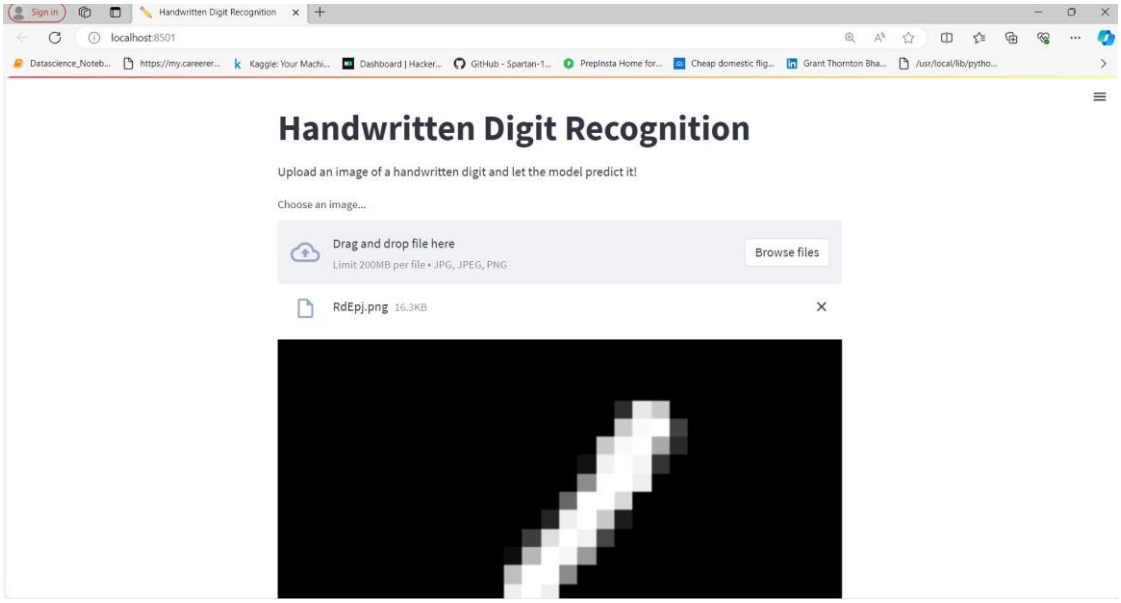


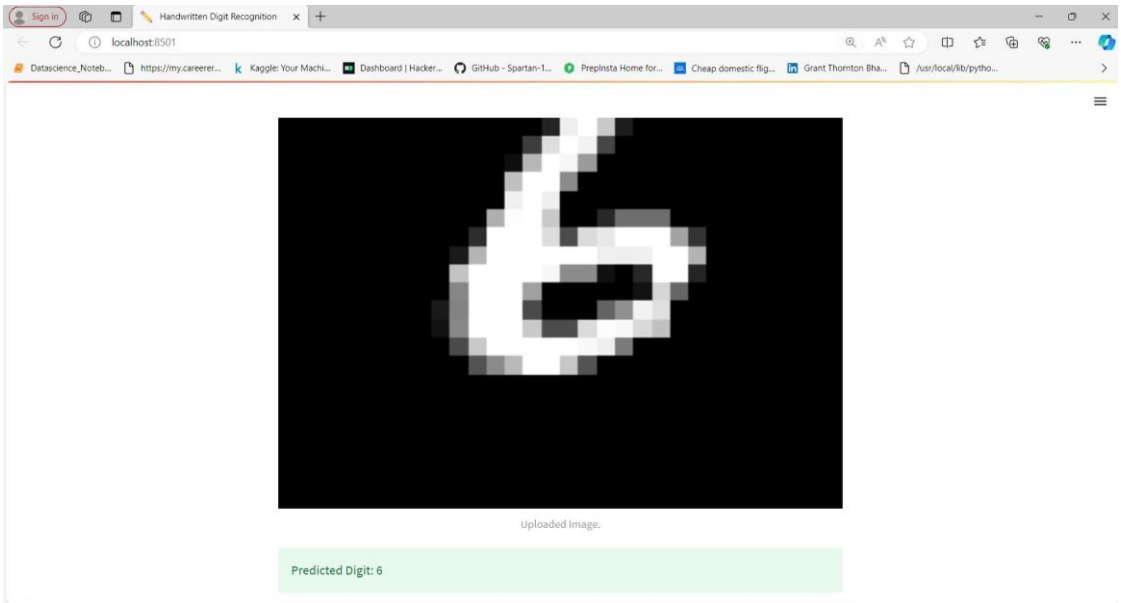
Streamlit Code:



RESULT AND DISCUSSION

The results obtained from training the ANN model show promising performance in terms of accuracy and loss. The model achieves a high accuracy on both the training and validation datasets (got 97.69% accuracy), indicating good generalization capability. Additionally, the deployed Streamlit application provides a seamless user experience for predicting handwritten digits.





CONCLUSION

In conclusion, this project successfully demonstrates the implementation of a handwritten digit recognition system using ANN and its deployment on Streamlit. The system shows promising results in accurately recognizing handwritten digits, with potential applications in digit recognition tasks. Future work may involve further optimization of the model architecture and exploring additional deployment options for wider accessibility

REFERENCES

1.LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

2.Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep Learning." MIT Press, 2016.

3.Simard, Patrice Y., et al. "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis." ICDAR, vol. 3, no. 2003, pp. 958-962, IEEE, 2003.

4.Baldi, Pierre, and Peter Sadowski. "Understanding dropout." Advances in neural information processing systems, vol. 28, pp. 2814-2822, 2015.

5.Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929-1958, 2014.

6.TensorFlow Documentation: https://www.tensorflow.org/guide

7.Keras Documentation: https://keras.io/

8.MNIST Database: http://yann.lecun.com/exdb/mnist/