Work Integrated Learning Programmes

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Department of Computer Science and Information Technology

Artificial Intelligence: S1-24_SSWTZC444

Name – NIDHI SINGH

Roll – 2021WB15811

Lab Work Sheet – 3

Lab Exercise:

[A] Problem Statement: In a Coin Game, two players take turns to pick coins from a set of coins arranged in a row. At each turn, a player can choose either the leftmost or the rightmost coin from the remaining coins. The player with the highest total value of coins at the end of the game wins.

Goal:

- Implement the **MiniMax Algorithm** to simulate the optimal strategy for both players in a **Coin Game**. Each player should make a move that maximizes their coin value while minimizing the opponent's gain. The output should indicate the **maximum coins** a player can win, given that both players play optimally.

Challenge:

- To determine the optimal strategy for each player by evaluating all possible moves and outcomes.
- The MiniMax algorithm should explore all possible future states of the game and make decisions accordingly, ensuring that both players are playing optimally.
- The algorithm needs to account for both players acting with the best interest in mind at each step.

Input:

-A list of integers representing the values of the coins in a row.

coins = [8, 15, 3, 7]

-The list represents a set of coins arranged in a row with the following values:

- Coin 1: 8
- Coin 2: 15
- Coin 3: 3
- Coin 4: 7

Task:

Implement the **MiniMax Algorithm** to simulate the **Coin Game**.

The algorithm should work as follows:

1. A player can choose either the leftmost or the rightmost coin in the row.
2. The game continues until no coins remain.

3. The algorithm should calculate the maximum coins the first player can win, assuming both players play optimally.

Expected Output:

- The maximum coins the first player can win: 22

Ans :-

To solve this problem, we will implement the MiniMax algorithm with recursion and memoization. The key challenge is to determine the maximum coins the first player can win while both players play optimally. Let's break the solution into steps:

Algorithm:-

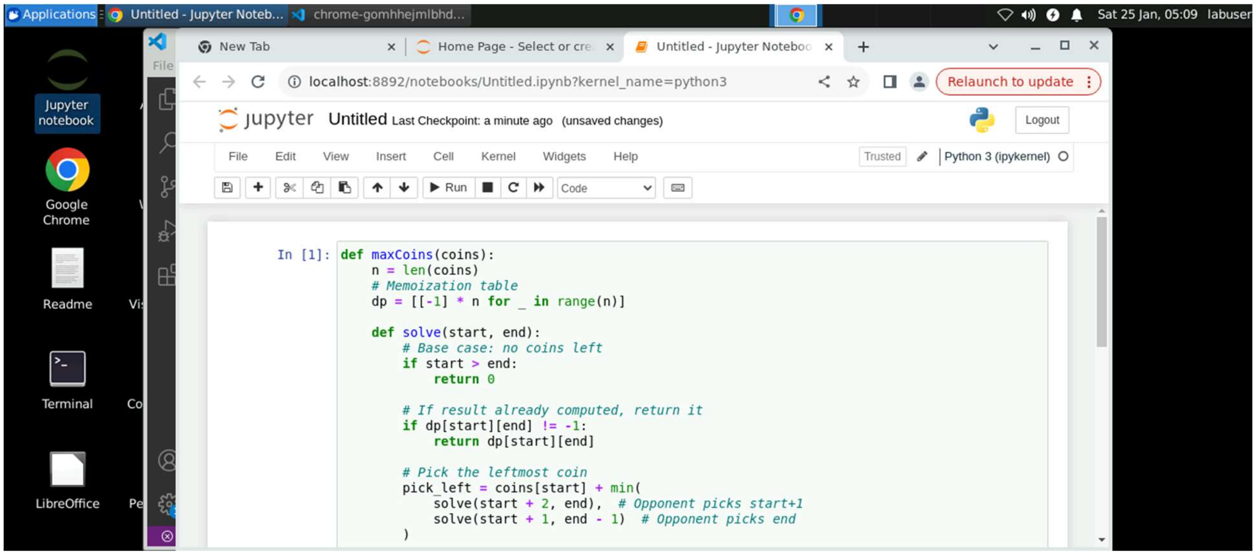
Algorithm

1. **Problem Modeling**:
    - At each turn, the current player can choose the leftmost or rightmost coin.
    - The goal is to maximize the current player's total while minimizing the opponent's total.
    - Use the MiniMax principle: maximize the player's gain while minimizing the opponent's future gain.
2. **Recursive Function**:
    - Use a recursive function maxCoins(start, end) where:
        - start and end represent the range of the remaining coins.
        - It returns the maximum coins the current player can collect from the range [start, end].
3. **Base Case**:
    - If start > end, return 0 (no coins are left).
4. **Choice Evaluation**:
    - If the current player picks the leftmost coin (coins[start]):
        - The opponent will play optimally for the remaining range [start+1, end].
    - If the current player picks the rightmost coin (coins[end]):
        - The opponent will play optimally for the remaining range [start, end-1].
5. **Memoization**:
    - Use a 2D array dp to store intermediate results for subproblems to avoid redundant calculations.
6. **Output**:
    - Call the recursive function for the entire range of coins and output the result.

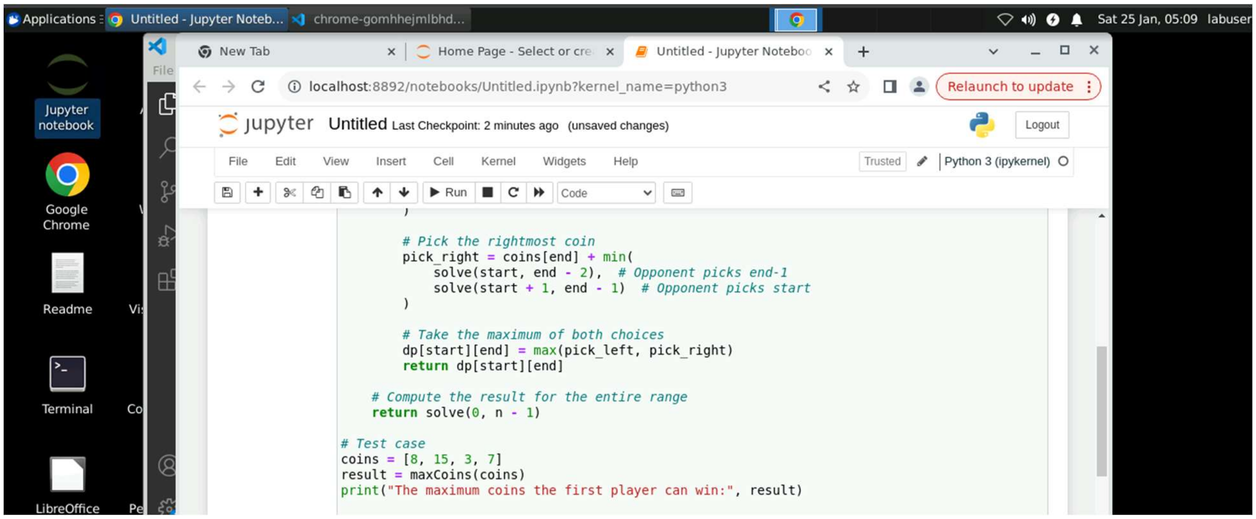
Below is the detailed explanatiom with reference to screenshot:-

```
def maxCoins(coins):
  n = len(coins)
  # Memoization table
  dp = [[-1] * n for _ in range(n)]
```

```
    def solve(start, end):
        # Base case: no coins left
        if start > end:
            return 0

        # If result already computed, return it
        if dp[start][end] != -1:
            return dp[start][end]

        # Pick the leftmost coin
        pick_left = coins[start] + min(
            solve(start + 2, end),  # Opponent picks start+1
            solve(start + 1, end - 1)  # Opponent picks end
        )

        # Pick the rightmost coin
        pick_right = coins[end] + min(
            solve(start, end - 2),  # Opponent picks end-1
            solve(start + 1, end - 1)  # Opponent picks start
        )

        # Take the maximum of both choices
        dp[start][end] = max(pick_left, pick_right)
        return dp[start][end]

    # Compute the result for the entire range
    return solve(0, n - 1)

# Test case
coins = [8, 15, 3, 7]
result = maxCoins(coins)
print("The maximum coins the first player can win:", result)
```

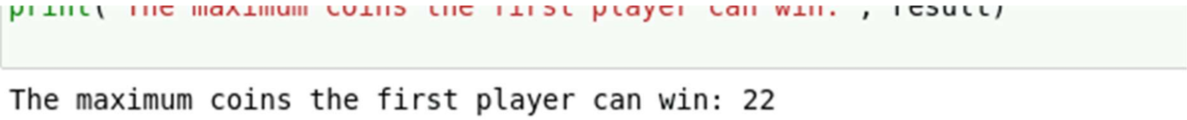

Applications
Untitled - Jupyter Noteb...
chrome-gomhhejmlbhd...
Sat 25 Jan, 05:09 labuser
Jupyter notebook
Google Chrome
Readme
Terminal
LibreOffice
New Tab
Home Page - Select or cre
Untitled - Jupyter Noteboo
localhost:8892/notebooks/Untitled.ipynb?kernel_name=python3
Relaunch to update
jupyter Untitled Last Checkpoint: a minute ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [1]: def maxCoins(coins):
            n = len(coins)
            # Memoization table
            dp = [[-1] * n for _ in range(n)]

            def solve(start, end):
                # Base case: no coins left
                if start > end:
                    return 0

                # If result already computed, return it
                if dp[start][end] != -1:
                    return dp[start][end]

                # Pick the leftmost coin
                pick_left = coins[start] + min(
                    solve(start + 2, end),  # Opponent picks start+1
                    solve(start + 1, end - 1)  # Opponent picks end
                )
```


Applications
Untitled - Jupyter Noteb...
chrome-gomhhejmlbhd...
Sat 25 Jan, 05:09 labuser
Jupyter notebook
Google Chrome
Readme
Terminal
LibreOffice
New Tab
Home Page - Select or cre
Untitled - Jupyter Noteboo
localhost:8892/notebooks/Untitled.ipynb?kernel_name=python3
Relaunch to update
jupyter Untitled Last Checkpoint: 2 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
                )

                # Pick the rightmost coin
                pick_right = coins[end] + min(
                    solve(start, end - 2),  # Opponent picks end-1
                    solve(start + 1, end - 1)  # Opponent picks start
                )

                # Take the maximum of both choices
                dp[start][end] = max(pick_left, pick_right)
                return dp[start][end]

            # Compute the result for the entire range
            return solve(0, n - 1)

        # Test case
        coins = [8, 15, 3, 7]
        result = maxCoins(coins)
        print("The maximum coins the first player can win:", result)
```

Output:-



```
print("The maximum coins the first player can win:", result)

The maximum coins the first player can win: 22
```

[B] Problem: Digit Classification

Classify handwritten digits (0-9) using a neural network.

Dataset: MNIST Handwritten Digits (https://www.kaggle.com/c/digit-recognizer/data)

Task:

1. Build a neural network model using Keras or PyTorch to classify handwritten digits.
2. Split the dataset into training and validation sets. Train the model for a few epochs, adjusting hyperparameters like the learning rate, batch size, and number of epochs as needed.
3. Evaluate accuracy, precision, and recall on the validation/test set.

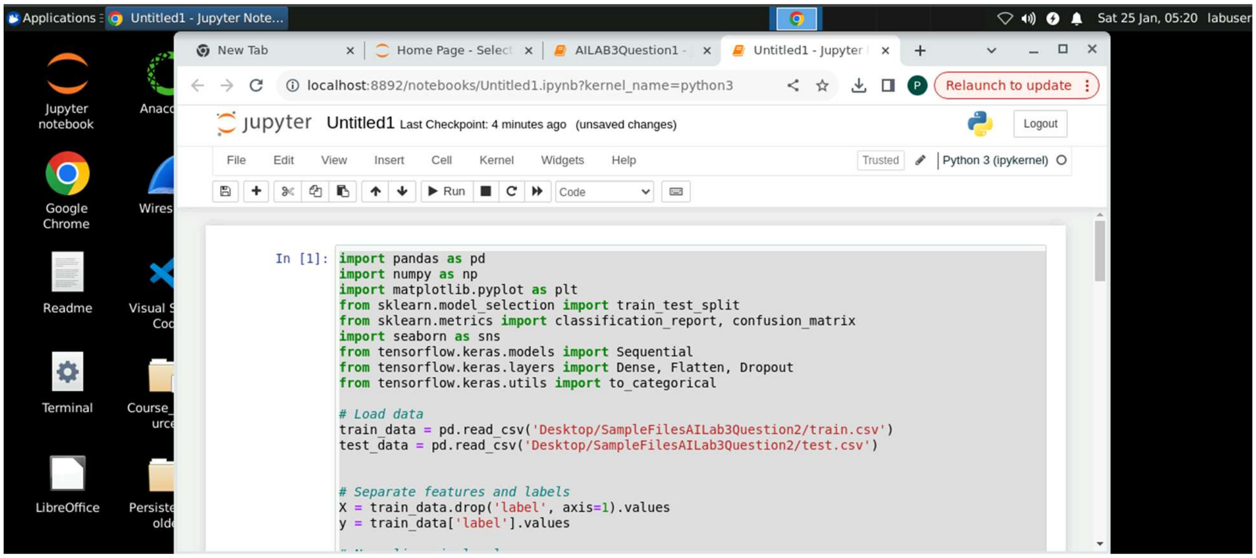
Expected Output:

- Display the model architecture and the number of parameters.
- Show accuracy, precision, and recall.
- Confusion matrix for classification results.

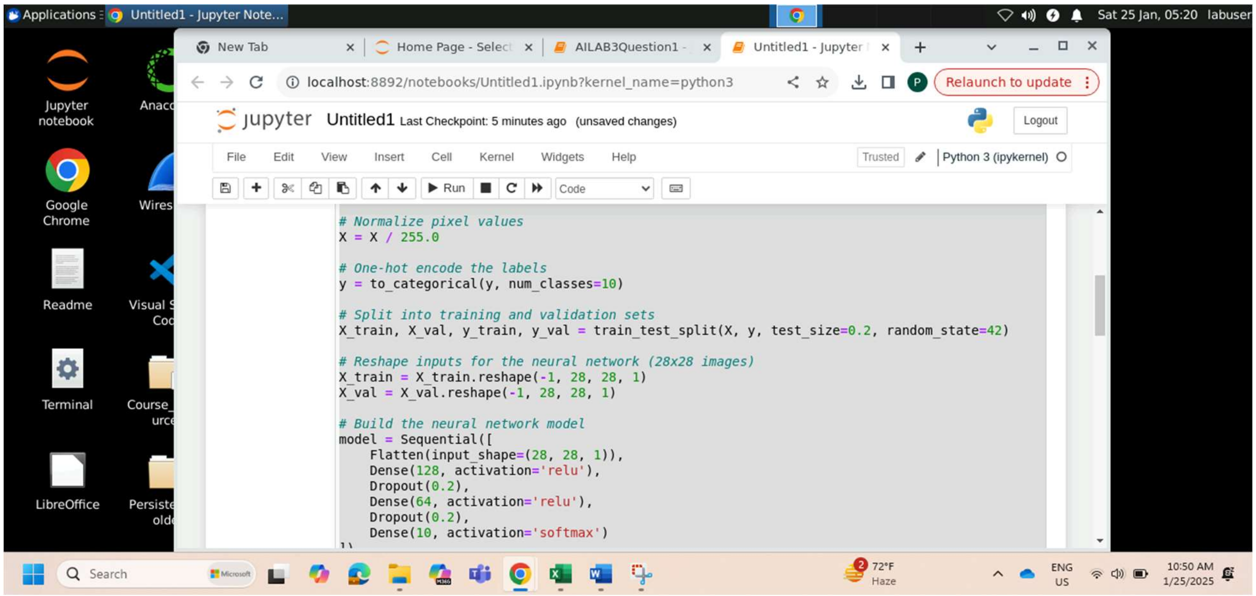
Ans :-

I'll create a Jupyter notebook code snippet for classifying handwritten digits using a neural network with Keras, leveraging the provided MNIST dataset (train.csv and test.csv). The code will include model building, training, evaluation, and visualization of results.
Let's proceed!
Mnist Digit Classifier
The code above:

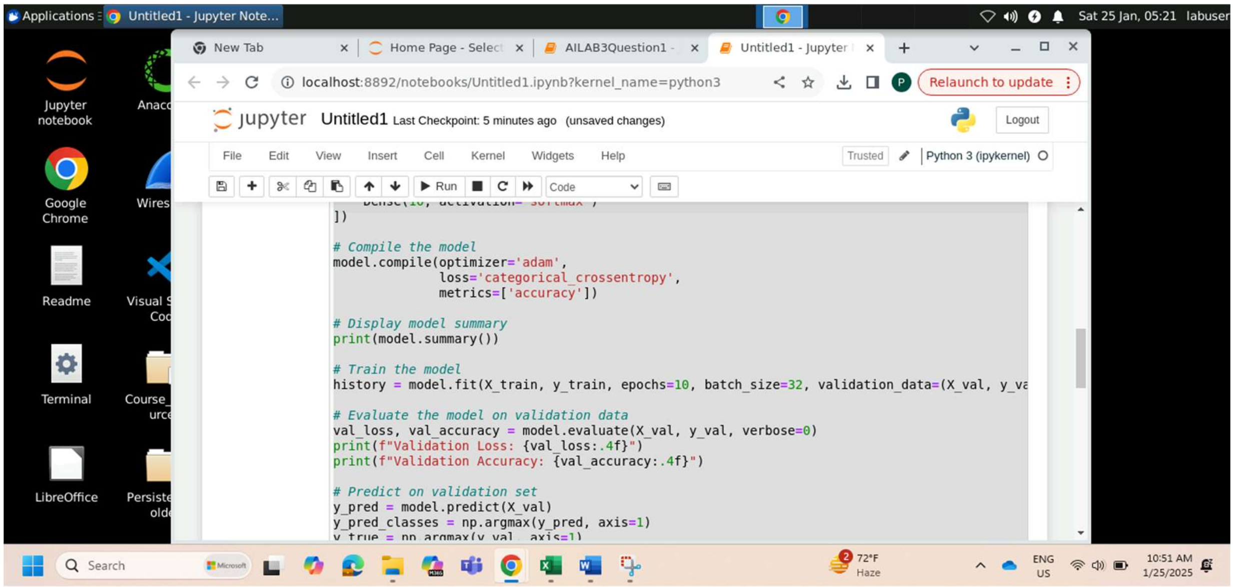
1. **Loads and preprocesses the data**:
   - Reads train.csv for training and validation.
   - Normalizes pixel values to the range [0, 1].
   - One-hot encodes labels.
   - Splits the data into training and validation sets.
2. **Builds a neural network model**:
   - Uses a simple feedforward architecture with dropout layers for regularization.
3. **Trains the model**:
   - Uses the Adam optimizer and categorical cross-entropy loss.
   - Trains for 10 epochs, with a batch size of 32.
4. **Evaluates the model**:
   - Outputs accuracy, precision, recall, and a confusion matrix.
5. **Saves the trained model**:
   - Saves the model as mnist_digit_classifier.h5.


localhost:8892/notebooks/Untitled1.ipynb?kernel_name=python3
jupyter Untitled1 Last Checkpoint: 4 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [1]:

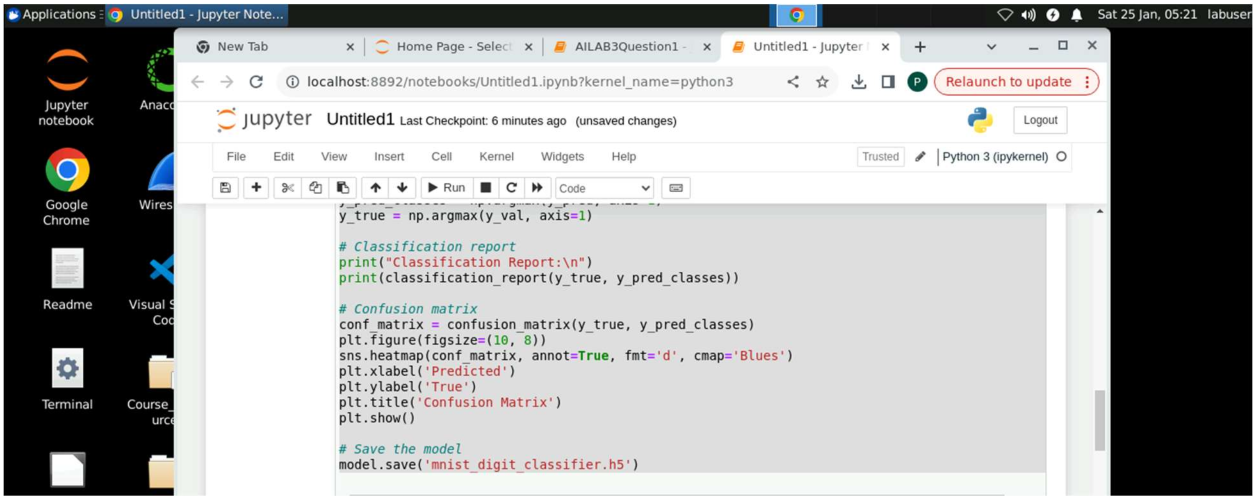
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.utils import to_categorical

# Load data
train_data = pd.read_csv('Desktop/SampleFilesAILab3Question2/train.csv')
test_data = pd.read_csv('Desktop/SampleFilesAILab3Question2/test.csv')


# Separate features and labels
X = train_data.drop('label', axis=1).values
y = train_data['label'].values
```


localhost:8892/notebooks/Untitled1.ipynb?kernel_name=python3
jupyter Untitled1 Last Checkpoint: 5 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```python
# Normalize pixel values
X = X / 255.0

# One-hot encode the labels
y = to_categorical(y, num_classes=10)

# Split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Reshape inputs for the neural network (28x28 images)
X_train = X_train.reshape(-1, 28, 28, 1)
X_val = X_val.reshape(-1, 28, 28, 1)

# Build the neural network model
model = Sequential([
    Flatten(input_shape=(28, 28, 1)),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax')
])
```

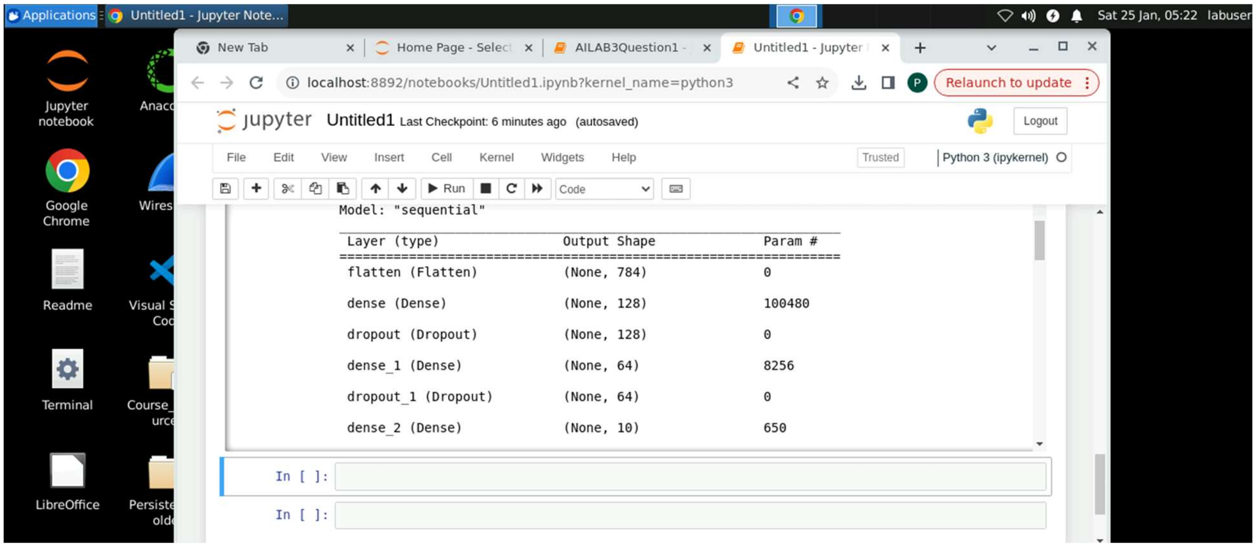


```
    Dense(10, activation='softmax')
])

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

# Display model summary
print(model.summary())

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_va

# Evaluate the model on validation data
val_loss, val_accuracy = model.evaluate(X_val, y_val, verbose=0)
print(f"Validation Loss: {val_loss:.4f}")
print(f"Validation Accuracy: {val_accuracy:.4f}")

# Predict on validation set
y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_val, axis=1)
```

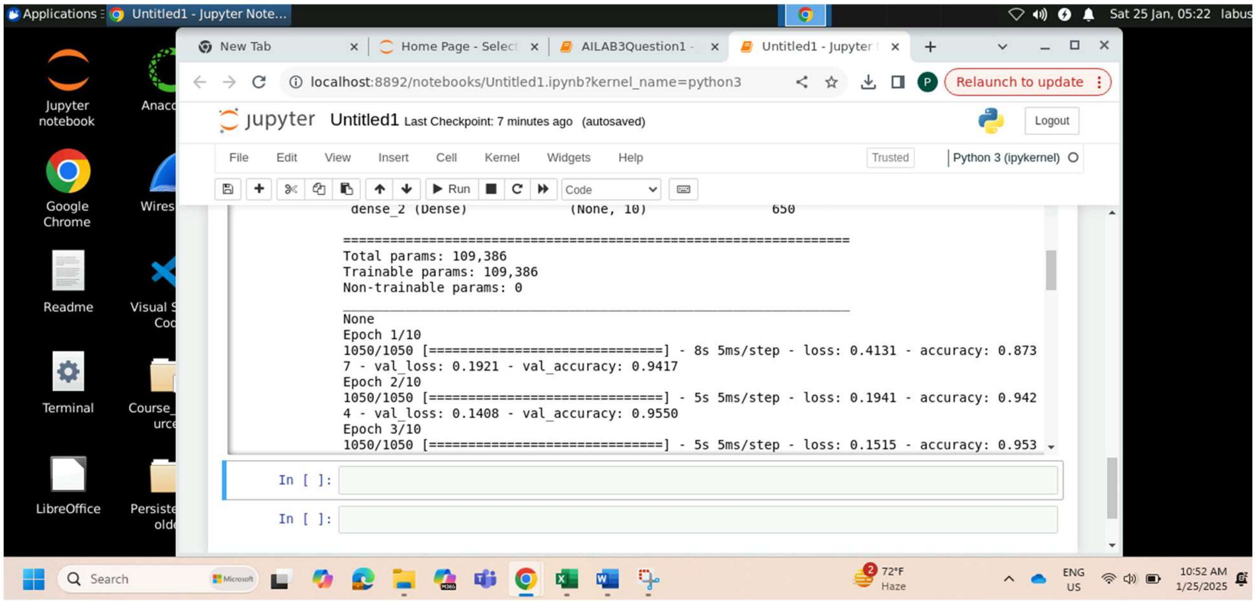


```
y_true = np.argmax(y_val, axis=1)

# Classification report
print("Classification Report:\n")
print(classification_report(y_true, y_pred_classes))

# Confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Save the model
model.save('mnist_digit_classifier.h5')
```

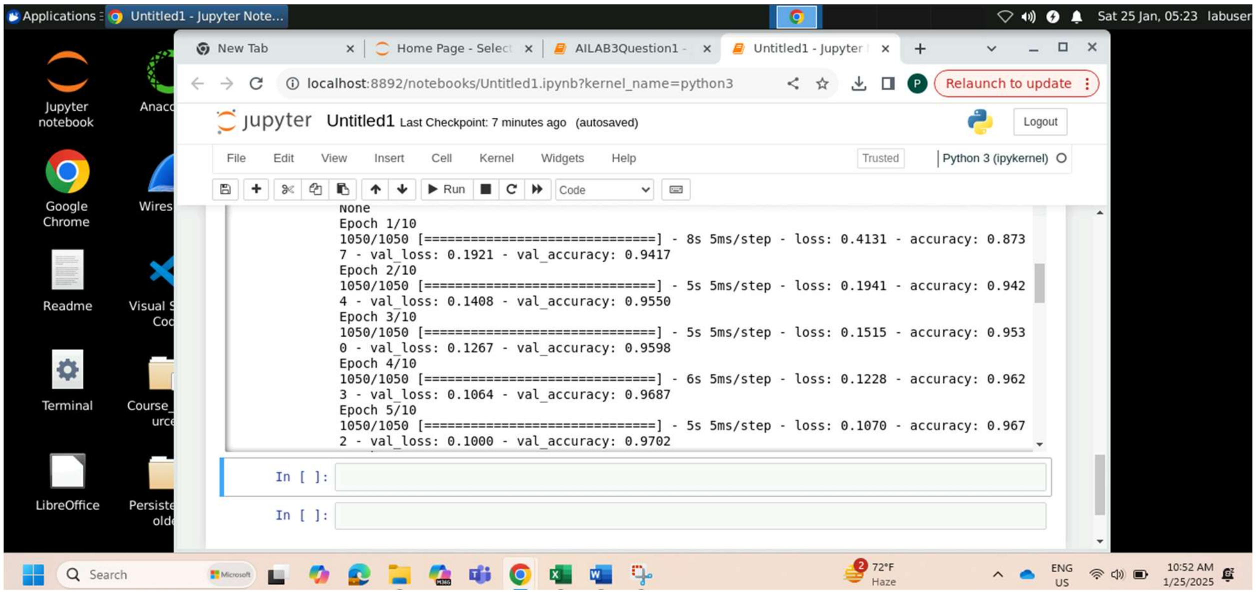
Output:-



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

 dense (Dense)               (None, 128)               100480

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 10)                650
```



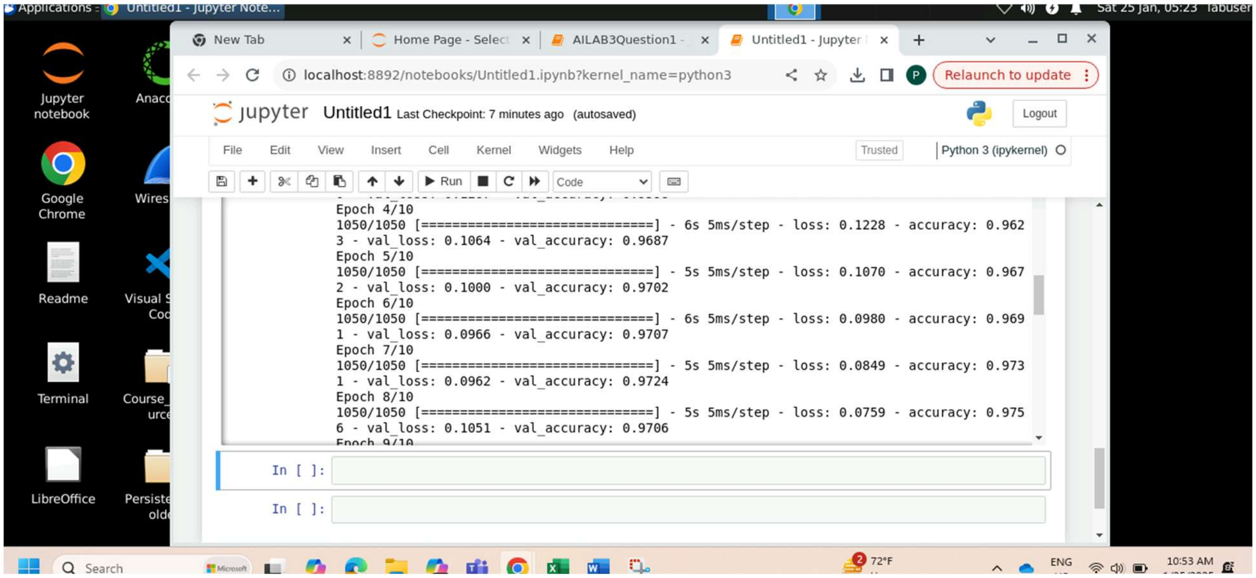
```
 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
1050/1050 [==============================] - 8s 5ms/step - loss: 0.4131 - accuracy: 0.873
7 - val_loss: 0.1921 - val_accuracy: 0.9417
Epoch 2/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.1941 - accuracy: 0.942
4 - val_loss: 0.1408 - val_accuracy: 0.9550
Epoch 3/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.1515 - accuracy: 0.953
```


Applications : Untitled1 - Jupyter Note...
Sat 25 Jan, 05:23 labuser
New Tab
Home Page - Select
AILAB3Question1 -
Untitled1 - Jupyter
localhost:8892/notebooks/Untitled1.ipynb?kernel_name=python3
Relaunch to update
jupyter Untitled1 Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Jupyter notebook
Google Chrome
Readme
Terminal
LibreOffice
Search
72°F Haze
ENG US
10:52 AM 1/25/2025

```
None
Epoch 1/10
1050/1050 [==============================] - 8s 5ms/step - loss: 0.4131 - accuracy: 0.873
7 - val_loss: 0.1921 - val_accuracy: 0.9417
Epoch 2/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.1941 - accuracy: 0.942
4 - val_loss: 0.1408 - val_accuracy: 0.9550
Epoch 3/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.1515 - accuracy: 0.953
0 - val_loss: 0.1267 - val_accuracy: 0.9598
Epoch 4/10
1050/1050 [==============================] - 6s 5ms/step - loss: 0.1228 - accuracy: 0.962
3 - val_loss: 0.1064 - val_accuracy: 0.9687
Epoch 5/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.1070 - accuracy: 0.967
2 - val_loss: 0.1000 - val_accuracy: 0.9702
```

```
In [ ]:
```

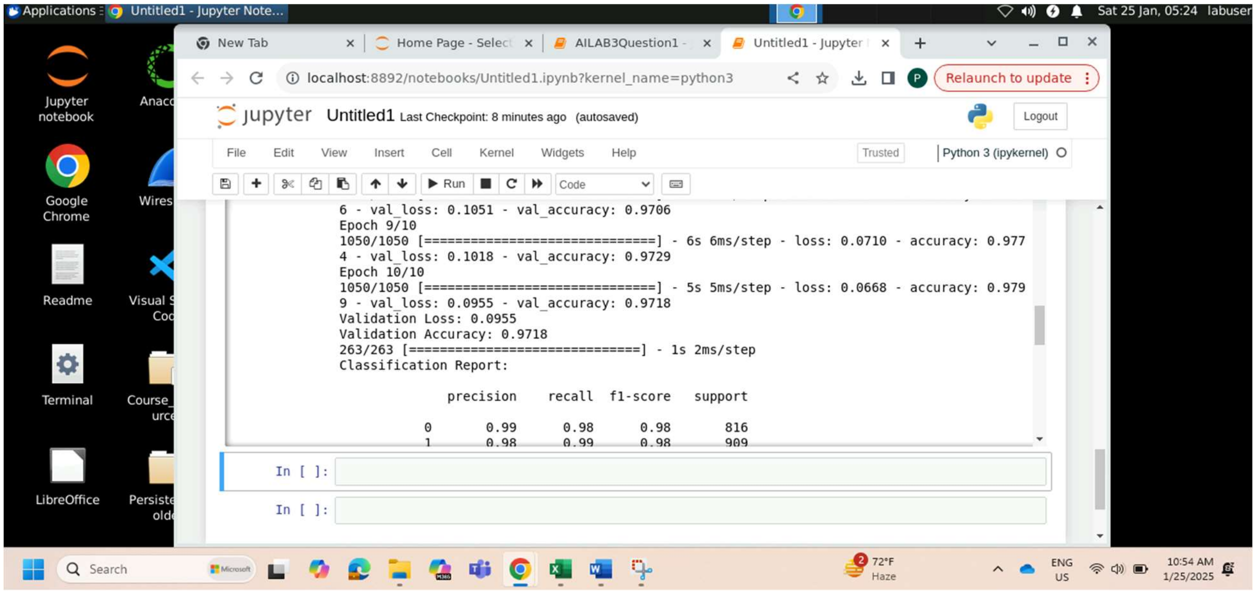
```
In [ ]:
```


Applications : Untitled1 - Jupyter Note...
Sat 25 Jan, 05:23 labuser
New Tab
Home Page - Select
AILAB3Question1 -
Untitled1 - Jupyter
localhost:8892/notebooks/Untitled1.ipynb?kernel_name=python3
Relaunch to update
jupyter Untitled1 Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Jupyter notebook
Google Chrome
Readme
Terminal
LibreOffice
Search
72°F
ENG
10:53 AM

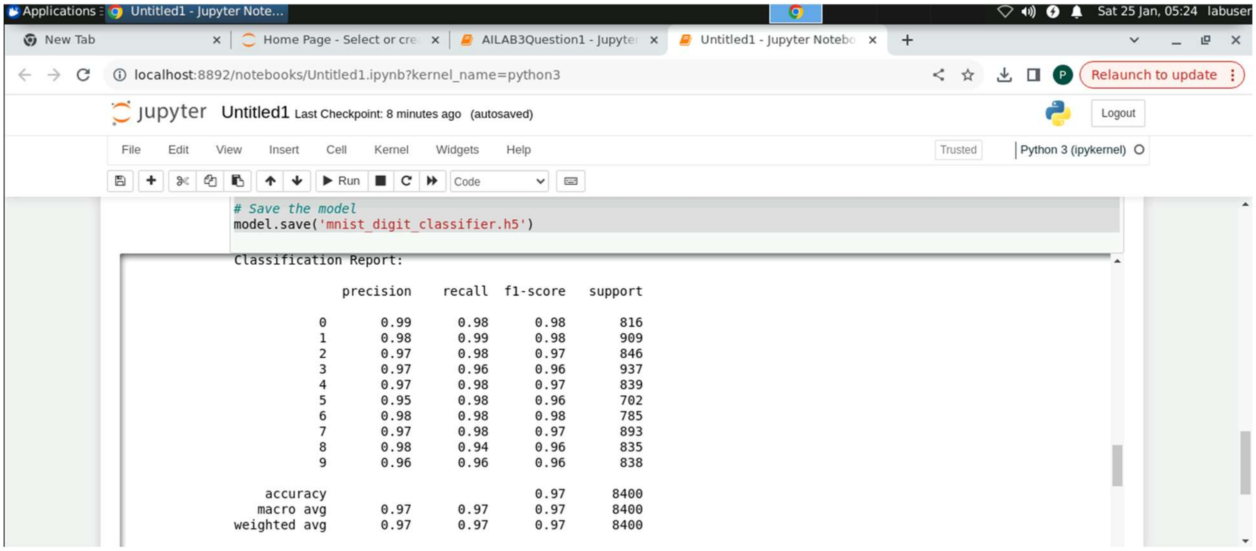
```
Epoch 4/10
1050/1050 [==============================] - 6s 5ms/step - loss: 0.1228 - accuracy: 0.962
3 - val_loss: 0.1064 - val_accuracy: 0.9687
Epoch 5/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.1070 - accuracy: 0.967
2 - val_loss: 0.1000 - val_accuracy: 0.9702
Epoch 6/10
1050/1050 [==============================] - 6s 5ms/step - loss: 0.0980 - accuracy: 0.969
1 - val_loss: 0.0966 - val_accuracy: 0.9707
Epoch 7/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0849 - accuracy: 0.973
1 - val_loss: 0.0962 - val_accuracy: 0.9724
Epoch 8/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0759 - accuracy: 0.975
6 - val_loss: 0.1051 - val_accuracy: 0.9706
Epoch 9/10
```

```
In [ ]:
```

```
In [ ]:
```

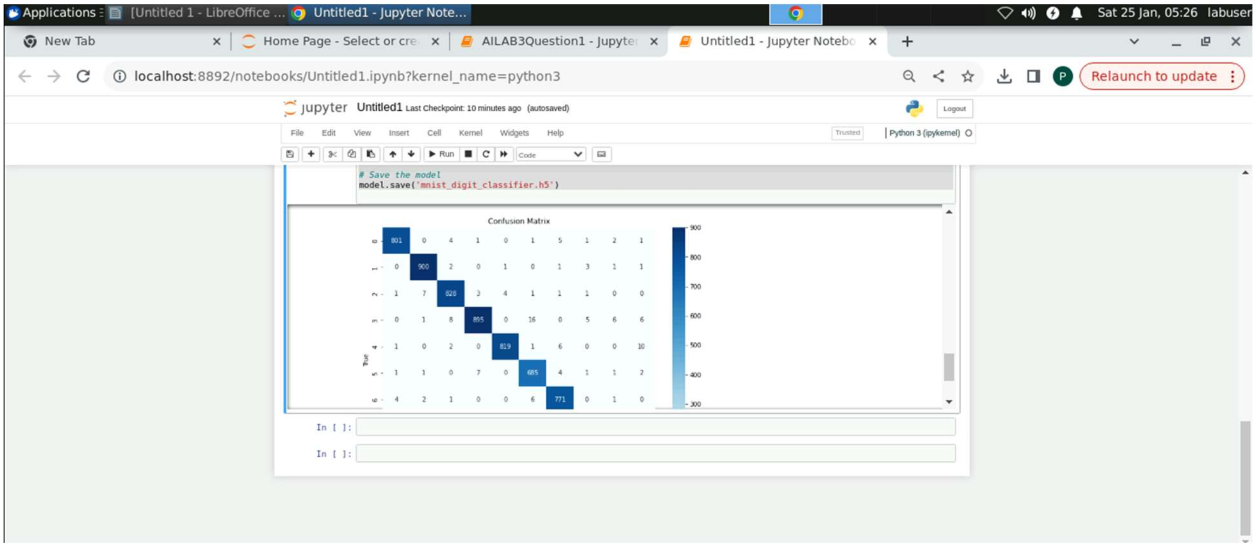

Untitled1 Last Checkpoint: 8 minutes ago (autosaved)
localhost:8892/notebooks/Untitled1.ipynb?kernel_name=python3
Python 3 (ipykernel)

```
6 - val_loss: 0.1051 - val_accuracy: 0.9706
Epoch 9/10
1050/1050 [==============================] - 6s 6ms/step - loss: 0.0710 - accuracy: 0.977
4 - val_loss: 0.1018 - val_accuracy: 0.9729
Epoch 10/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0668 - accuracy: 0.979
9 - val_loss: 0.0955 - val_accuracy: 0.9718
Validation Loss: 0.0955
Validation Accuracy: 0.9718
263/263 [==============================] - 1s 2ms/step
Classification Report:

              precision    recall  f1-score   support

           0       0.99      0.98      0.98       816
           1       0.98      0.99      0.98       909
```

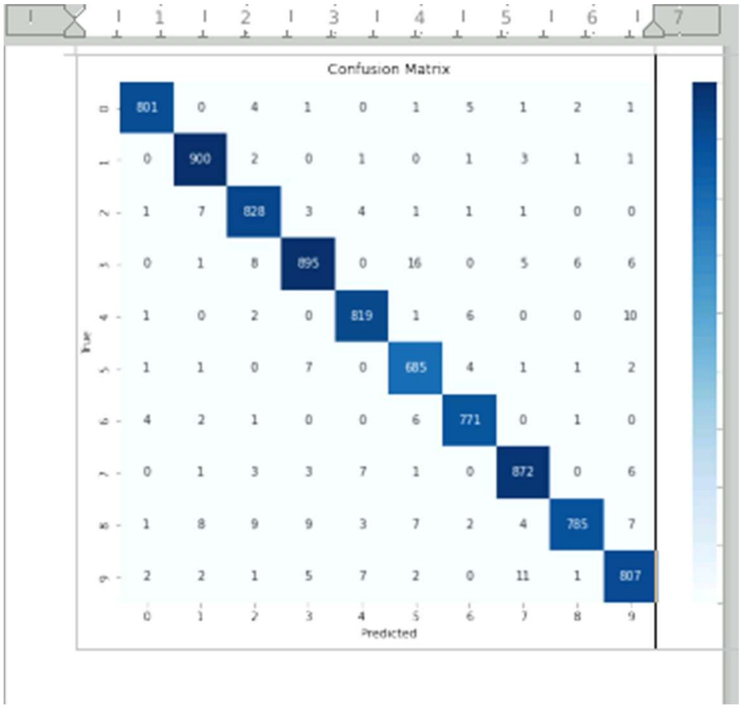

Untitled1 Last Checkpoint: 8 minutes ago (autosaved)
localhost:8892/notebooks/Untitled1.ipynb?kernel_name=python3
Python 3 (ipykernel)

```
# Save the model
model.save('mnist_digit_classifier.h5')
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.99      0.98      0.98       816
           1       0.98      0.99      0.98       909
           2       0.97      0.98      0.97       846
           3       0.97      0.96      0.96       937
           4       0.97      0.98      0.97       839
           5       0.95      0.98      0.96       702
           6       0.98      0.98      0.98       785
           7       0.97      0.98      0.97       893
           8       0.98      0.94      0.96       835
           9       0.96      0.96      0.96       838

    accuracy                           0.97      8400
   macro avg       0.97      0.97      0.97      8400
weighted avg       0.97      0.97      0.97      8400
```


Applications
[Untitled 1 - LibreOffice ...
Untitled1 - Jupyter Note...
Sat 25 Jan, 05:26 labuser
New Tab
Home Page - Select or cre
AILAB3Question1 - Jupyter
Untitled1 - Jupyter Notebo
localhost:8892/notebooks/Untitled1.ipynb?kernel_name=python3
Relaunch to update
jupyter Untitled1 Last Checkpoint: 10 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [ ]:
In [ ]:

```
# Save the model
model.save('mnist_digit_classifier.h5')
```



	0	1	2	3	4	5	6	7	8	9
0	801	0	4	1	0	1	5	1	2	1
1	0	900	2	0	1	0	1	3	1	1
2	1	7	828	3	4	1	1	1	0	0
3	0	1	8	895	0	16	0	5	6	6
4	1	0	2	0	819	1	6	0	0	10
5	1	1	0	7	0	685	4	1	1	2
6	4	2	1	0	0	6	771	0	1	0
7	0	1	3	3	7	1	0	872	0	6
8	1	8	9	9	3	7	2	4	785	7
9	2	2	1	5	7	2	0	11	1	807

[C] Problem Statement: Given a sentence "The boy eats an apple" and a defined context-free grammar (CFG), the task is to:

1. Construct the **parse tree** for the sentence based on the grammar.
2. Calculate the **probabilities** for each production rule used in the parse tree.

Goal:

- To accurately **parse** the sentence "The boy eats an apple" using the provided context-free grammar.
- To **calculate the probability** of generating the sentence based on the production rules of the grammar.
- To understand how different parts of the sentence contribute to the structure and how their respective probabilities are determined by the grammar.

Challenge:

- Properly applying the **context-free grammar** rules to generate the correct structure for the sentence.
- Calculating the **probabilities** based on the grammar, which involves multiplying the probabilities of the individual rules used in the parse tree.
- Ensuring that the **recursive nature of parsing** is handled correctly, with proper tracking of intermediate steps for correct calculation of probabilities.

Input:

- A sentence: **"The boy eats an apple"**
- A context-free grammar (CFG) for constructing the parse tree:

S → NP VP (1)

NP → Det N (2)

VP → V NP (3)

Det → 'The' | 'an' (4)

N → 'boy' | 'apple' (5)

V → 'eats' (6)

Task:

1. **Construct the Parse Tree**: Use the context-free grammar to generate the correct parse tree for the given sentence.
2. **Calculate Probabilities**: For each production rule used in the parse tree, calculate the probability based on the defined probabilities for the grammar rules.
3. **Multiply the probabilities** of each rule used to determine the overall probability of generating the sentence.

Expected Output:

1. **Parse Tree** for the sentence "The boy eats an apple":

```
         S
       /    \
     NP      VP
   /   \   /  \
 Det    N  V     NP
  |     | |   /   \
 The   boy eats Det     N
                |      |
               an    apple
```

2. **Total Probability** of the sentence "The boy eats an apple": 0.0625

Ans :-

Given CFG Rules

1. **S → NP VP**
2. **NP → Det N**
3. **VP → V NP**
4. **Det → 'The' | 'an'**
5. **N → 'boy' | 'apple'**
6. **V → 'eats'**

Assumed Probabilities for Grammar Rules

To calculate probabilities, we assign equal probabilities to all alternatives of a non-terminal symbol unless otherwise specified. For simplicity, assume the probabilities for each production rule are:

1. **S → NP VP**: P=1.0P = 1.0P=1.0 (since there is only one way to expand SSS).
2. **NP → Det N**: P=1.0P = 1.0P=1.0 (only one way to expand NPNPNP).

3. **VP → V NP**: P=1.0P = 1.0P=1.0.
4. **Det → 'The'**: P=0.5P = 0.5P=0.5, **Det → 'an'**: P=0.5P = 0.5P=0.5.
5. **N → 'boy'**: P=0.5P = 0.5P=0.5, **N → 'apple'**: P=0.5P = 0.5P=0.5.
6. **V → 'eats'**: P=1.0P = 1.0P=1.0 (only one option).

Step 1: Constructing the Parse Tree

The parse tree represents how the sentence is derived from the grammar:

mathematica

CopyEdit

```
     S
    / \
   NP    VP
  / \  / \
 Det  N V   NP
  |   | |  / \
 The  boy eats Det  N
          |   |
         an  apple
```

- Start from SSS: Expand S→NPVPS → NP VPS→NPVP.
- Expand NP→DetNNP → Det NNP→DetN for both NPNPNP instances.
- Expand Det→’The’Det → 'The'Det→’The’, Det→’an’Det → 'an'Det→’an’, N→’boy’N → 'boy'N→’boy’, N→’apple’N → 'apple'N→’apple’, and V→’eats’V → 'eats'V→’eats’.

Step 2: Calculating Probabilities

Rules Used in the Parse Tree:

1. **S → NP VP**: P=1.0P = 1.0P=1.0
2. **NP → Det N** (for the first NPNPNP): P=1.0P = 1.0P=1.0
3. **Det → 'The'**: P=0.5P = 0.5P=0.5
4. **N → 'boy'**: P=0.5P = 0.5P=0.5
5. **VP → V NP**: P=1.0P = 1.0P=1.0
6. **V → 'eats'**: P=1.0P = 1.0P=1.0
7. **NP → Det N** (for the second NPNPNP): P=1.0P = 1.0P=1.0
8. **Det → 'an'**: P=0.5P = 0.5P=0.5
9. **N → 'apple'**: P=0.5P = 0.5P=0.5

Calculating Total Probability:

The total probability is the product of the probabilities of all rules used:

Step 3: Output

1. **Parse Tree**:

mathematica

CopyEdit

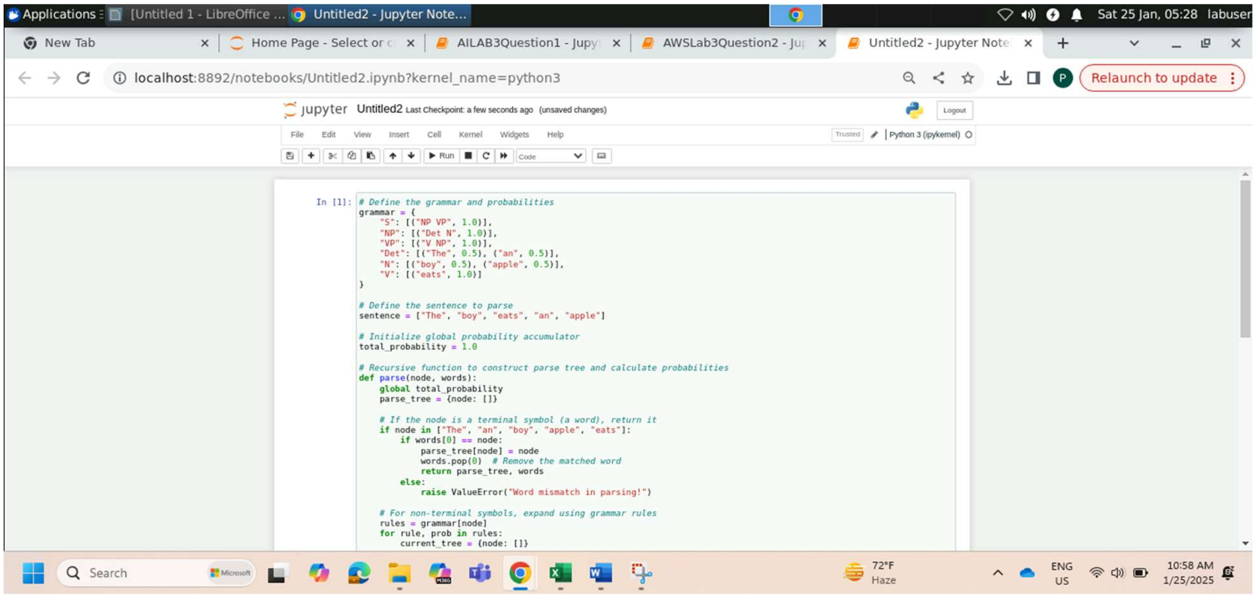
```
        S
      /  \
    NP     VP
  /  \   /  \
 Det    N  V     NP
  |     |  |    /  \
 The   boy eats Det   N
```

| |

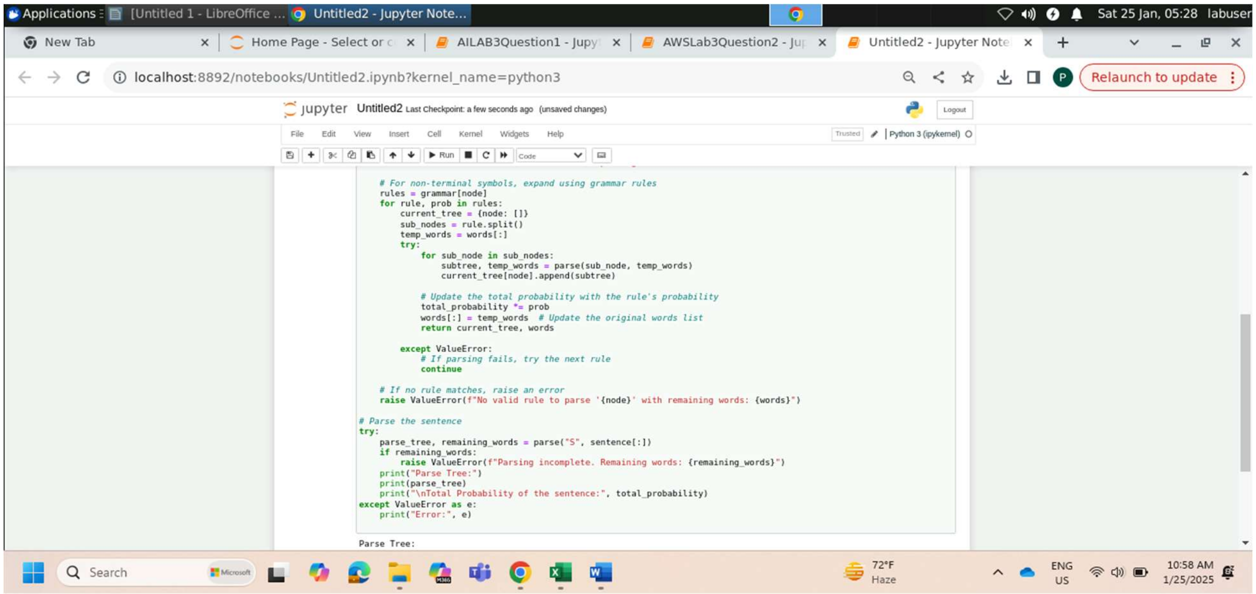
an   apple

2. **Total Probability**:

Total Probability of the sentence "The boy eats an apple"=0.0625\text{Total Probability of the sentence "The boy eats an apple"} = 0.0625Total Probability of the sentence "The boy eats an apple"=0.0625

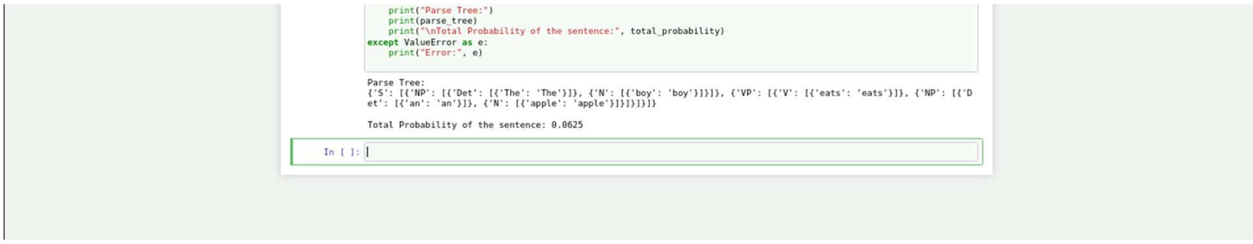

Applications
[Untitled 1 - LibreOffice ...
Untitled2 - Jupyter Note...
Sat 25 Jan, 05:28 labuser
New Tab
Home Page - Select or c
AILAB3Question1 - Jupy
AWSLab3Question2 - Ju
Untitled2 - Jupyter Note
localhost:8892/notebooks/Untitled2.ipynb?kernel_name=python3
Relaunch to update
jupyter Untitled2 Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Search
72°F
Haze
ENG
US
10:58 AM
1/25/2025

```
In [1]: # Define the grammar and probabilities
        grammar = {
            "S": [("NP VP", 1.0)],
            "NP": [("Det N", 1.0)],
            "VP": [("V NP", 1.0)],
            "Det": [("The", 0.5), ("an", 0.5)],
            "N": [("boy", 0.5), ("apple", 0.5)],
            "V": [("eats", 1.0)]
        }

        # Define the sentence to parse
        sentence = ["The", "boy", "eats", "an", "apple"]

        # Initialize global probability accumulator
        total_probability = 1.0

        # Recursive function to construct parse tree and calculate probabilities
        def parse(node, words):
            global total_probability
            parse_tree = {node: []}

            # If the node is a terminal symbol (a word), return it
            if node in ["The", "an", "boy", "apple", "eats"]:
                if words[0] == node:
                    parse_tree[node] = node
                    words.pop(0)  # Remove the matched word
                    return parse_tree, words
                else:
                    raise ValueError("Word mismatch in parsing!")

            # For non-terminal symbols, expand using grammar rules
            rules = grammar[node]
            for rule, prob in rules:
                current_tree = {node: []}
```


Applications
[Untitled 1 - LibreOffice ...
Untitled2 - Jupyter Note...
Sat 25 Jan, 05:28 labuser
New Tab
Home Page - Select or c
AILAB3Question1 - Jupy
AWSLab3Question2 - Ju
Untitled2 - Jupyter Note
localhost:8892/notebooks/Untitled2.ipynb?kernel_name=python3
Relaunch to update
jupyter Untitled2 Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Search
72°F
Haze
ENG
US
10:58 AM
1/25/2025

```
    # For non-terminal symbols, expand using grammar rules
    rules = grammar[node]
    for rule, prob in rules:
        current_tree = {node: []}
        sub_nodes = rule.split()
        temp_words = words[:]
        try:
            for sub_node in sub_nodes:
                subtree, temp_words = parse(sub_node, temp_words)
                current_tree[node].append(subtree)

            # Update the total probability with the rule's probability
            total_probability *= prob
            words[:] = temp_words  # Update the original words list
            return current_tree, words

        except ValueError:
            # If parsing fails, try the next rule
            continue

    # If no rule matches, raise an error
    raise ValueError(f"No valid rule to parse '{node}' with remaining words: {words}")

# Parse the sentence
try:
    parse_tree, remaining_words = parse("S", sentence[:])
    if remaining_words:
        raise ValueError(f"Parsing incomplete. Remaining words: {remaining_words}")
    print("Parse Tree:")
    print(parse_tree)
    print("\nTotal Probability of the sentence:", total_probability)
except ValueError as e:
    print("Error:", e)

Parse Tree:
```

Output:-



```
    print("Parse Tree:")
    print(parse_tree)
    print("\nTotal Probability of the sentence:", total_probability)
except ValueError as e:
    print("Error:", e)
```

```
Parse Tree:
{'S': [{'NP': [{'Det': [{'The': 'The'}]}, {'N': [{'boy': 'boy'}]}]}, {'VP': [{'V': [{'eats': 'eats'}]}, {'NP': [{'D
et': [{'an': 'an'}]}, {'N': [{'apple': 'apple'}]}]}]}]}

Total Probability of the sentence: 0.0625
```