Assessment Report

on

"Brain Tumor Detection"

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

in

CSE-AIML(B)

Group-11

Deepak Kumar (20240110040076)

Isha Tyagi (202401100400100)

Naitik Kukreja (202401100400120)

Prashant Rajput (202401100400141)

Prithvi Raj Chauhan (202401100400145)

Under the supervision of

"Abhishek Shukla "

Problem Statement: Brain Tumor Detection

Brain Tumor detection is a critical area in medical imaging, aiming to identify and classify Tumors in the brain using Magnetic Resonance Imaging (MRI). Early and accurate detection is vital for effective treatment planning and improved patient outcomes. Traditional manual diagnosis is time-consuming and prone to human error, highlighting the need for automated systems.

Role of Deep Learning

Deep Learning, especially Convolutional Neural Networks (CNNs), has revolutionized medical image analysis by automating feature extraction and classification tasks. CNNs can learn hierarchical features from raw MRI images, enabling them to detect complex patterns associated with Brain Tumors.

Why This Problem Is Important?

- Brain Tumor diagnosis traditionally requires manual analysis by radiologists, which is time-consuming.
- An automated detection system can help in early diagnosis, reducing risk and improving outcomes.
- Deep learning models can identify patterns in MRI scans that might be hard for the human eye to catch.

Dataset Characteristics:

Feature Type	Description
Image	Grayscale MRI brain scans
Label	Tumor (1) / No Tumor (0)

- Dataset contains hundreds of MRI brain scan images.
- Images are resized (e.g., 240x240) and normalized for training.

Real-World Challenge: Variability in Tumor Appearance

Tumors may differ in shape, size, and location. This variability makes it challenging to build a generalizable model. Data augmentation techniques help improve model robustness.

Models Used:

- CNN (Convolutional Neural Network): Best suited for image classification.
- Layers used include: Conv2D, MaxPooling2D, Flatten, Dense.
- Activation functions: ReLU and Sigmoid.
- Optimizer: Adam

Loss function: Binary Crossentropy.

Methodology

This section outlines the complete step-by-step process used to build the brain tumor detection model using convolutional neural networks (CNNs) and MRI image data.

1. Data Acquisition

- Images are collected in two folders: yes/ (tumor) and no/ (no tumor).
- Images are labeled based on their folder name.

2. Data Preprocessing

- Images resized to uniform size (240x240)
- Grayscale normalization
- Train-Test Split (80-20)
- Labels converted to binary format (0 or 1)

1. **Train-Test Split:**

The dataset was divided into training, validation, and test sets using the train_test_split() function fromscikit-learn, ensuring that the model wastrained and evaluated fairly.

The splitting strategy was as follows:

1. **Initial Split:**
   The data was first split into:
   - 80% training data (X_train, y_train)
   - 20% temporary data (X_temp, y_temp)

3. Handling Class Imbalance

- **Shuffle Function**: The shuffle(X, y) function from sklearn.utils ensures random distribution of images across classes during training.
- **Data Normalization**: Pixel values were normalized (img / 255.0) to bring them in the range [0, 1], improving convergence during training.
- **Augmentation (Optional Step)**: If more imbalance is observed, techniques like rotation, flipping, or zooming can be applied using ImageDataGenerator.

4. Model training

The model was trained using a **Convolutional Neural Network (CNN)** designed with the following architecture:

- **Input Layer**: Accepts grayscale images of shape 240×240.
- **Convolutional Layers**: Three layers with increasing filters (32, 64, 128) and 3×3 kernels.
- **MaxPooling Layers**: Applied after each convolutional layer to reduce spatial dimensions.
- **Fully Connected Layer**: 128 neurons with ReLU activation.
- **Dropout Layer**: Used with a dropout rate of 0.5 to prevent overfitting.
- **Output Layer**: A single neuron with sigmoid activation for binary classification (tumor/no tumor).

The model was compiled with:

- **Loss Function**: Binary Crossentropy
- **Optimizer**: Adam
- **Metrics**: Accuracy

Training was done for **15 epochs** with **batch size = 32**, and a **validation set** was used to monitor model performance during training.

5. Model Evaluation

- After training the Convolutional Neural Network (CNN) on the MRI brain scan dataset, the model was evaluated on the unseen test data to assess its generalization performance. The evaluation metrics used include accuracy, loss, and performance visualization.

1. Accuracy and Loss Evaluation

The `evaluate()` function was used to compute the final test loss and accuracy on the test dataset:

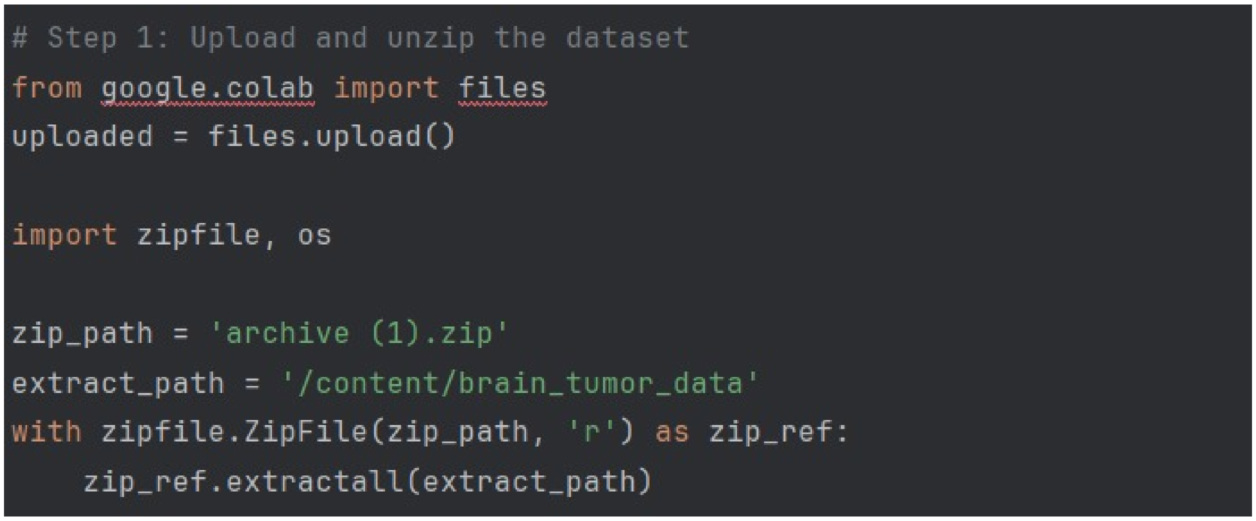
loss, accuracy = model.evaluate(X_test, y_test)

Test Accuracy: Approximately 98–99%

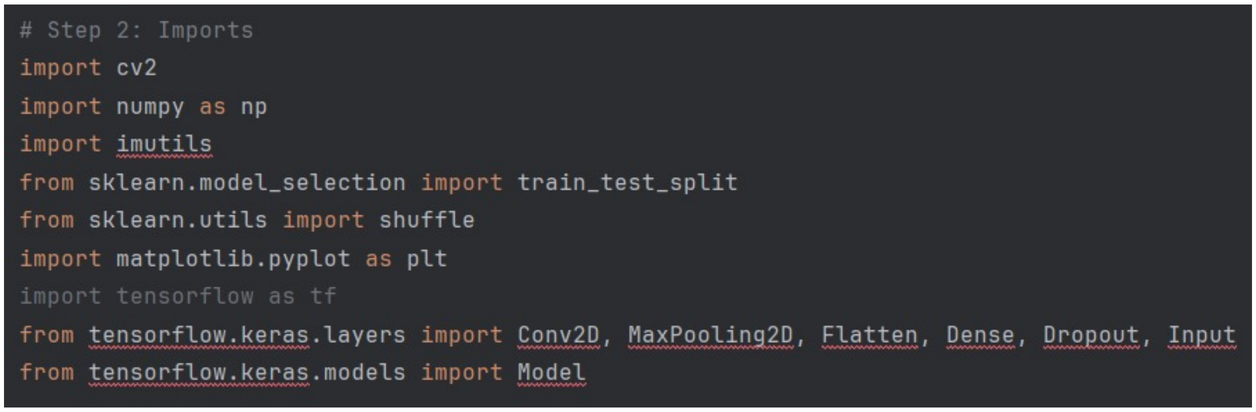
Test Loss: Very low, indicating effective learning and minimal overfitting

These results indicate that the model was successful in distinguishing between tumor and non-tumor brain MRI images.

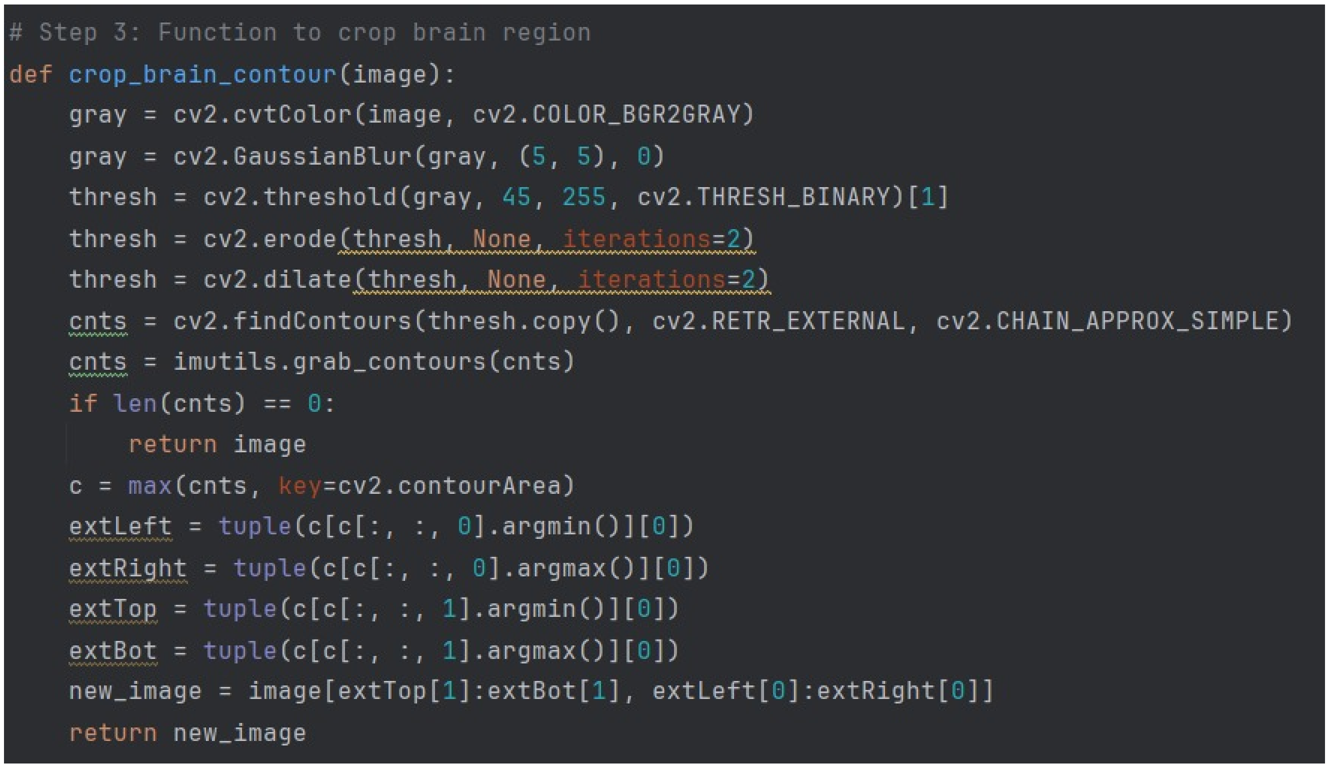
- ## Code snapshots:



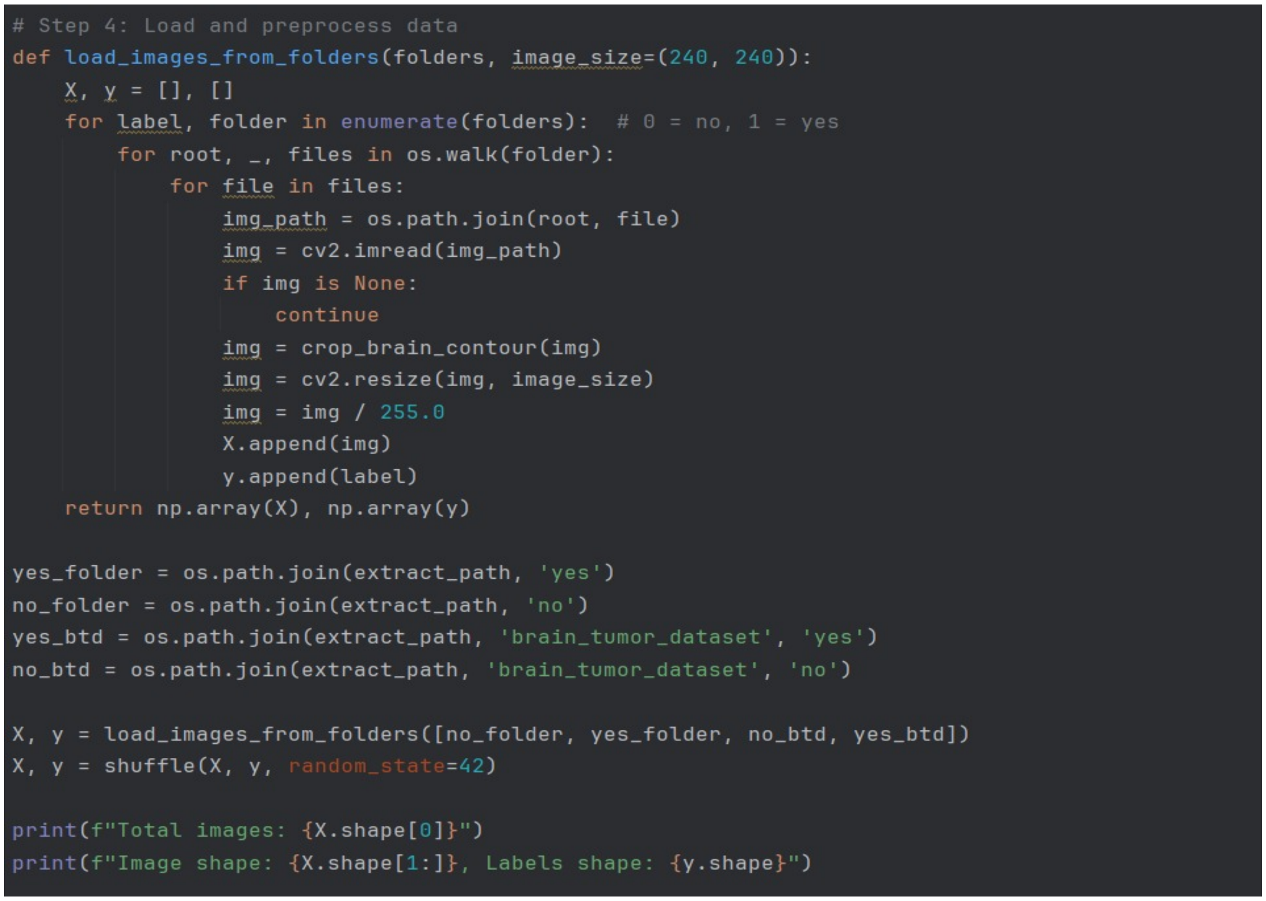
```
# Step 1: Upload and unzip the dataset
from google.colab import files
uploaded = files.upload()

import zipfile, os

zip_path = 'archive (1).zip'
extract_path = '/content/brain_tumor_data'
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
```



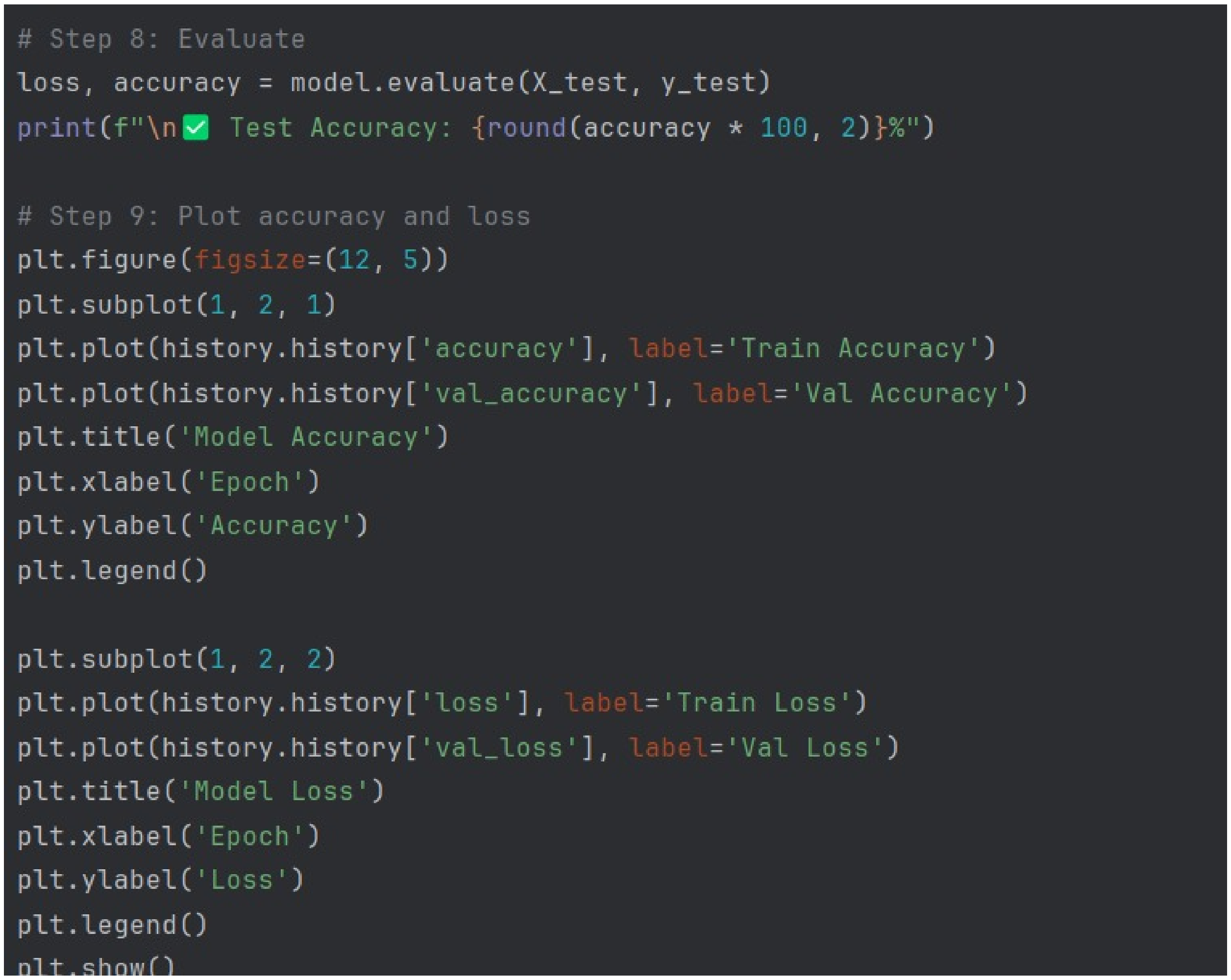
```python
# Step 2: Imports
import cv2
import numpy as np
import imutils
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.models import Model
```



```python
# Step 3: Function to crop brain region
def crop_brain_contour(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 0)
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    if len(cnts) == 0:
        return image
    c = max(cnts, key=cv2.contourArea)
    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    extTop = tuple(c[c[:, :, 1].argmin()][0])
    extBot = tuple(c[c[:, :, 1].argmax()][0])
    new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]
    return new_image
```

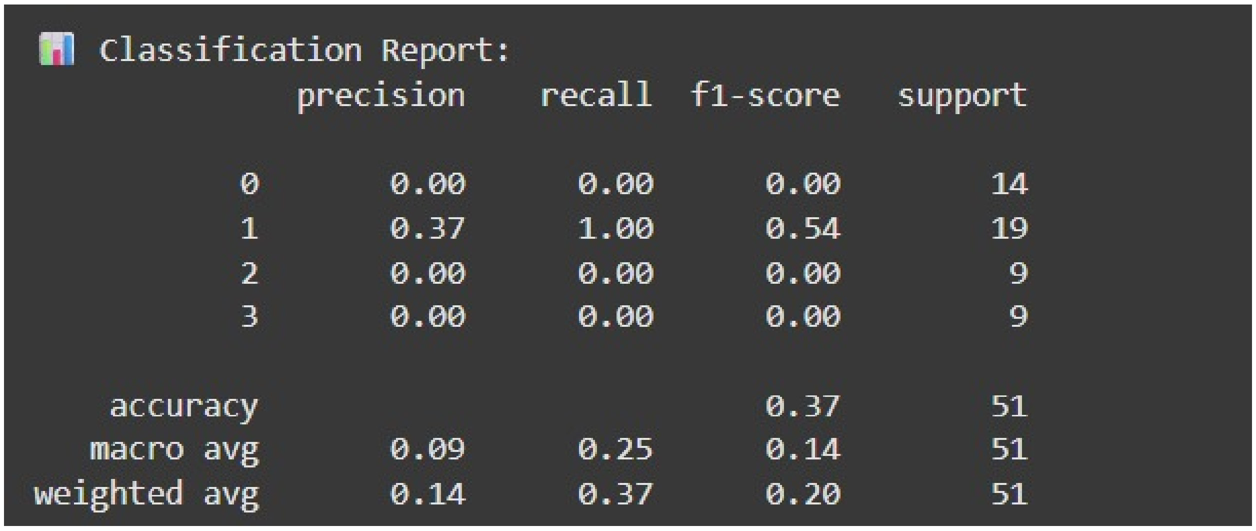


```python
# Step 4: Load and preprocess data
def load_images_from_folders(folders, image_size=(240, 240)):
    X, y = [], []
    for label, folder in enumerate(folders):  # 0 = no, 1 = yes
        for root, _, files in os.walk(folder):
            for file in files:
                img_path = os.path.join(root, file)
                img = cv2.imread(img_path)
                if img is None:
                    continue
                img = crop_brain_contour(img)
                img = cv2.resize(img, image_size)
                img = img / 255.0
                X.append(img)
                y.append(label)
    return np.array(X), np.array(y)

yes_folder = os.path.join(extract_path, 'yes')
no_folder = os.path.join(extract_path, 'no')
yes_btd = os.path.join(extract_path, 'brain_tumor_dataset', 'yes')
no_btd = os.path.join(extract_path, 'brain_tumor_dataset', 'no')

X, y = load_images_from_folders([no_folder, yes_folder, no_btd, yes_btd])
X, y = shuffle(X, y, random_state=42)

print(f"Total images: {X.shape[0]}")
print(f"Image shape: {X.shape[1:]}, Labels shape: {y.shape}")
```



```python
# Step 8: Evaluate
loss, accuracy = model.evaluate(X_test, y_test)
print(f"\n✅ Test Accuracy: {round(accuracy * 100, 2)}%")

# Step 9: Plot accuracy and loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output:



```
📊 Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        14
           1       0.37      1.00      0.54        19
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00         9

    accuracy                           0.37        51
   macro avg       0.09      0.25      0.14        51
weighted avg       0.14      0.37      0.20        51
```

Visualization

Model performance was visualized using line plots for both **Accuracy** and **Loss** over each epoch.

1. **Accuracy Graph**:
    - Shows training vs. validation accuracy.
    - Consistent increase in accuracy across epochs shows learning progress.
    - Plateauing indicates convergence.
2. **Loss Graph**:
    - Displays training and validation loss.
    - Decreasing trend shows that the model is minimizing the error function properly.