# untitled3

December 6, 2023

[ ]:
```
import numpy as np
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Assuming the dataset is in the root of your Google Drive
base_directory = '/content/drive/My Drive/Alzheimer Dataset'
train_directory = os.path.join(base_directory, 'training')
test_directory = os.path.join(base_directory, 'testing')

# Use ImageDataGenerator for on-the-fly data augmentation
batch_size = 64  # Experiment with different batch sizes
target_size = (224, 224)  # Adjust the target size based on your model's input␣
 ↪requirements
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,␣
 ↪zoom_range=0.2, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

# Flow from directory for on-the-fly loading and preprocessing during training
train_generator = train_datagen.flow_from_directory(
    train_directory,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='categorical',  # Use 'categorical' for multi-class␣
 ↪classification
    shuffle=True
)

test_generator = test_datagen.flow_from_directory(
    test_directory,
    target_size=target_size,
```

```
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Build a more complex CNN model
model = Sequential()
model.add(Conv2D(64, (3, 3), input_shape=(224, 224, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(4, activation='softmax'))

# Compile the model with a lower learning rate
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model with 10 epochs
num_epochs = 10
history = model.fit(train_generator, steps_per_epoch=len(train_generator),␣
 ↪epochs=num_epochs, validation_data=test_generator,␣
 ↪validation_steps=len(test_generator))

# Evaluate the model on the test set
y_pred = model.predict(test_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_generator.classes

# Output accuracy and classification report
print("Accuracy: ", accuracy_score(y_true, y_pred_classes))
print("Classification Report:\n", classification_report(y_true, y_pred_classes,␣
 ↪target_names=test_generator.class_indices, zero_division=1))
```

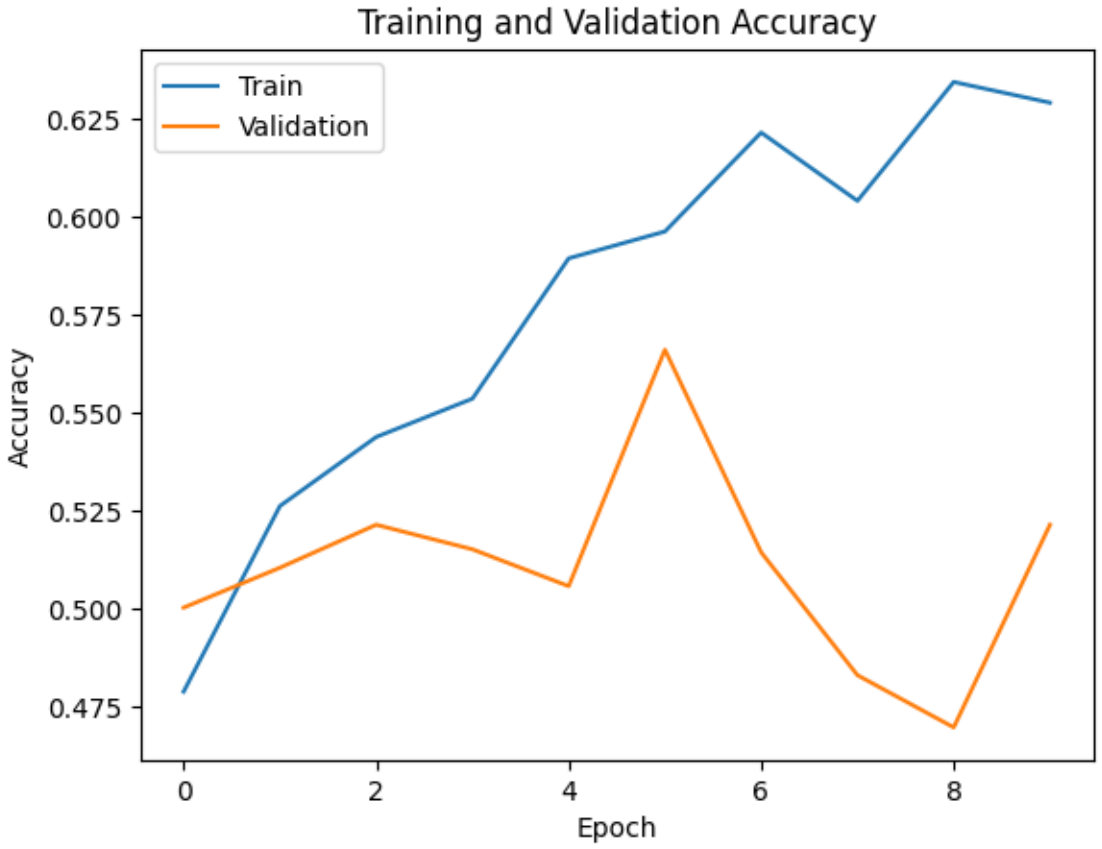
```
Mounted at /content/drive
Found 5121 images belonging to 4 classes.
Found 1279 images belonging to 4 classes.
Epoch 1/10
81/81 [==============================] - 1015s 12s/step - loss: 1.9429 -
accuracy: 0.4858 - val_loss: 1.0100 - val_accuracy: 0.5137
Epoch 2/10
81/81 [==============================] - 85s 1s/step - loss: 0.9782 - accuracy:
0.5286 - val_loss: 0.9791 - val_accuracy: 0.5129
Epoch 3/10
81/81 [==============================] - 85s 1s/step - loss: 0.9226 - accuracy:
0.5589 - val_loss: 0.9663 - val_accuracy: 0.5387
```

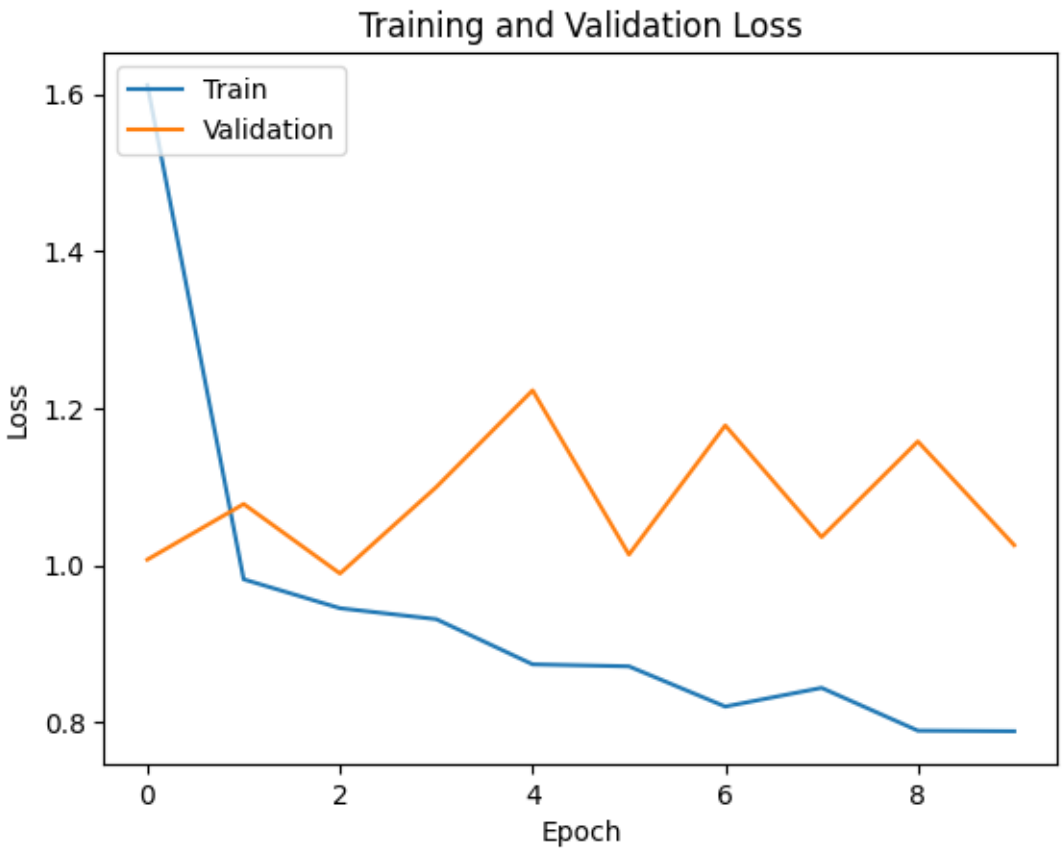
```
Epoch 4/10
81/81 [==============================] - 86s 1s/step - loss: 0.8908 - accuracy:
0.5702 - val_loss: 0.9471 - val_accuracy: 0.5238
Epoch 5/10
81/81 [==============================] - 86s 1s/step - loss: 0.8693 - accuracy:
0.5868 - val_loss: 0.9772 - val_accuracy: 0.5387
Epoch 6/10
81/81 [==============================] - 86s 1s/step - loss: 0.8581 - accuracy:
0.5973 - val_loss: 0.9506 - val_accuracy: 0.5324
Epoch 7/10
81/81 [==============================] - 84s 1s/step - loss: 0.8365 - accuracy:
0.5971 - val_loss: 1.0052 - val_accuracy: 0.5403
Epoch 8/10
81/81 [==============================] - 84s 1s/step - loss: 0.8550 - accuracy:
0.5925 - val_loss: 0.9900 - val_accuracy: 0.5012
Epoch 9/10
81/81 [==============================] - 84s 1s/step - loss: 0.8167 - accuracy:
0.6126 - val_loss: 0.9546 - val_accuracy: 0.5246
Epoch 10/10
81/81 [==============================] - 84s 1s/step - loss: 0.8040 - accuracy:
0.6161 - val_loss: 0.9721 - val_accuracy: 0.5473
20/20 [==============================] - 4s 186ms/step
Accuracy:  0.547302580140735
Classification Report:
                    precision    recall  f1-score   support

    MildDemented       0.38      0.03      0.05       179
ModerateDemented       1.00      0.00      0.00        12
     NonDemented       0.59      0.70      0.64       640
VeryMildDemented       0.49      0.55      0.52       448

        accuracy                           0.55      1279
       macro avg       0.62      0.32      0.30      1279
    weighted avg       0.53      0.55      0.51      1279
```

**Training and Validation curves using matplotlib**

[ ]:
```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

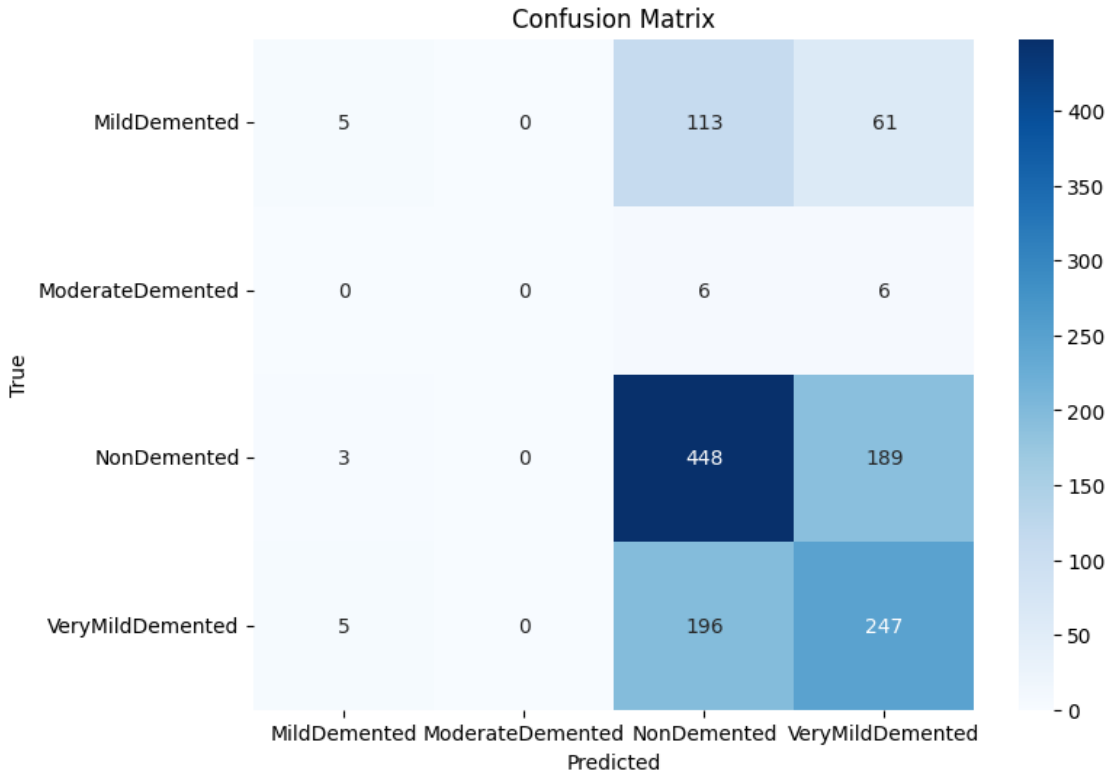





[ ]:
```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Get predictions for the test set
y_pred = model.predict(test_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_generator.classes

# Generate confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)

# Plot confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=test_generator.class_indices.keys(), yticklabels=test_generator.
 ↪class_indices.keys())
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
```

```
plt.ylabel('True')
plt.show()
```

```
20/20 [==============================] - 7s 380ms/step
```




### Evaluation Model

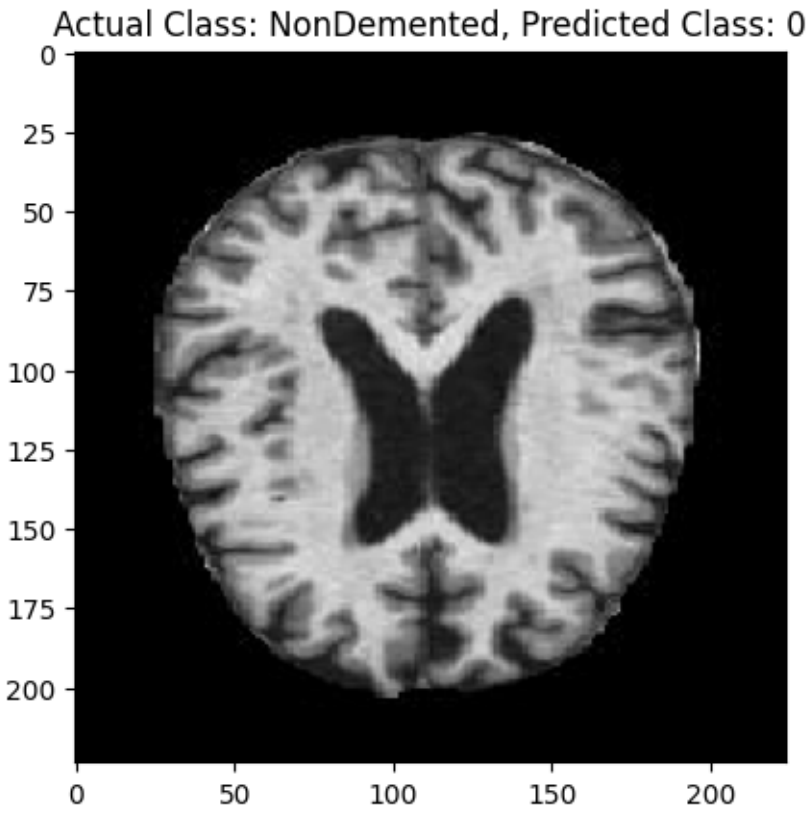
[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix

# Evaluate the model on the test set
test_results = model.evaluate(test_generator)

# Make predictions on the test set
y_pred = model.predict(test_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_generator.classes

# Output accuracy and classification report
print("Test Loss:", test_results[0])
print("Test Accuracy:", test_results[1])
print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred_classes))
print("Classification Report:\n", classification_report(y_true, y_pred_classes,␣
  ↪target_names=test_generator.class_indices, zero_division=1))
```

```
20/20 [==============================] - 4s 203ms/step - loss: 0.9782 -
accuracy: 0.5512
20/20 [==============================] - 3s 170ms/step
Test Loss: 0.9781922698020935
Test Accuracy: 0.5512118935585022
Confusion Matrix:
 [[ 34   0  67  78]
 [  1   2   5   4]
 [ 21   0 420 199]
 [ 14   0 185 249]]
Classification Report:
                   precision    recall  f1-score   support

    MildDemented       0.49      0.19      0.27       179
ModerateDemented       1.00      0.17      0.29        12
     NonDemented       0.62      0.66      0.64       640
VeryMildDemented       0.47      0.56      0.51       448

        accuracy                           0.55      1279
       macro avg       0.64      0.39      0.43      1279
    weighted avg       0.55      0.55      0.54      1279
```

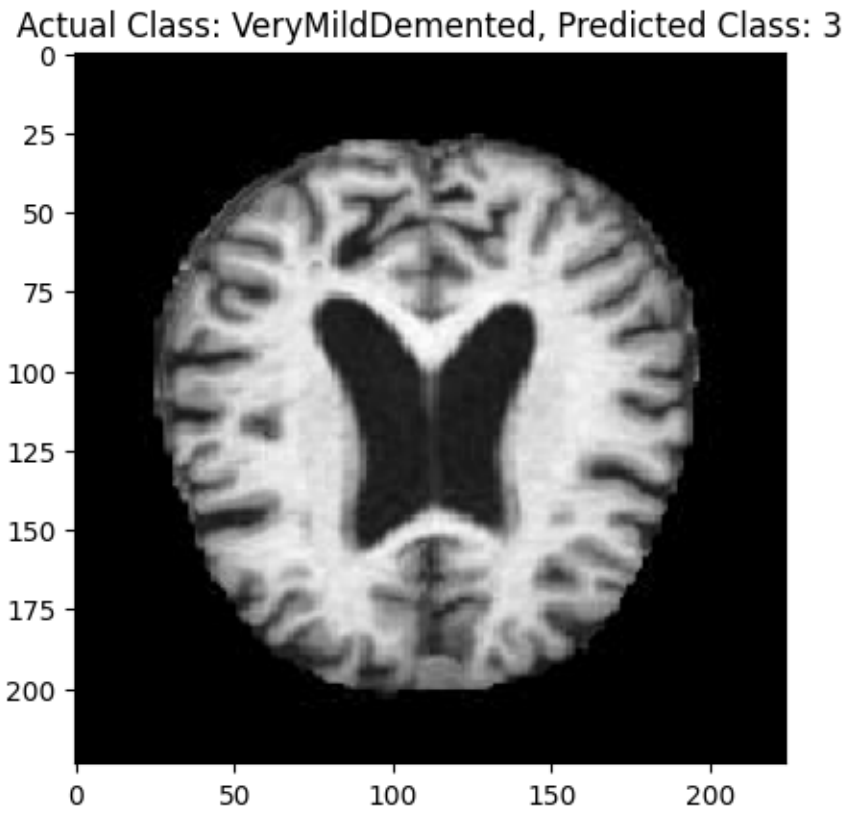
[ ]:
```python
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model

# Load the trained model
model_path = 'Alzheimer_1_74.04.h5'  # Replace with the actual path to your␣
 ↪model
loaded_model = load_model(model_path)

# Directory containing sample images
sample_directory = '/content/drive/My Drive/Alzheimer Dataset/testing'  # Use␣
 ↪the testing directory

# List of classes
classes = os.listdir(sample_directory)

# Visualize predictions for one sample image from each class
for class_name in classes:
    class_directory = os.path.join(sample_directory, class_name)
    sample_images = os.listdir(class_directory)[:1]  # Take one sample image␣
 ↪from each class
```

```
    for image_file in sample_images:
        # Load and preprocess the image
        img_path = os.path.join(class_directory, image_file)
        img = image.load_img(img_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array /= 255.0  # Normalize the pixel values

        # Make predictions
        predictions = loaded_model.predict(img_array)
        predicted_class = np.argmax(predictions[0])

        # Display the image and predicted class
        plt.imshow(img)
        plt.title(f'Actual Class: {class_name}, Predicted Class:␣
↪{predicted_class}')
        plt.show()
```

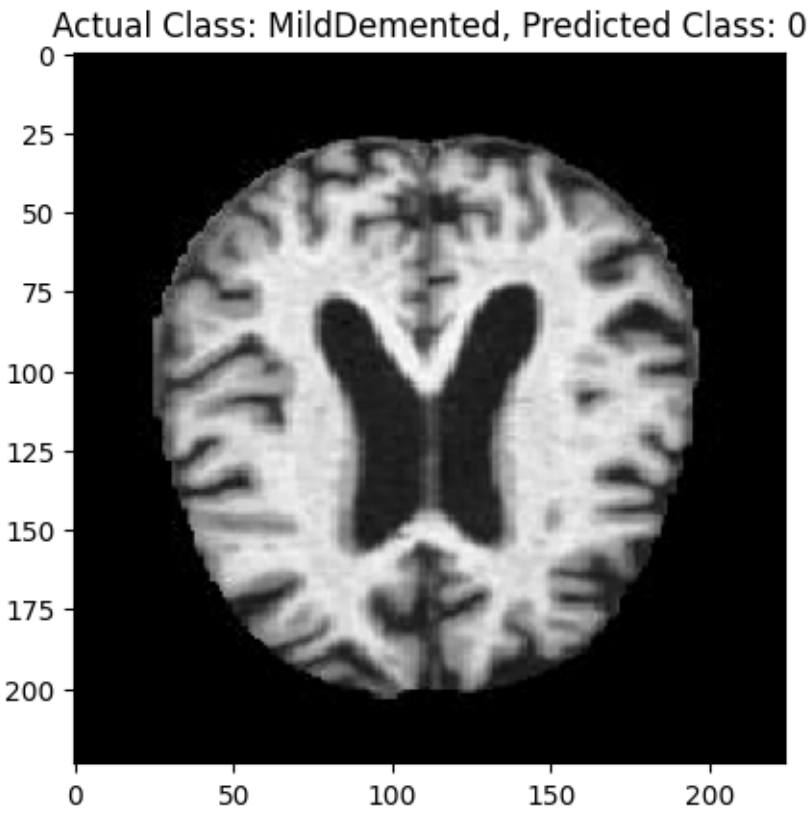
```
1/1 [==============================] - 0s 63ms/step
```



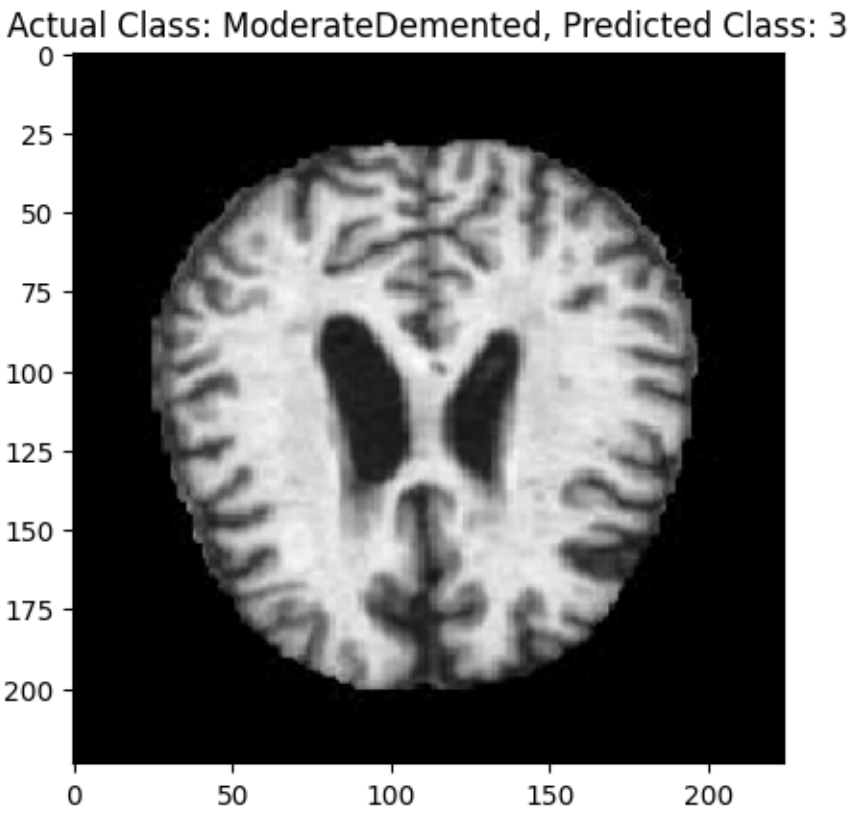

```
1/1 [==============================] - 0s 17ms/step
```




```
1/1 [==============================] - 0s 22ms/step
```



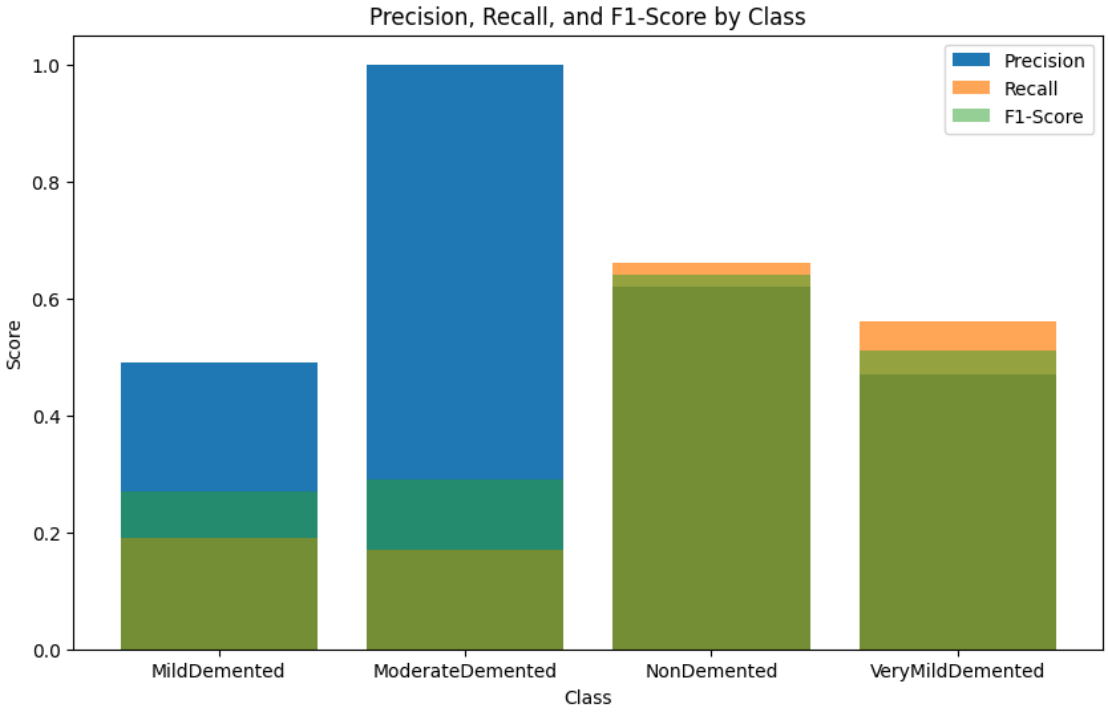

```
1/1 [==============================] - 0s 22ms/step
```



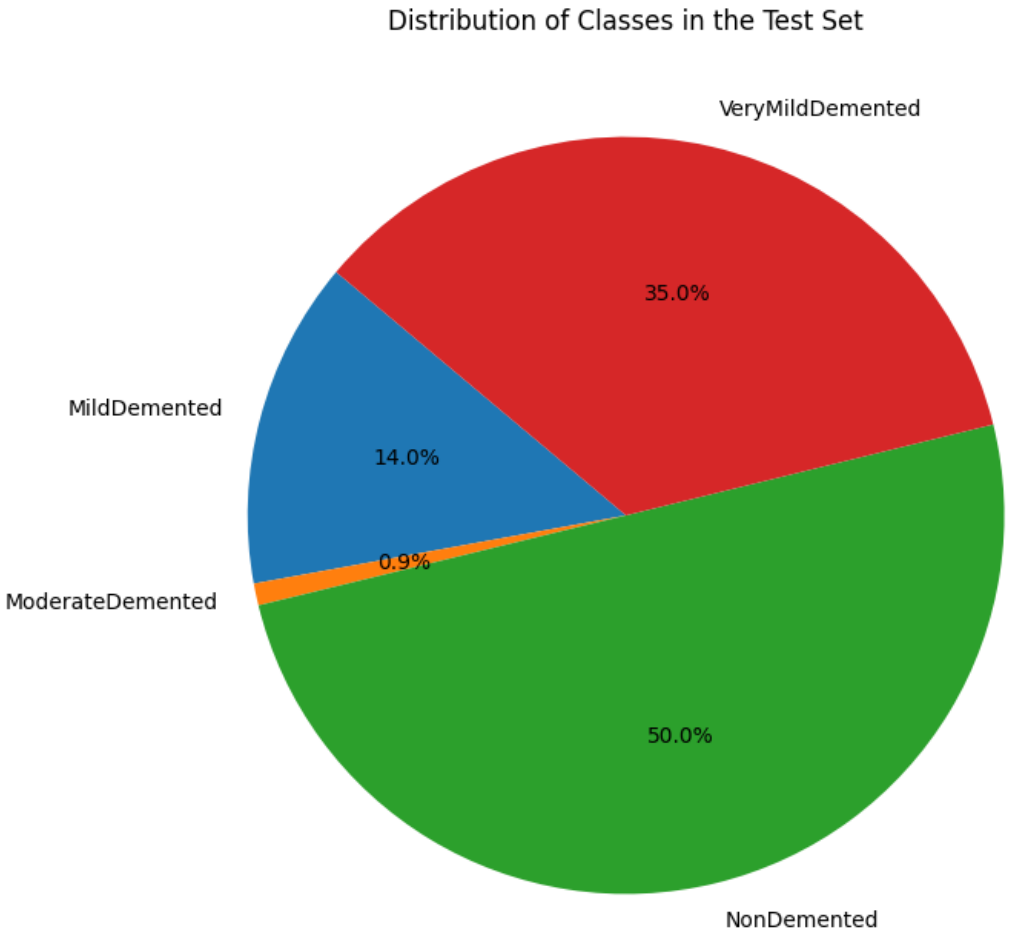

***Bar Graph to visually represent the result Of Classification model.***

[ ]:
```
import matplotlib.pyplot as plt

# Data for plotting
categories = ['MildDemented', 'ModerateDemented', 'NonDemented',␣
 ↪'VeryMildDemented']
precision = [0.49, 1.00, 0.62, 0.47]
recall = [0.19, 0.17, 0.66, 0.56]
f1_score = [0.27, 0.29, 0.64, 0.51]

# Bar graph for Precision, Recall, and F1-Score
plt.figure(figsize=(10, 6))
plt.bar(categories, precision, label='Precision')
plt.bar(categories, recall, label='Recall', alpha=0.7)
plt.bar(categories, f1_score, label='F1-Score', alpha=0.5)

plt.title('Precision, Recall, and F1-Score by Class')
plt.xlabel('Class')
plt.ylabel('Score')
plt.legend()
```

```
plt.show()
```




[ ]:
```
# Data for pie chart
support = [179, 12, 640, 448]

plt.figure(figsize=(8, 8))
plt.pie(support, labels=categories, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Classes in the Test Set')
plt.show()
```


Distribution of Classes in the Test Set
VeryMildDemented
35.0%
MildDemented
14.0%
0.9%
ModerateDemented
50.0%
NonDemented