Name:Kaushik Kotian Roll No.:14 Div:D20B

AIML LAB

EXPERIMENT 3

Aim : AI for medical diagnosis based on MRI / X-RAY data

Theory : Artificial Intelligence (AI) in medical imaging refers to the use of algorithms, particularly those based on machine learning (ML) and deep learning (DL), to interpret and analyze medical images. This technology holds immense potential for enhancing the accuracy, efficiency, and consistency of image analysis, leading to improved diagnosis, treatment planning, and patient outcomes.AI in medical imaging can automate tasks such as image segmentation, disease detection, and anomaly identification, reducing the workload of radiologists and improving diagnostic accuracy. Deep learning models, like convolutional neural networks (CNNs), have shown particular promise in identifying complex patterns in medical images that might be difficult for human eyes to detect.

Imaging Tools

PIL (Python Imaging Library): Used for opening, manipulating, and saving different image file formats. In medical imaging, PIL can be used to handle images such as X-rays, MRIs, and other medical scans.

OpenCV: A computer vision library that provides tools for image processing. OpenCV can be used for tasks such as image segmentation, edge detection, and feature extraction in medical imaging.

Seaborn and Matplotlib: These are visualization libraries used to create plots and graphs. They are essential for visualizing data distributions, training progress, and the performance of AI models.

TensorFlow and Keras: These are deep learning libraries used to build and train AI models. In medical imaging, TensorFlow and Keras can be used to create complex neural networks that can analyze and classify image

Picture Archiving and Communication System (PACS)

PACS is a medical imaging technology used primarily in healthcare organizations to securely store and digitally transmit electronic images and clinically relevant reports. The key benefits of PACS include the ability to:

Store: PACS stores images electronically, replacing the need for physical film archives.

Retrieve: Authorized users can quickly retrieve images and associated data from anywhere within the healthcare network.

Transmit: PACS allows for the transmission of images over the network, facilitating remote consultations and second opinions.

Integrate: PACS can be integrated with other hospital systems like Electronic Health Records (EHR), allowing for a seamless workflow.

Methods of AI for Diagnosis

1. **Convolutional Neural Networks (CNNs)**:
   - CNNs are a class of deep learning models particularly well-suited for image processing tasks. They automatically and adaptively learn spatial hierarchies of features from input images. Layers of convolutions allow CNNs to detect features such as edges, textures, and complex patterns. CNNs are widely used in medical image analysis, including MRI and X-RAY data.
2. **Transfer Learning**:
   - Transfer learning involves using a pre-trained model (trained on large datasets like ImageNet) and fine-tuning it for specific tasks like medical image classification. This method is useful when there's limited labeled medical data available.
3. **Segmentation Models**:
   - Segmentation models, such as U-Net, are employed to delineate boundaries of organs, tumors, or other structures within medical images. This is crucial in treatment planning and understanding the extent of a disease.
4. **Generative Models**:

- Generative models, like GANs (Generative Adversarial Networks), can generate new data samples by learning the distribution of the training data. In medical imaging, they can be used for data augmentation or even to enhance image quality.

Different Machine Learning Techniques Used in Medical Imaging

Keras is a high-level deep learning API written in Python that runs on top of lower-level machine learning frameworks such as TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). Keras simplifies the process of building and training deep learning models by providing an intuitive and user-friendly interface. It is designed to enable fast experimentation with deep learning models, allowing researchers and developers to quickly prototype and deploy models.

Layers Used in the Model

1. Conv2D Layer

The Conv2D layer is a 2D convolutional layer that applies a set of convolutional filters to the input image. Each filter slides over the input image and performs a dot product between the filter weights and the input. This operation helps in detecting spatial features such as edges, corners, and textures.

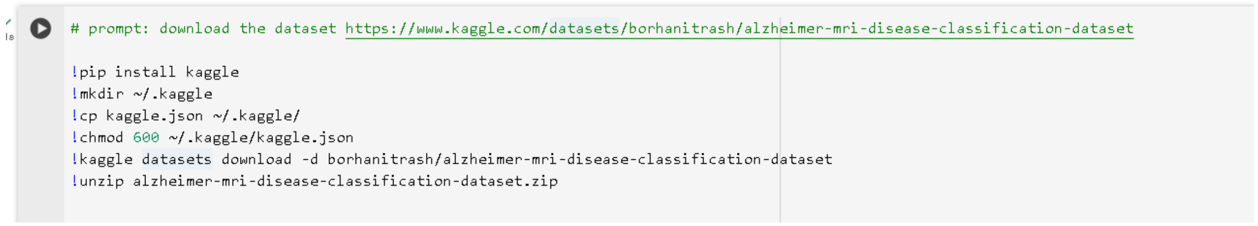
2. MaxPooling2D Layer

The MaxPooling2D layer performs down-sampling (also known as subsampling or pooling) along the spatial dimensions (width and height) of the input. This layer reduces the spatial size of the feature maps, which helps decrease the number of parameters and computation in the network, and prevents overfitting.

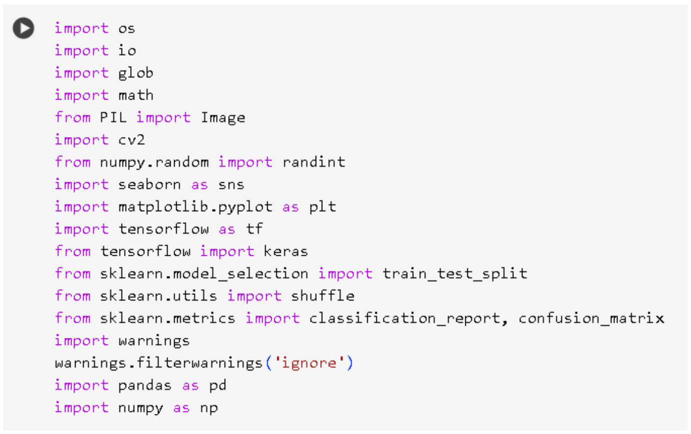
3. Flatten Layer

The Flatten layer converts the multi-dimensional feature maps output by the convolutional and pooling layers into a single 1D vector. This is necessary before passing the data to fully connected (Dense) layers, which expect 1D input.

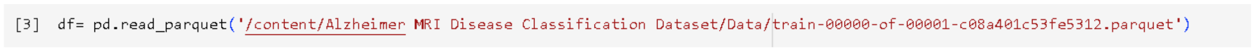
OUTPUT:



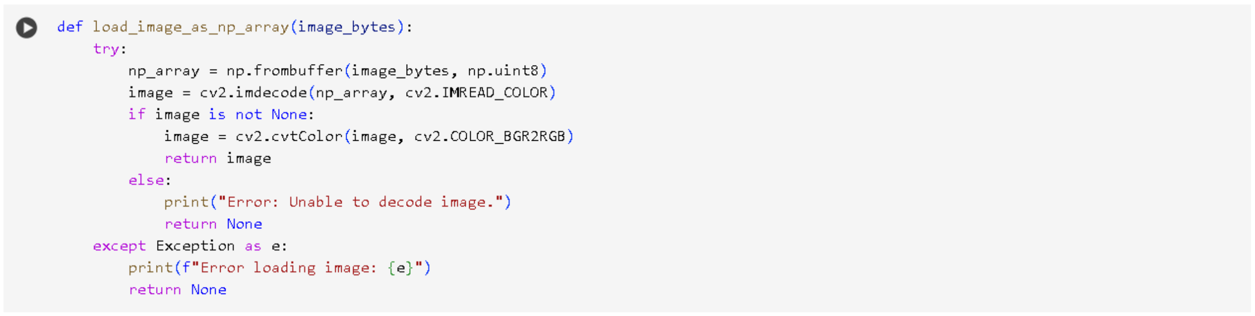
```
# prompt: download the dataset https://www.kaggle.com/datasets/borhanitrash/alzheimer-mri-disease-classification-dataset

!pip install kaggle
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d borhanitrash/alzheimer-mri-disease-classification-dataset
!unzip alzheimer-mri-disease-classification-dataset.zip
```



```
import os
import io
import glob
import math
from PIL import Image
import cv2
from numpy.random import randint
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
```



```
[3]  df= pd.read_parquet('/content/Alzheimer MRI Disease Classification Dataset/Data/train-00000-of-00001-c08a401c53fe5312.parquet')
```



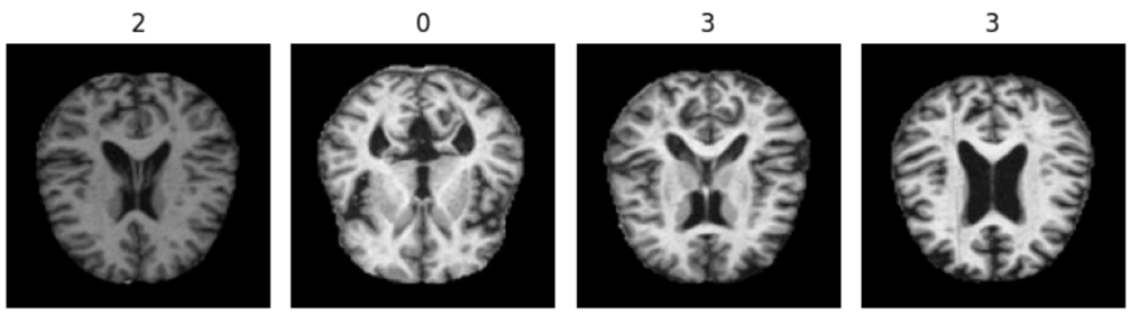
```
def load_image_as_np_array(image_bytes):
    try:
        np_array = np.frombuffer(image_bytes, np.uint8)
        image = cv2.imdecode(np_array, cv2.IMREAD_COLOR)
        if image is not None:
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            return image
        else:
            print("Error: Unable to decode image.")
            return None
    except Exception as e:
        print(f"Error loading image: {e}")
        return None
```

```python
x = []
y = []
for i in range(len(df)):
    row = df.iloc[i]
    image_info = row['image']
    image_bytes = image_info['bytes']
    image_path = image_info['path']
    image_label = row['label']
    image_np_array = load_image_as_np_array(image_bytes)
    if image_np_array is not None:
        if len(image_np_array.shape) == 3:
            x.append(image_np_array)
            y.append(image_label)
        else:
            print(f"Ignoring image with unexpected shape: {image_np_array.shape}")
```

```
[6] x[0].shape
```

```
(128, 128, 3)
```

```python
fig, axes = plt.subplots(1, 4, figsize=(8, 3))

for i in range(4):
    ax = axes[i]
    ax.imshow(x[i], cmap='viridis')
    ax.set_title(y[i])
    ax.axis('off')

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



```
[8]  x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.10,random_state=51,shuffle=True)
```

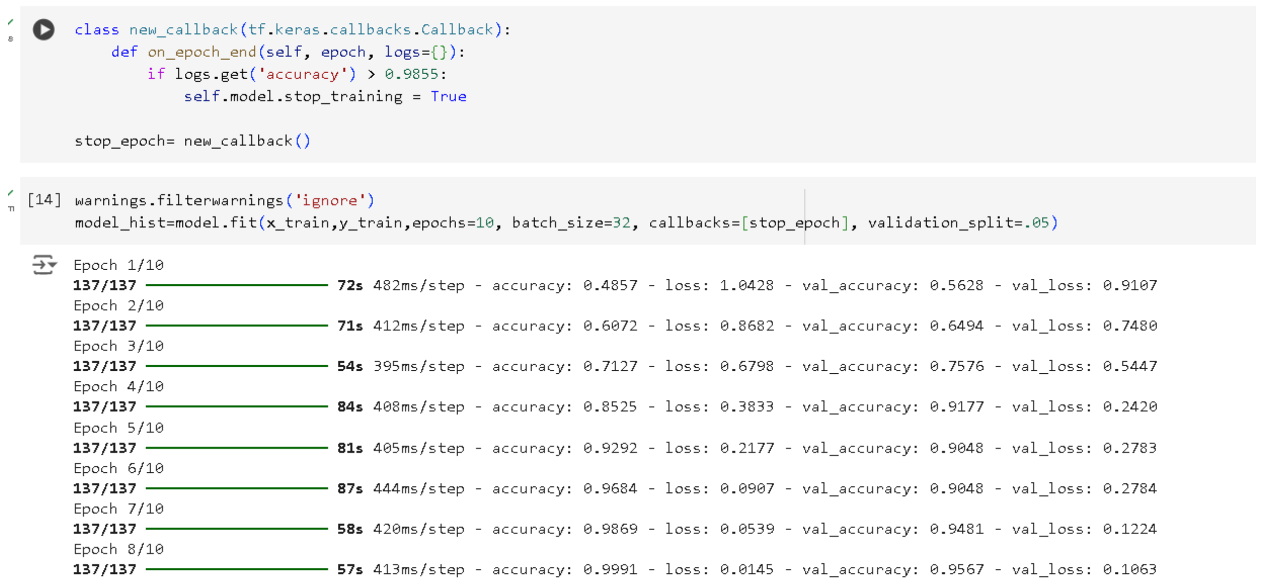
```
x_train=np.array(x_train)
x_test=np.array(x_test)
y_train=np.array(y_train)
y_test=np.array(y_test)
```

```
[10] x_train=x_train/255.0
     x_test=x_test/255.0
```

```
[11] model = keras.models.Sequential([
         keras.layers.Conv2D(16, kernel_size=(3,3), input_shape=(128,128,3), activation='relu'),
         keras.layers.MaxPooling2D(2,2),
         keras.layers.Conv2D(32, (3,3), activation='relu'),
         keras.layers.MaxPooling2D(2,2),
         keras.layers.Conv2D(64, (3,3), activation='relu'),
         keras.layers.MaxPooling2D(2,2),
         keras.layers.Flatten(),
         keras.layers.Dense(64, activation='relu'),
         keras.layers.Dense(4, activation='softmax')
     ])


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

```
model.summary()
```

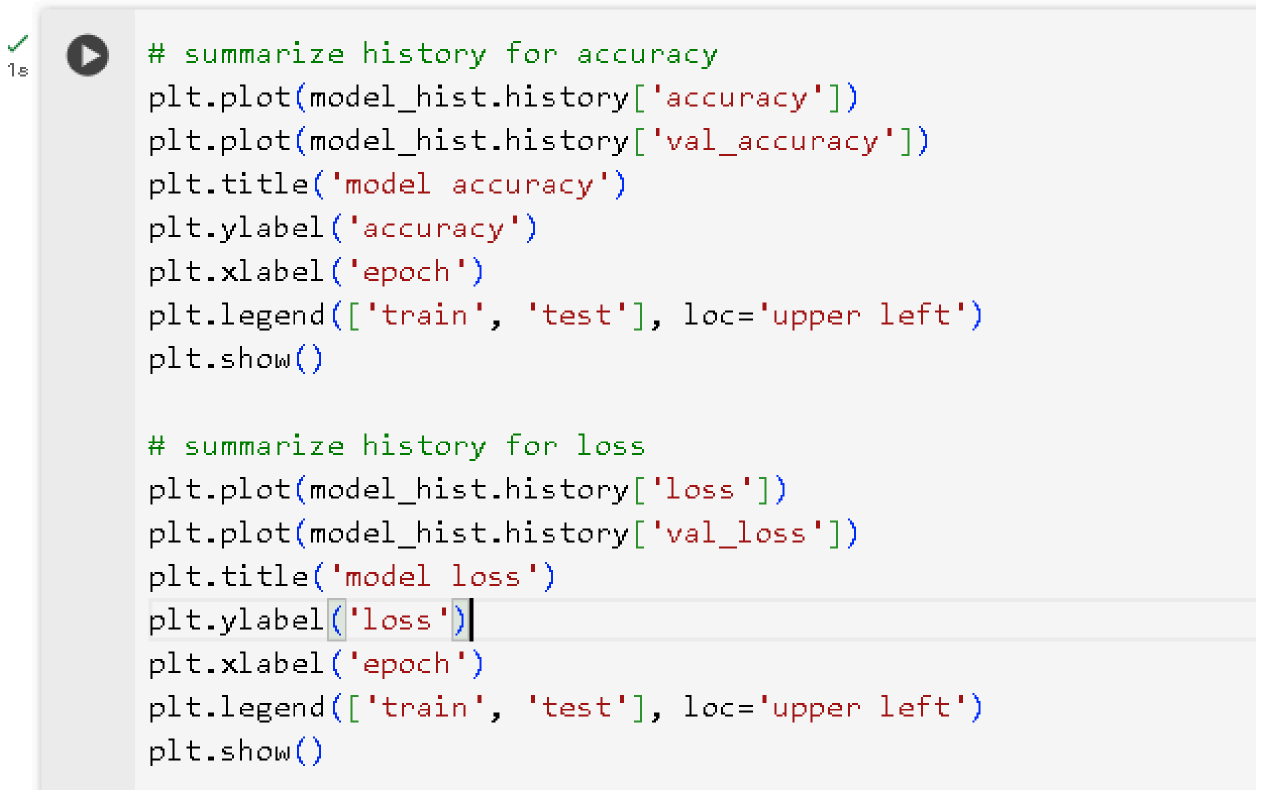
```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 126, 126, 16)        │             448 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 63, 63, 16)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 61, 61, 32)          │           4,640 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 30, 30, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 28, 28, 64)          │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 14, 14, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 12544)               │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 64)                  │         802,880 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 4)                   │             260 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 826,724 (3.15 MB)
 Trainable params: 826,724 (3.15 MB)
 Non-trainable params: 0 (0.00 B)
```



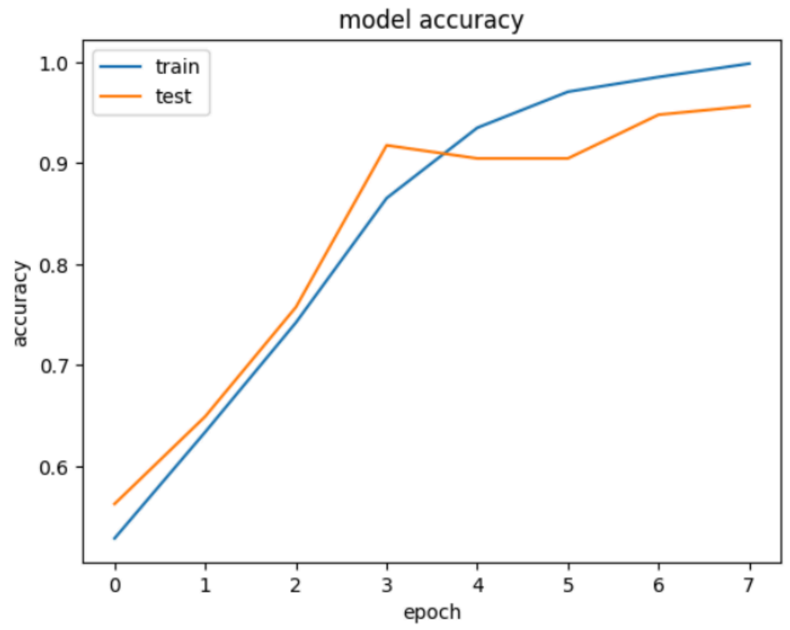
```
class new_callback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('accuracy') > 0.9855:
            self.model.stop_training = True

stop_epoch= new_callback()
```

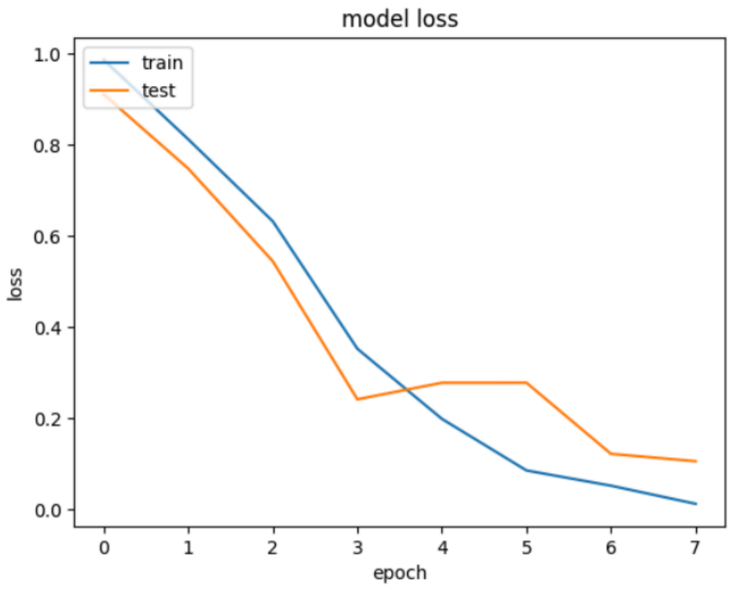
```
[14] warnings.filterwarnings('ignore')
     model_hist=model.fit(x_train,y_train,epochs=10, batch_size=32, callbacks=[stop_epoch], validation_split=.05)
```

```
Epoch 1/10
137/137 ━━━━━━━━━━━━━━━━━━━━ 72s 482ms/step - accuracy: 0.4857 - loss: 1.0428 - val_accuracy: 0.5628 - val_loss: 0.9107
Epoch 2/10
137/137 ━━━━━━━━━━━━━━━━━━━━ 71s 412ms/step - accuracy: 0.6072 - loss: 0.8682 - val_accuracy: 0.6494 - val_loss: 0.7480
Epoch 3/10
137/137 ━━━━━━━━━━━━━━━━━━━━ 54s 395ms/step - accuracy: 0.7127 - loss: 0.6798 - val_accuracy: 0.7576 - val_loss: 0.5447
Epoch 4/10
137/137 ━━━━━━━━━━━━━━━━━━━━ 84s 408ms/step - accuracy: 0.8525 - loss: 0.3833 - val_accuracy: 0.9177 - val_loss: 0.2420
Epoch 5/10
137/137 ━━━━━━━━━━━━━━━━━━━━ 81s 405ms/step - accuracy: 0.9292 - loss: 0.2177 - val_accuracy: 0.9048 - val_loss: 0.2783
Epoch 6/10
137/137 ━━━━━━━━━━━━━━━━━━━━ 87s 444ms/step - accuracy: 0.9684 - loss: 0.0907 - val_accuracy: 0.9048 - val_loss: 0.2784
Epoch 7/10
137/137 ━━━━━━━━━━━━━━━━━━━━ 58s 420ms/step - accuracy: 0.9869 - loss: 0.0539 - val_accuracy: 0.9481 - val_loss: 0.1224
Epoch 8/10
137/137 ━━━━━━━━━━━━━━━━━━━━ 57s 413ms/step - accuracy: 0.9991 - loss: 0.0145 - val_accuracy: 0.9567 - val_loss: 0.1063
```



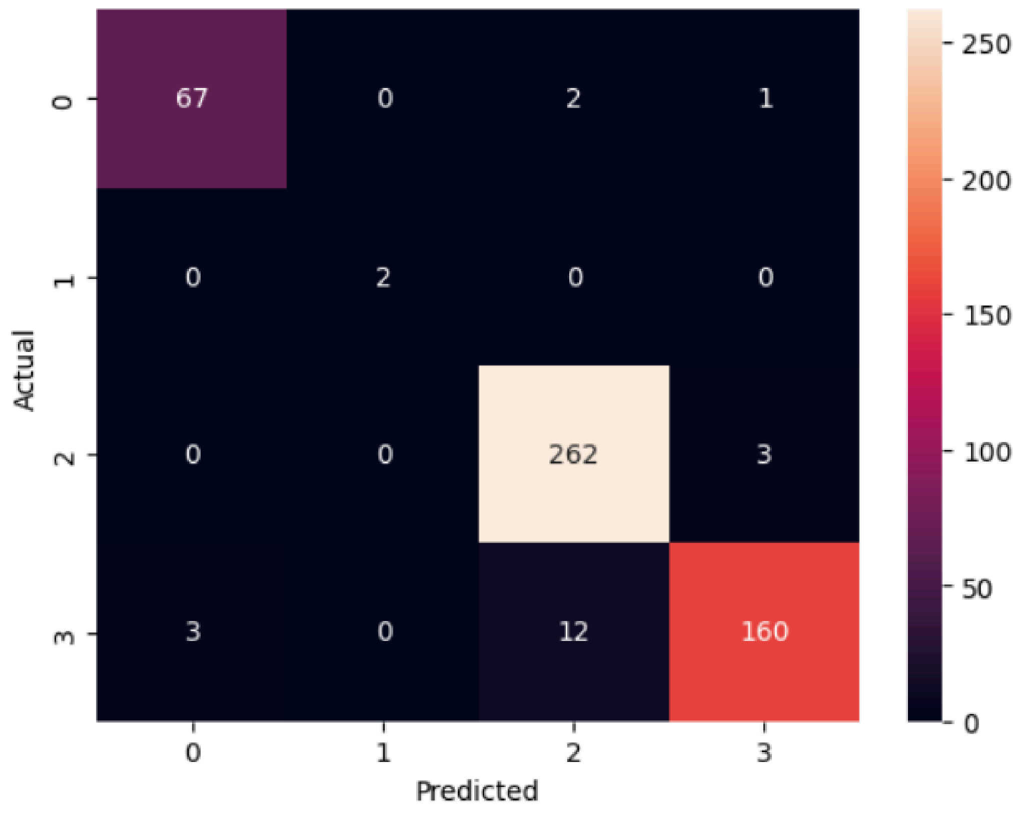
```
# summarize history for accuracy
plt.plot(model_hist.history['accuracy'])
plt.plot(model_hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(model_hist.history['loss'])
plt.plot(model_hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```


model accuracy
train
test
accuracy
1.0
0.9
0.8
0.7
0.6
0
1
2
3
4
5
6
7
epoch


model loss
train
test
loss
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
epoch

```
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
cm = confusion_matrix(y_test, y_pred_classes)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 112ms/step
```



Conclusion :

The confusion matrix shows that the model is good at identifying "Non_Demented" and "Very_Mild_Demented" people. However, it has trouble telling apart "Mild_Demented" from "Very_Mild_Demented," and it also struggles with "Moderate_Demented" since it doesn’t predict this class well. The model might need some adjustments or improvements to better differentiate between the different levels of dementia, especially between mild and very mild cases.