

Data and Artificial Intelligence

Cyber Shujaa Program

Week 10 Assignment

Deep Learning

Student Name: Deborah Kwamboka Omae

Student ID: CS-DA02-25075

Table of Contents

Data and Artificial Intelligence	1
Cyber Shujaa Program.....	1
Week 10 Assignment Deep Learning	1
Introduction	4
Objectives.....	4
Tasks Completed	4
Step1 : Load the MNIST dataset using tensorflow.keras.datasets.....	4
Step 2: preprocess the data	6
Step 3: Building the ANN model	8
Step 4: Compiling the model.....	9
step 5: Training the model	9
step 6: evaluation test set.....	10
Step 7: visualizing training history	11
Step 8: confusion matrix	12
Step 9: printing classification report	13
Link to code.....	14
Conclusion.....	14

Table of figures

Figure 1: screenshot showing the loaded mnist dataset	6
Figure 2: screenshot showing code.....	7
Figure 3: screenshot showing the result of printing the sample images	8
Figure 4: screenshot showing code.....	9
Figure 5: screenshot showing the training of the model	10
Figure 6: screenshot showing the accuracy	11
Figure 7: screenshot showing the plot.....	12
Figure 8: screenshot showing the confusion matrix	13
Figure 9: screenshot showing classification report.....	14

Introduction

In this assignment, I was required to apply my understanding of Artificial Neural Networks and TensorFlow/Keras to build, train, evaluate, and document an image classification model using the MNIST dataset.

MNIST stands for Modified National Institute of Standards and Technology dataset. It is a benchmark dataset widely used in training and testing machine learning and deep learning models for handwritten digit recognition.

Item	Description
Images	70,000 grayscale images of digits (0–9)
Size	Each image is 28×28 pixels (784 total)
Labels	10 classes (digits 0 to 9)
Split	60,000 training images, 10,000 test images

Objectives

- Preprocess and explore image data
- Design and build the ANN architecture
- Compile, Train, and Validate the deep learning model
- Evaluate the model on the test set and report the final test accuracy
- Visualize model training history
- Save and load trained models using the modern Keras format

Tasks Completed

Step1 : Load the MNIST dataset using tensorflow.keras.datasets.

I imported the necessary libraries first and then loaded the data

Code

Step 1: Import Libraries

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense, Dropout

from tensorflow.keras.utils import to_categorical

from sklearn.metrics import confusion_matrix, classification_report
```

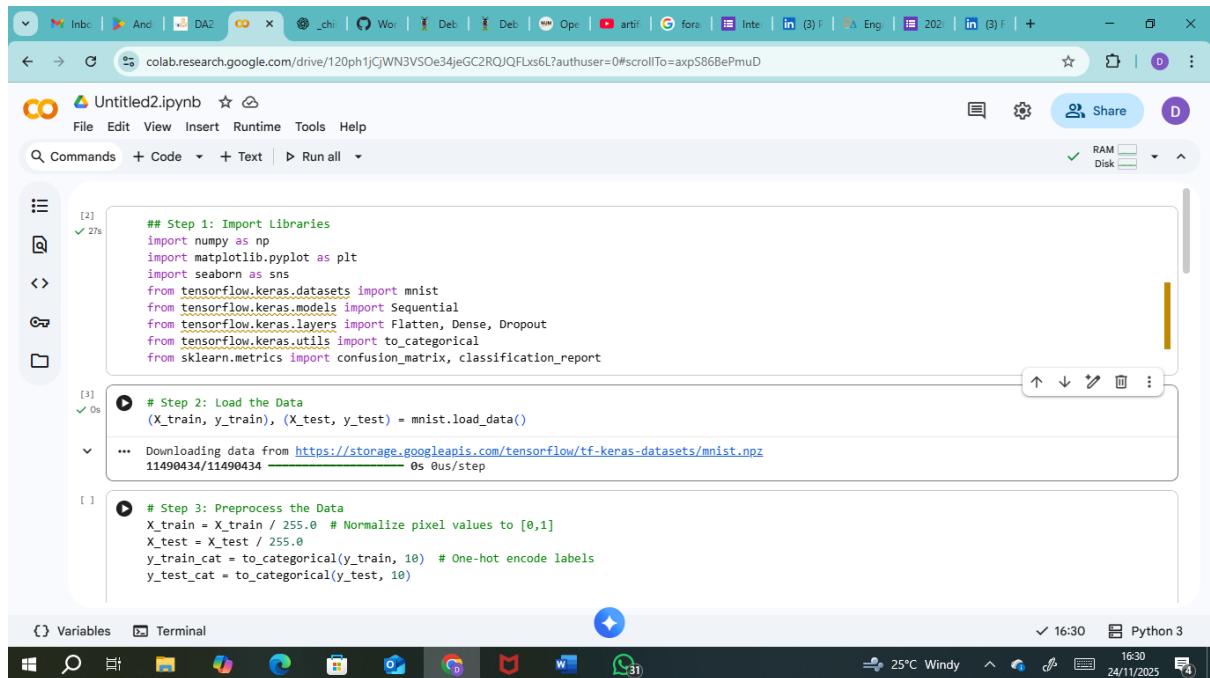
Step 2: Load the Data

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Code explanation

This line loads the MNIST dataset and splits it into training and test sets, giving you the images and their correct digit labels. The MNIST dataset already comes pre split.

Screenshot:



The screenshot shows a Google Colab notebook titled "Untitled2.ipynb". The code in the first cell imports necessary libraries (numpy, matplotlib.pyplot, seaborn, tensorflow.keras.datasets, tensorflow.keras.models, tensorflow.keras.layers, tensorflow.keras.utils, and sklearn.metrics). The second cell loads the MNIST dataset using `mnist.load_data()`. The third cell shows the download progress of the dataset from a Google Cloud storage URL. The fourth cell preprocesses the data by normalizing pixel values to [0,1] and one-hot encoding the labels. The code is written in Python 3.

```

## Step 1: Import Libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report

# Step 2: Load the Data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Step 3: Preprocess the Data
X_train = X_train / 255.0 # Normalize pixel values to [0,1]
X_test = X_test / 255.0
y_train_cat = to_categorical(y_train, 10) # One-hot encode labels
y_test_cat = to_categorical(y_test, 10)

```

Figure 1: screenshot showing the loaded mnist dataset

Step 2: preprocess the data

In this step I normalize the pixel values to a [0,1] range and One-hot encode the labels using `to_categorical` and plotted some digits from the dataset.

Code:

```
# Step 3: Preprocess the Data
```

```
X_train = X_train / 255.0 # Normalize pixel values to [0,1]
```

```
X_test = X_test / 255.0
```

```
y_train_cat = to_categorical(y_train, 10) # One-hot encode labels
```

```
y_test_cat = to_categorical(y_test, 10)
```

```
# Plot some digits from dataset
```

```
selected_indices = [10, 25, 75, 300, 501, 999, 1234, 1500, 1999] # Choose which image indices to display
```

```
plt.figure(figsize=(8, 8))
```

for i, idx in enumerate(selected_indices):

plt.subplot(3, 3, i + 1)

plt.imshow(X_train[idx], cmap='gray')

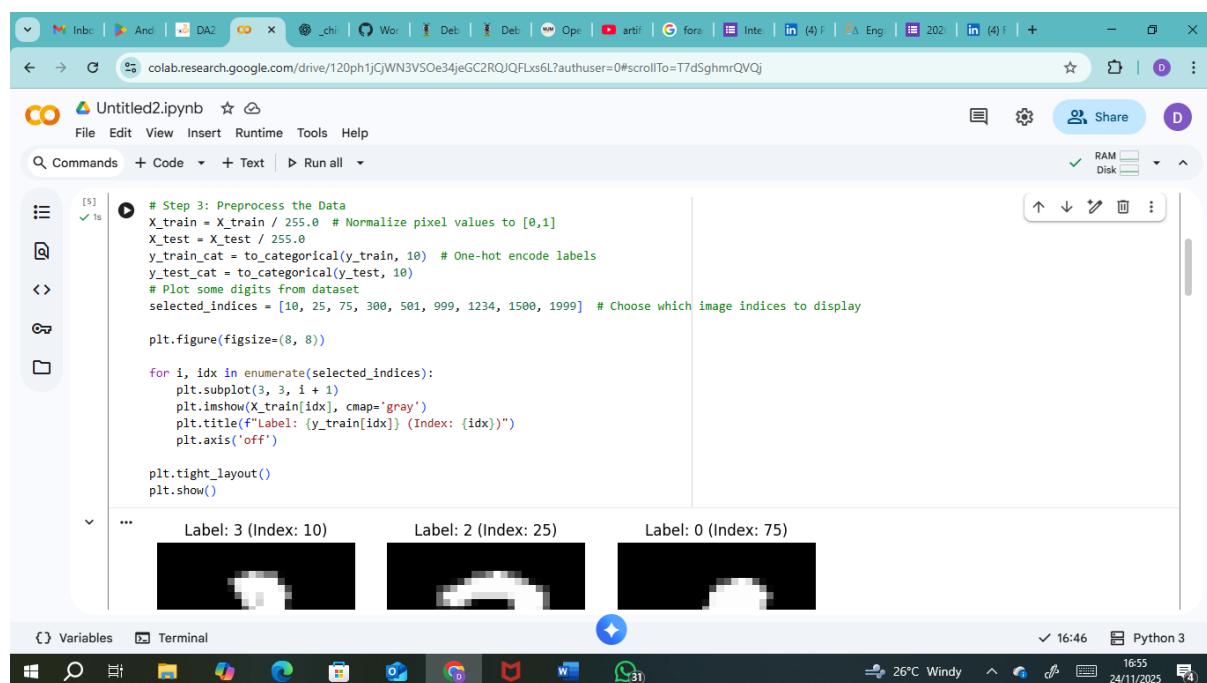
plt.title(f"Label: {y_train[idx]} (Index: {idx})")

plt.axis('off')

plt.tight_layout()

plt.show()

screenshot:



The screenshot shows a Google Colab notebook titled "Untitled2.ipynb". The code cell contains the following Python code:

```
# Step 3: Preprocess the Data
X_train = X_train / 255.0 # Normalize pixel values to [0,1]
X_test = X_test / 255.0
y_train_cat = to_categorical(y_train, 10) # One-hot encode labels
y_test_cat = to_categorical(y_test, 10)
# Plot some digits from dataset
selected_indices = [10, 25, 75, 300, 501, 999, 1234, 1500, 1999] # Choose which image indices to display

plt.figure(figsize=(8, 8))

for i, idx in enumerate(selected_indices):
    plt.subplot(3, 3, i + 1)
    plt.imshow(X_train[idx], cmap='gray')
    plt.title(f"Label: {y_train[idx]} (Index: {idx})")
    plt.axis('off')

plt.tight_layout()
plt.show()
```

The output of the code shows three digit images with their labels:

- Label: 3 (Index: 10)
- Label: 2 (Index: 25)
- Label: 0 (Index: 75)

Figure 2: screenshot showing code

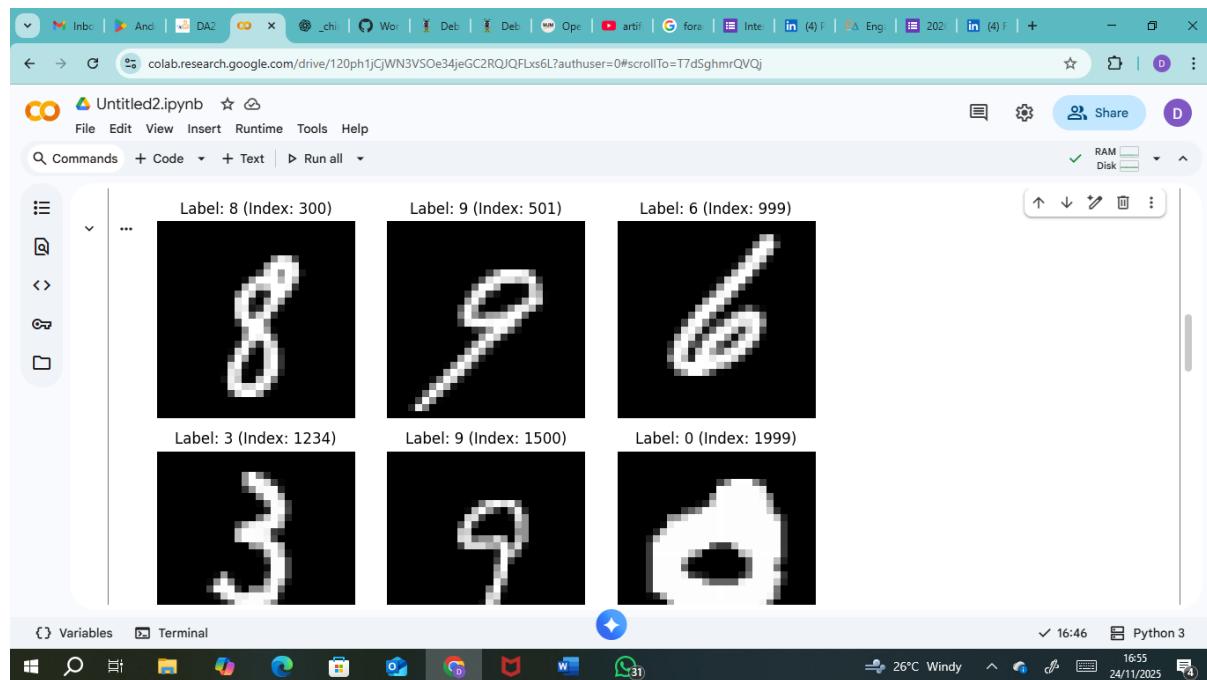


Figure 3: screenshot showing the result of printing the sample images

Step 3: Building the ANN model

In this step I used the Sequential model. I Included at least a flatten layer as input layer, Two Dense hidden layers (e.g., 128 and 64 neurons) with ReLU activation, Dropout layers (e.g., 0.3) for regularization, Output layer with 10 neurons and softmax activation.

Code:

```
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])
```

Screenshot:

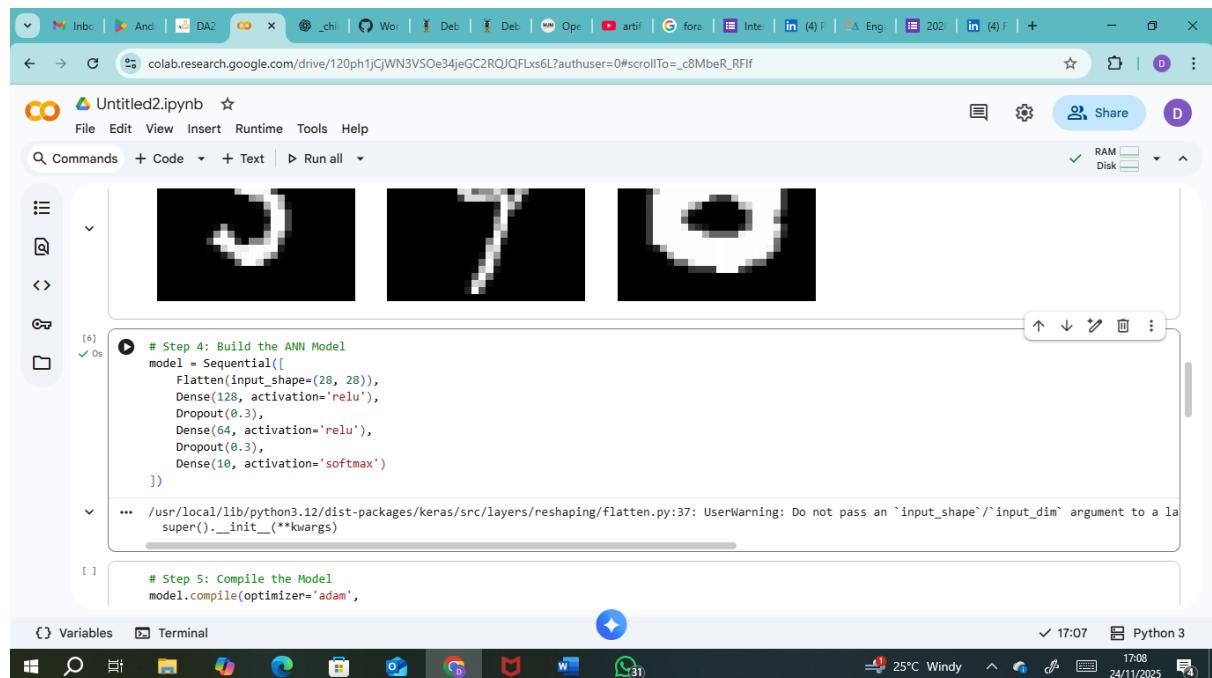


Figure 4: screenshot showing code

Step 4: Compiling the model

I then compiled with adam optimizer and categorical_crossentropyloss.

Code:

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

step 5: Training the model

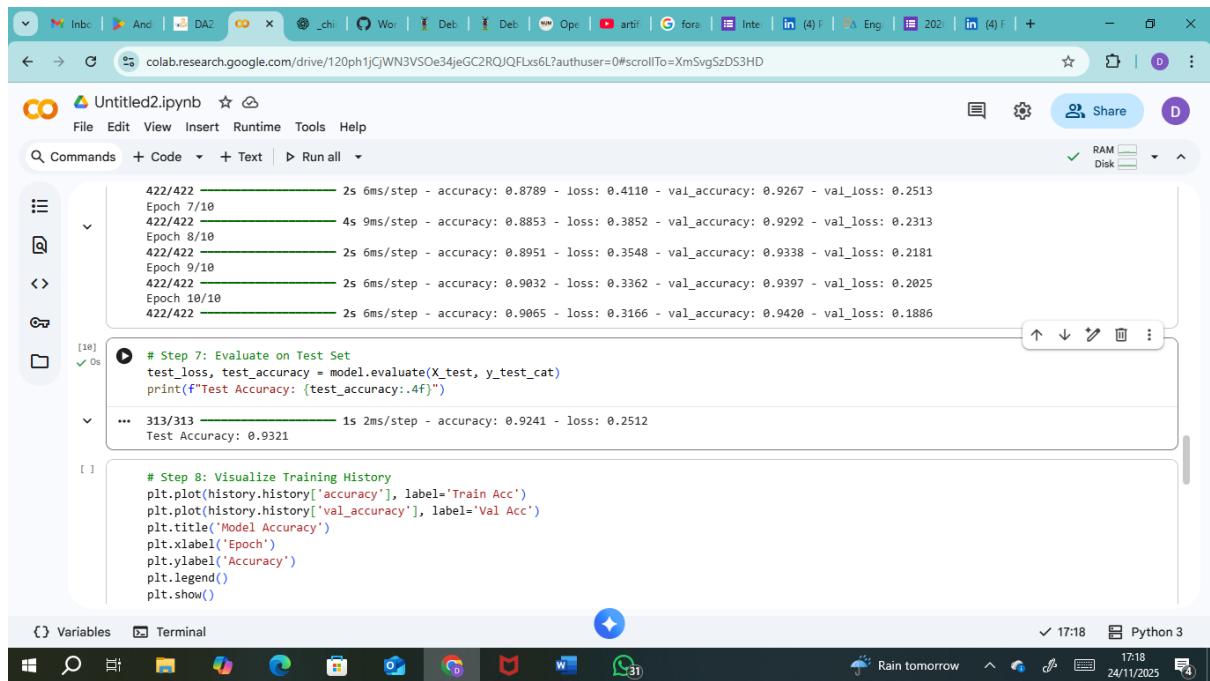
In this step I trained the model for 10 epochs, using a batch_size of 128 and a validation_split of 0.1.

code:

```
history = model.fit(X_train, y_train_cat,
                     epochs=10,
                     batch_size=128,
```

validation_split=0.1)

screenshot:



The screenshot shows a Google Colab notebook titled "Untitled2.ipynb". The notebook displays the following code and its execution results:

```

File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[10] 422/422 - 2s 6ms/step - accuracy: 0.8789 - loss: 0.4110 - val_accuracy: 0.9267 - val_loss: 0.2513
Epoch 7/10
422/422 - 4s 9ms/step - accuracy: 0.8853 - loss: 0.3852 - val_accuracy: 0.9292 - val_loss: 0.2313
Epoch 8/10
422/422 - 2s 6ms/step - accuracy: 0.8951 - loss: 0.3548 - val_accuracy: 0.9338 - val_loss: 0.2181
Epoch 9/10
422/422 - 2s 6ms/step - accuracy: 0.9032 - loss: 0.3362 - val_accuracy: 0.9397 - val_loss: 0.2025
Epoch 10/10
422/422 - 2s 6ms/step - accuracy: 0.9065 - loss: 0.3166 - val_accuracy: 0.9420 - val_loss: 0.1886

# Step 7: Evaluate on Test Set
test_loss, test_accuracy = model.evaluate(X_test, y_test_cat)
print(f"Test Accuracy: {test_accuracy:.4f}")

... 313/313 - 1s 2ms/step - accuracy: 0.9241 - loss: 0.2512
Test Accuracy: 0.9321

# Step 8: Visualize Training History
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```

The notebook interface includes a sidebar with file navigation, a toolbar with various icons, and a status bar at the bottom showing the time (17:18), date (24/11/2025), and Python version (Python 3). The status bar also indicates "Rain tomorrow".

Figure 5: screenshot showing the training of the model

step 6: evaluation test set

In this step I used accuracy as the evaluation metrics

Code:

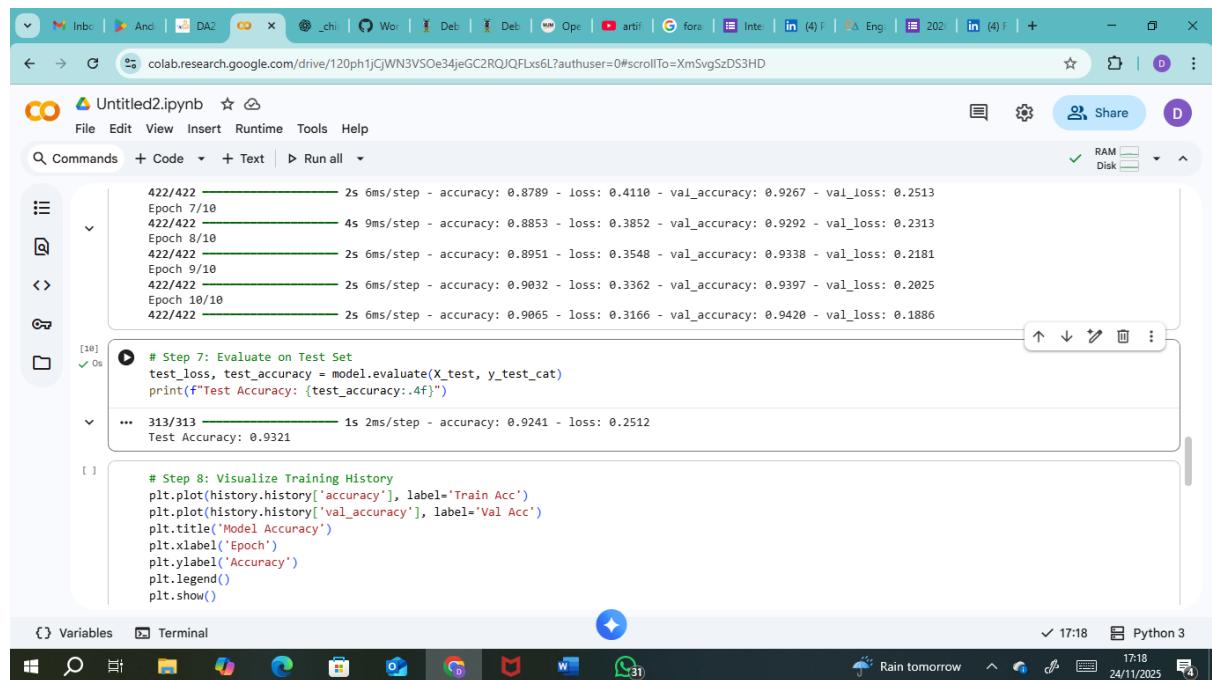
```

test_loss, test_accuracy = model.evaluate(X_test, y_test_cat)

print(f"Test Accuracy: {test_accuracy:.4f}")

```

screenshot:



The screenshot shows a Google Colab notebook titled "Untitled2.ipynb". The code cell [10] contains the following Python code:

```

# Step 7: Evaluate on Test Set
test_loss, test_accuracy = model.evaluate(X_test, y_test_cat)
print(f"Test Accuracy: {test_accuracy:.4f}")

# Step 8: Visualize Training History
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```

The output of the first part of the code shows the training and validation accuracy for each epoch from 1 to 10. The output of the second part of the code shows a line plot comparing the training accuracy (blue line) and validation accuracy (orange line) across 10 epochs.

Figure 6: screenshot showing the accuracy

Step 7: visualizing training history

In this step I visualized the training history

Code:

```

plt.plot(history.history['accuracy'], label='Train Acc')

plt.plot(history.history['val_accuracy'], label='Val Acc')

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

```

screenshot:

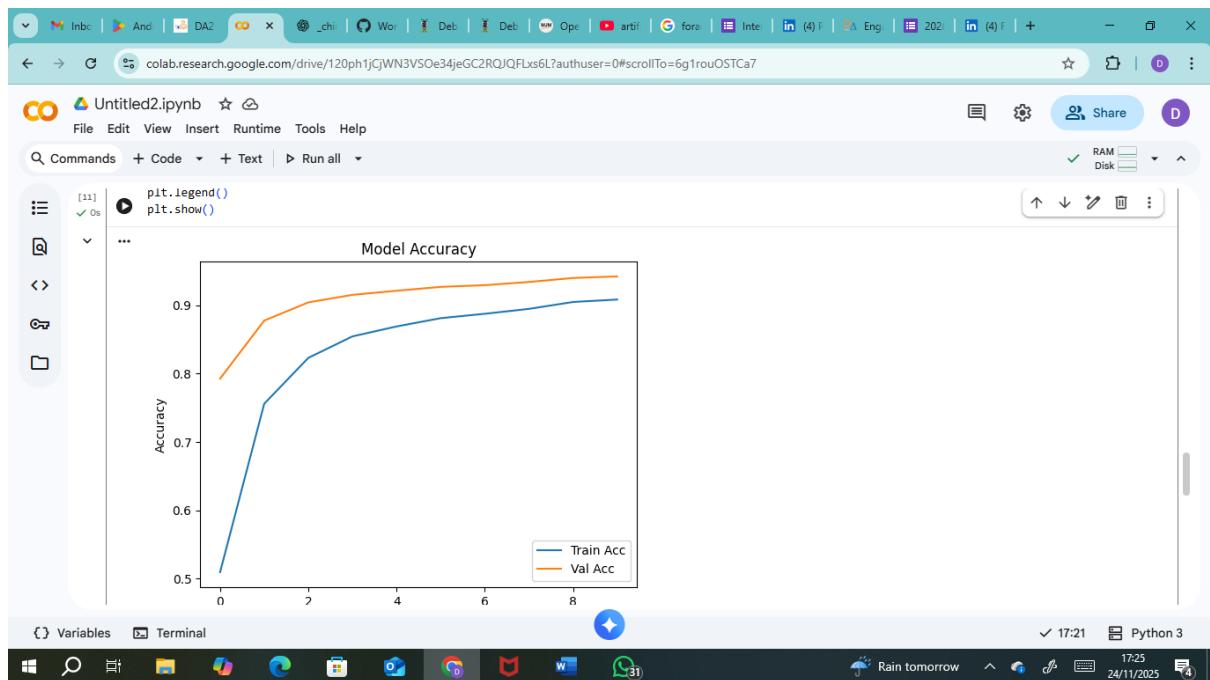


Figure 7: screenshot showing the plot

Step 8: confusion matrix

In this step I display a confusion matrix using seaborn.heatmap.

Code:

```
y_pred = model.predict(X_test)

y_pred_classes = np.argmax(y_pred, axis=1)

cm = confusion_matrix(y_test, y_pred_classes)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()
```

screenshot:

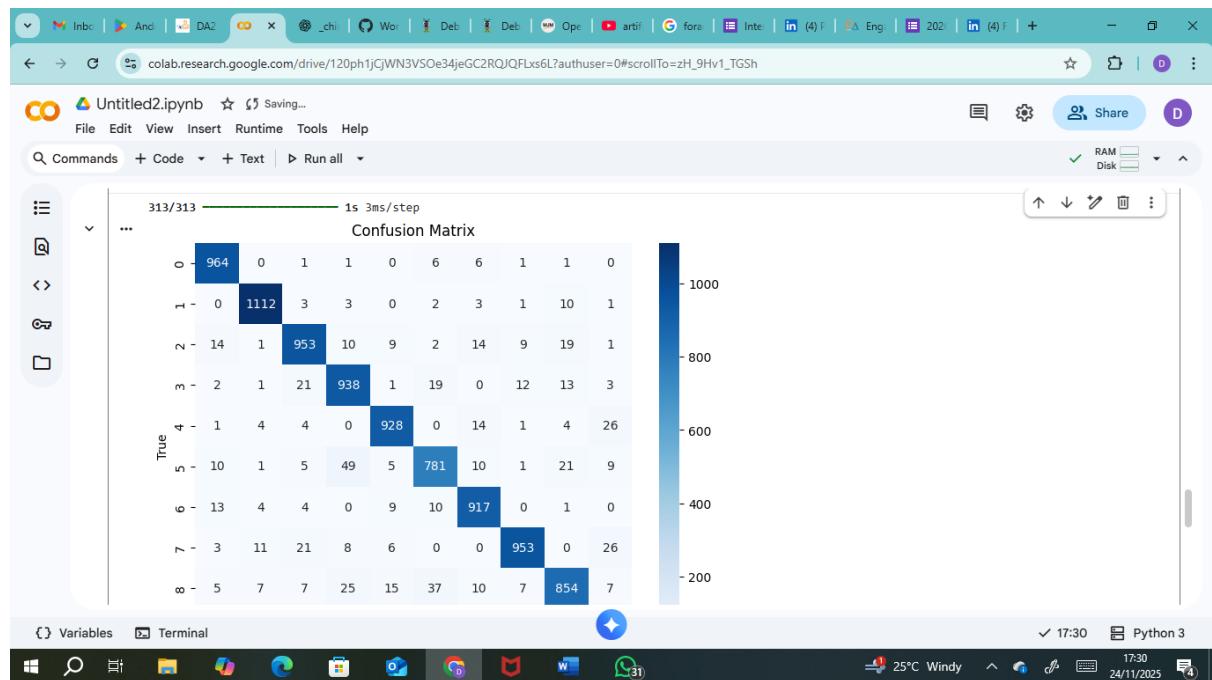


Figure 8: screenshot showing the confusion matrix

Step 9: printing classification report

In this step I printed a classification report showing precision, recall, and F1-score and saved the trained model in the native Keras format.

Code:

```
print(classification_report(y_test, y_pred_classes))

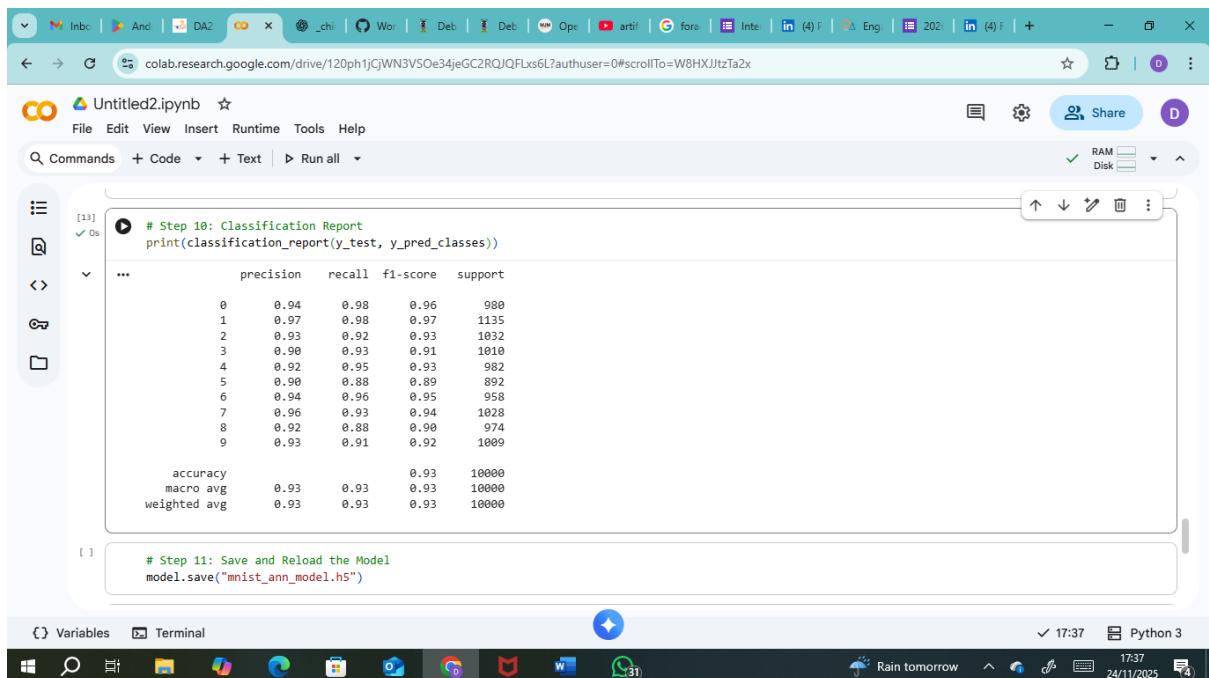
model.save("mnist_ann_model.h5")
```

```
from tensorflow.keras.models import load_model

reloaded_model = load_model("mnist_ann_model.h5")

reloaded_model.evaluate(X_test, y_test_cat)
```

screenshot:



```
# Step 10: Classification Report
print(classification_report(y_test, y_pred_classes))

...      precision    recall    f1-score   support
          0       0.94     0.98     0.96     980
          1       0.97     0.98     0.97    1135
          2       0.93     0.92     0.93    1032
          3       0.90     0.93     0.91    1010
          4       0.92     0.95     0.93    982
          5       0.98     0.88     0.89     892
          6       0.94     0.96     0.95     958
          7       0.96     0.93     0.94    1028
          8       0.92     0.88     0.90     974
          9       0.93     0.91     0.92    1009

accuracy                           0.93    10000
macro avg       0.93     0.93     0.93    10000
weighted avg    0.93     0.93     0.93    10000

# Step 11: Save and Reload the Model
model.save("mnist_ann_model.h5")
```

Figure 9: screenshot showing classification report

Link to code

Link to Code:

<https://colab.research.google.com/drive/120ph1jCjWN3VSOe34jeGC2RQJQFLxs6L?usp=sharing>

Conclusion

In this assignment, I successfully built and evaluated an Artificial Neural Network model for handwritten digit recognition using the MNIST dataset. The workflow covered all key deep learning steps, including data preprocessing, visualization, model design, training, and performance evaluation. The ANN achieved strong accuracy on both the validation and test sets, demonstrating its ability to generalize well to unseen data. Through the confusion matrix and classification report, I gained deeper insight into the model's strengths and weaknesses across different digit classes. Overall, this assignment strengthened my

understanding of neural network architectures, model optimization, and practical deep learning workflows using TensorFlow/Keras.