oasis-alzheimer-s-detection

July 23, 2025

1 Import libraries

[1]:
```
# import system libs
import os
import time
import shutil
import pathlib

import itertools

# import data handling tools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

# import Deep learning Libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.layers import Conv2D,
    ↪MaxPooling2D,Flatten,Dense,Activation,Dropout,BatchNormalization
from tensorflow.keras import regularizers

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy
```

```
(detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

2 Import Dataset

[2]:
```python
path1 = []
path2 = []
path3 = []
path4 = []
for dirname, _, filenames in os.walk('/kaggle/input/imagesoasis/Data/Non␣
 ↪Demented'):
    for filename in filenames:
        path1.append(os.path.join(dirname, filename))

for dirname, _, filenames in os.walk('/kaggle/input/imagesoasis/Data/Mild␣
 ↪Dementia'):
    for filename in filenames:
        path2.append(os.path.join(dirname, filename))

for dirname, _, filenames in os.walk('/kaggle/input/imagesoasis/Data/Moderate␣
 ↪Dementia'):
    for filename in filenames:
        path3.append(os.path.join(dirname, filename))

for dirname, _, filenames in os.walk('/kaggle/input/imagesoasis/Data/Very mild␣
 ↪Dementia'):
    for filename in filenames:
        path4.append(os.path.join(dirname, filename))
```

[3]:
```python
paths=[]
paths.append(path1)
paths.append(path2)
paths.append(path3)
paths.append(path4)
```

[4]:
```python
# Generate data paths with labels
def define_paths(data_dir):
    filepaths = []
    labels = []

    folds = os.listdir(data_dir)
    for fold in folds:
        foldpath = os.path.join(data_dir, fold)
        filelist = os.listdir(foldpath)
        for file in filelist:
            fpath = os.path.join(foldpath, file)
```

```
            filepaths.append(fpath)
            labels.append(fold)

    return filepaths, labels


# Concatenate data paths with labels into one dataframe ( to later be fitted␣
↪into the model )
def define_df(files, classes):
    Fseries = pd.Series(files, name= 'filepaths')
    Lseries = pd.Series(classes, name='labels')
    return pd.concat([Fseries, Lseries], axis= 1)

# Split dataframe to train, valid, and test
def split_data(data_dir):
    # train dataframe
    files, classes = define_paths(data_dir)
    df = define_df(files, classes)
    strat = df['labels']
    train_df, dummy_df = train_test_split(df,  train_size= 0.8, shuffle= True,␣
↪random_state= 123, stratify= strat)

    # valid and test dataframe
    strat = dummy_df['labels']
    valid_df, test_df = train_test_split(dummy_df,  train_size= 0.5, shuffle=␣
↪True, random_state= 123, stratify= strat)

    return train_df, valid_df, test_df
```

[5]:
```
def create_gens (train_df, valid_df, test_df, batch_size):
    '''
    This function takes train, validation, and test dataframe and fit them into␣
↪image data generator, because model takes data from image data generator.
    Image data generator converts images into tensors. '''


    # define model parameters
    img_size = (224, 224)
    channels = 3 # either BGR or Grayscale
    color = 'rgb'
    img_shape = (img_size[0], img_size[1], channels)

    # Recommended : use custom function for test data batch size, else we can␣
↪use normal batch size.
    ts_length = len(test_df)
    test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length +␣
↪1) if ts_length%n == 0 and ts_length/n <= 80]))
```

```
    test_steps = ts_length // test_batch_size

    # This function which will be used in image data generator for data␣
↪augmentation, it just take the image and return it again.
    def scalar(img):
        return img

    tr_gen = ImageDataGenerator(preprocessing_function= scalar,␣
↪horizontal_flip= True)
    ts_gen = ImageDataGenerator(preprocessing_function= scalar)

    train_gen = tr_gen.flow_from_dataframe( train_df, x_col= 'filepaths',␣
↪y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                    color_mode= color, shuffle= True,␣
↪batch_size= batch_size)

    valid_gen = ts_gen.flow_from_dataframe( valid_df, x_col= 'filepaths',␣
↪y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                    color_mode= color, shuffle= True,␣
↪batch_size= batch_size)

    # Note: we will use custom test_batch_size, and make shuffle= false
    test_gen = ts_gen.flow_from_dataframe( test_df, x_col= 'filepaths', y_col=␣
↪'labels', target_size= img_size, class_mode= 'categorical',
                                    color_mode= color, shuffle= False,␣
↪batch_size= test_batch_size)

    return train_gen, valid_gen, test_gen
```

[6]:
```
data_dir = '/kaggle/input/imagesoasis/Data'


# Get splitted data
train_df, valid_df, test_df = split_data(data_dir)

# Get Generators
batch_size = 40
train_gen, valid_gen, test_gen = create_gens(train_df, valid_df, test_df,␣
↪batch_size)
```

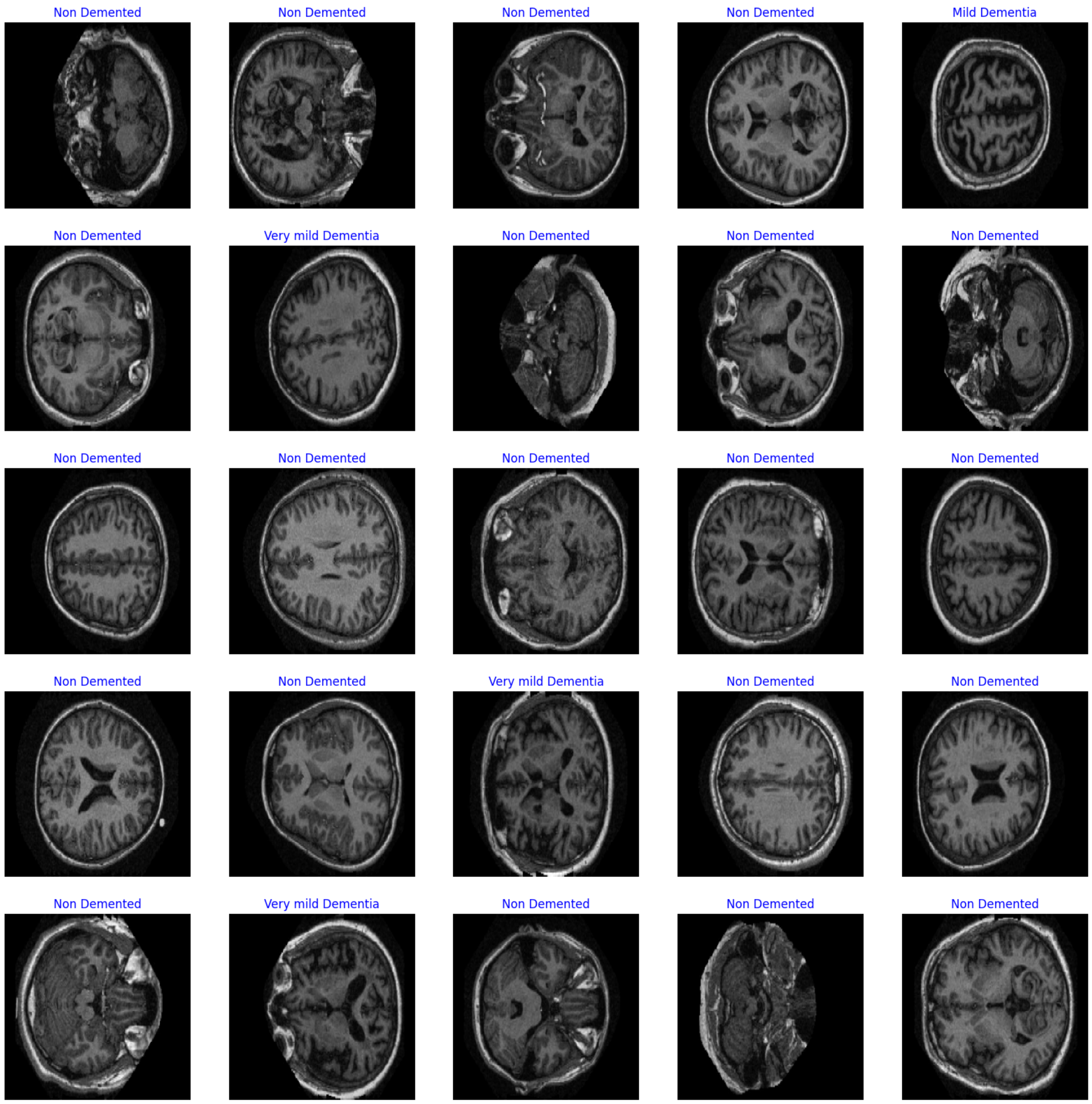
```
Found 69149 validated image filenames belonging to 4 classes.
Found 8644 validated image filenames belonging to 4 classes.
Found 8644 validated image filenames belonging to 4 classes.
```

[7]:
```
def show_image(gen):

    '''
```

```
    This fuction take the data generator and show sample of
    the images
    '''
    # return classes, images to be displayed
    g_dict = gen.class_indices #define dictionary{'class':index}
    classes = list(g_dict.keys()) #define list of dictionary's␣
↪kays(classes),classes names: string
    images, labels = next(gen) # get a batch size samples from the generator

    #calculate number of displayed samples
    length = len(labels) # length of batch size
    sample = min(length, 25) # check if sample less than 25 images


    plt.figure(figsize=(20,20))
    for i in range(sample):
        plt.subplot(5, 5, i+1)
        image = images[i]/255 # scales data to range(0-255)
        plt.imshow(image)
        index = np.argmax(labels[i]) # get image index
        class_name = classes[index] # get class of image
        plt.title(class_name, color='blue', fontsize=12)
        plt.axis('off')
    plt.show()
```

[8]:

```
show_image(train_gen)
```



3 Creating Model: CNN

[9]:
```
IMAGE_SIZE = 224
BATCH_SIZE = 32
```

[10]:
```
model = keras.Sequential([
Conv2D(filters=64, kernel_size=(3,3),padding="same",
activation='relu', input_shape= (IMAGE_SIZE,IMAGE_SIZE,3)),
Conv2D(filters=64, kernel_size=(3,3),padding="same",
activation='relu', input_shape= (IMAGE_SIZE,IMAGE_SIZE,3)),
MaxPooling2D(pool_size=(2,2)),
Conv2D(filters=128, kernel_size=(3,3),padding="same",
```

```
activation='relu', input_shape= (IMAGE_SIZE,IMAGE_SIZE,3)),
Conv2D(filters=128, kernel_size=(3,3),padding="same",
activation='relu', input_shape= (IMAGE_SIZE,IMAGE_SIZE,3)),
MaxPooling2D(pool_size=(2,2)),
Conv2D(filters=256, kernel_size=(3,3),padding="same",
activation='relu', input_shape= (IMAGE_SIZE,IMAGE_SIZE,3)),
MaxPooling2D(pool_size=(2, 2)),
Flatten(),
Dropout(0.4),
Dense(256,activation='relu'),
Dense(128,activation='relu'),
Dense(64,activation='relu'),
Dense(4, activation='softmax',
dtype='float32')
])
```

[11]:
```
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001),
loss='categorical_crossentropy',metrics=['accuracy'])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 64)      1792

 conv2d_1 (Conv2D)           (None, 224, 224, 64)      36928

 max_pooling2d (MaxPooling2D  (None, 112, 112, 64)     0
 )

 conv2d_2 (Conv2D)           (None, 112, 112, 128)     73856

 conv2d_3 (Conv2D)           (None, 112, 112, 128)     147584

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 128)      0
 2D)

 conv2d_4 (Conv2D)           (None, 56, 56, 256)       295168

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 256)      0
 2D)

 flatten (Flatten)           (None, 200704)            0

 dropout (Dropout)           (None, 200704)            0
```

```
 dense (Dense)                (None, 256)               51380480

 dense_1 (Dense)              (None, 128)               32896

 dense_2 (Dense)              (None, 64)                8256

 dense_3 (Dense)              (None, 4)                 260

=================================================================
Total params: 51,977,220
Trainable params: 51,977,220
Non-trainable params: 0
_________________________________________________________________
None
```

[12]:

```
callbacks = [
EarlyStopping(
monitor='val_loss',
patience=5,
restore_best_weights=True),

ModelCheckpoint(
filepath='CCN.h5',
monitor='val_loss',
save_best_only=True,
save_weights_only=False,
mode='min',
verbose=1),

keras.callbacks.ReduceLROnPlateau(
monitor='val_loss',
factor=0.5,
patience=3,
min_lr=1e-7,
verbose=1)]
```

[ ]:

```
epochs = 20
history = model.fit(train_gen,epochs= epochs,
validation_data = test_gen,callbacks = callbacks)
```

```
Epoch 1/20
1729/1729 [==============================] - ETA: 0s - loss: 0.2974 - accuracy:
0.9029
Epoch 1: val_loss improved from inf to 0.07214, saving model to CCN.h5
1729/1729 [==============================] - 747s 424ms/step - loss: 0.2974 -
accuracy: 0.9029 - val_loss: 0.0721 - val_accuracy: 0.9785 - lr: 1.0000e-04
Epoch 2/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0434 - accuracy:
```

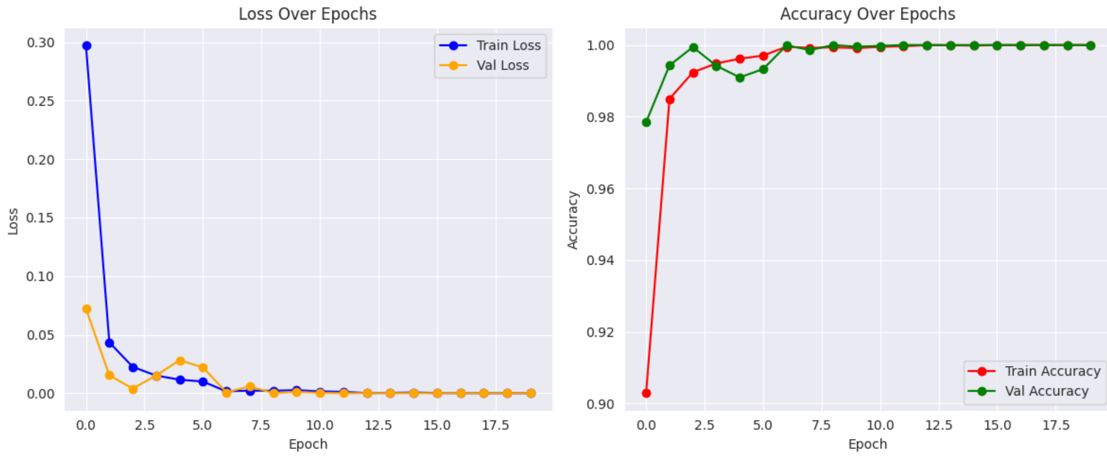
```
0.9850
Epoch 2: val_loss improved from 0.07214 to 0.01527, saving model to CCN.h5
1729/1729 [==============================] - 688s 398ms/step - loss: 0.0434 -
accuracy: 0.9850 - val_loss: 0.0153 - val_accuracy: 0.9943 - lr: 1.0000e-04
Epoch 3/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0224 - accuracy:
0.9924
Epoch 3: val_loss improved from 0.01527 to 0.00387, saving model to CCN.h5
1729/1729 [==============================] - 687s 397ms/step - loss: 0.0224 -
accuracy: 0.9924 - val_loss: 0.0039 - val_accuracy: 0.9994 - lr: 1.0000e-04
Epoch 4/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0150 - accuracy:
0.9949
Epoch 4: val_loss did not improve from 0.00387
1729/1729 [==============================] - 680s 393ms/step - loss: 0.0150 -
accuracy: 0.9949 - val_loss: 0.0151 - val_accuracy: 0.9942 - lr: 1.0000e-04
Epoch 5/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0116 - accuracy:
0.9962
Epoch 5: val_loss did not improve from 0.00387
1729/1729 [==============================] - 675s 390ms/step - loss: 0.0116 -
accuracy: 0.9962 - val_loss: 0.0281 - val_accuracy: 0.9910 - lr: 1.0000e-04
Epoch 6/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0099 - accuracy:
0.9971
Epoch 6: val_loss did not improve from 0.00387

Epoch 6: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
1729/1729 [==============================] - 670s 387ms/step - loss: 0.0099 -
accuracy: 0.9971 - val_loss: 0.0221 - val_accuracy: 0.9933 - lr: 1.0000e-04
Epoch 7/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0016 - accuracy:
0.9995
Epoch 7: val_loss improved from 0.00387 to 0.00064, saving model to CCN.h5
1729/1729 [==============================] - 675s 390ms/step - loss: 0.0016 -
accuracy: 0.9995 - val_loss: 6.4394e-04 - val_accuracy: 0.9999 - lr: 5.0000e-05
Epoch 8/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0021 - accuracy:
0.9992
Epoch 8: val_loss did not improve from 0.00064
1729/1729 [==============================] - 674s 390ms/step - loss: 0.0021 -
accuracy: 0.9992 - val_loss: 0.0058 - val_accuracy: 0.9986 - lr: 5.0000e-05
Epoch 9/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0020 - accuracy:
0.9994
Epoch 9: val_loss improved from 0.00064 to 0.00005, saving model to CCN.h5
1729/1729 [==============================] - 675s 390ms/step - loss: 0.0020 -
accuracy: 0.9994 - val_loss: 4.7086e-05 - val_accuracy: 1.0000 - lr: 5.0000e-05
```

```
Epoch 10/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0026 - accuracy:
0.9991
Epoch 10: val_loss did not improve from 0.00005
1729/1729 [==============================] - 671s 388ms/step - loss: 0.0026 -
accuracy: 0.9991 - val_loss: 0.0010 - val_accuracy: 0.9995 - lr: 5.0000e-05
Epoch 11/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0015 - accuracy:
0.9995
Epoch 11: val_loss did not improve from 0.00005
1729/1729 [==============================] - 668s 386ms/step - loss: 0.0015 -
accuracy: 0.9995 - val_loss: 3.0757e-04 - val_accuracy: 0.9998 - lr: 5.0000e-05
Epoch 12/20
1729/1729 [==============================] - ETA: 0s - loss: 0.0011 - accuracy:
0.9997
Epoch 12: val_loss improved from 0.00005 to 0.00001, saving model to CCN.h5

Epoch 12: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
1729/1729 [==============================] - 676s 391ms/step - loss: 0.0011 -
accuracy: 0.9997 - val_loss: 6.6219e-06 - val_accuracy: 1.0000 - lr: 5.0000e-05
Epoch 13/20
1729/1729 [==============================] - ETA: 0s - loss: 7.9264e-05 -
accuracy: 1.0000
Epoch 13: val_loss improved from 0.00001 to 0.00000, saving model to CCN.h5
1729/1729 [==============================] - 673s 389ms/step - loss: 7.9264e-05
- accuracy: 1.0000 - val_loss: 6.6772e-07 - val_accuracy: 1.0000 - lr:
2.5000e-05
Epoch 14/20
1729/1729 [==============================] - ETA: 0s - loss: 2.4535e-04 -
accuracy: 0.9999
Epoch 14: val_loss did not improve from 0.00000
1729/1729 [==============================] - 676s 391ms/step - loss: 2.4535e-04
- accuracy: 0.9999 - val_loss: 1.3947e-05 - val_accuracy: 1.0000 - lr:
2.5000e-05
Epoch 15/20
1729/1729 [==============================] - ETA: 0s - loss: 5.7687e-04 -
accuracy: 0.9999
Epoch 15: val_loss did not improve from 0.00000

Epoch 15: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
1729/1729 [==============================] - 671s 388ms/step - loss: 5.7687e-04
- accuracy: 0.9999 - val_loss: 8.8111e-06 - val_accuracy: 1.0000 - lr:
2.5000e-05
Epoch 16/20
1729/1729 [==============================] - ETA: 0s - loss: 1.0817e-04 -
accuracy: 1.0000
Epoch 16: val_loss did not improve from 0.00000
1729/1729 [==============================] - 671s 388ms/step - loss: 1.0817e-04
```

```
- accuracy: 1.0000 - val_loss: 2.1796e-06 - val_accuracy: 1.0000 - lr:
1.2500e-05
Epoch 17/20
1729/1729 [==============================] - ETA: 0s - loss: 5.1841e-05 -
accuracy: 1.0000
Epoch 17: val_loss did not improve from 0.00000
1729/1729 [==============================] - 671s 388ms/step - loss: 5.1841e-05
- accuracy: 1.0000 - val_loss: 4.9437e-06 - val_accuracy: 1.0000 - lr:
1.2500e-05
Epoch 18/20
1729/1729 [==============================] - ETA: 0s - loss: 3.4733e-05 -
accuracy: 1.0000
Epoch 18: val_loss improved from 0.00000 to 0.00000, saving model to CCN.h5

Epoch 18: ReduceLROnPlateau reducing learning rate to 6.24999984211172e-06.
1729/1729 [==============================] - 674s 389ms/step - loss: 3.4733e-05
- accuracy: 1.0000 - val_loss: 5.1673e-08 - val_accuracy: 1.0000 - lr:
1.2500e-05
Epoch 19/20
1729/1729 [==============================] - ETA: 0s - loss: 2.1252e-05 -
accuracy: 1.0000
Epoch 19: val_loss improved from 0.00000 to 0.00000, saving model to CCN.h5
1729/1729 [==============================] - 675s 390ms/step - loss: 2.1252e-05
- accuracy: 1.0000 - val_loss: 3.6366e-08 - val_accuracy: 1.0000 - lr:
6.2500e-06
Epoch 20/20
1021/1729 [================>…] - ETA: 4:23 - loss: 2.1535e-05 -
accuracy: 1.0000
```

[16]:
```
#Plotting training and validation loss and accuracy
plt.figure(figsize=(12, 5))
#  Loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss', marker='o',color='blue')
plt.plot(history.history['val_loss'], label='Val Loss',␣
↪marker='o',color='orange')
plt.title('Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
#  Accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy',␣
↪marker='o',color='red')
plt.plot(history.history['val_accuracy'], label='Val Accuracy',␣
↪marker='o',color='green')
plt.title('Accuracy Over Epochs')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()
10
```



[16]: 10

4 Accuracy and Prediction

[17]:
```
pred_probs = model.predict(test_gen, verbose=1)
y_pred = np.argmax(pred_probs, axis=1)
```

```
2161/2161 [==============================] - 26s 12ms/step
```

[18]:
```
y_true = test_gen.classes
```

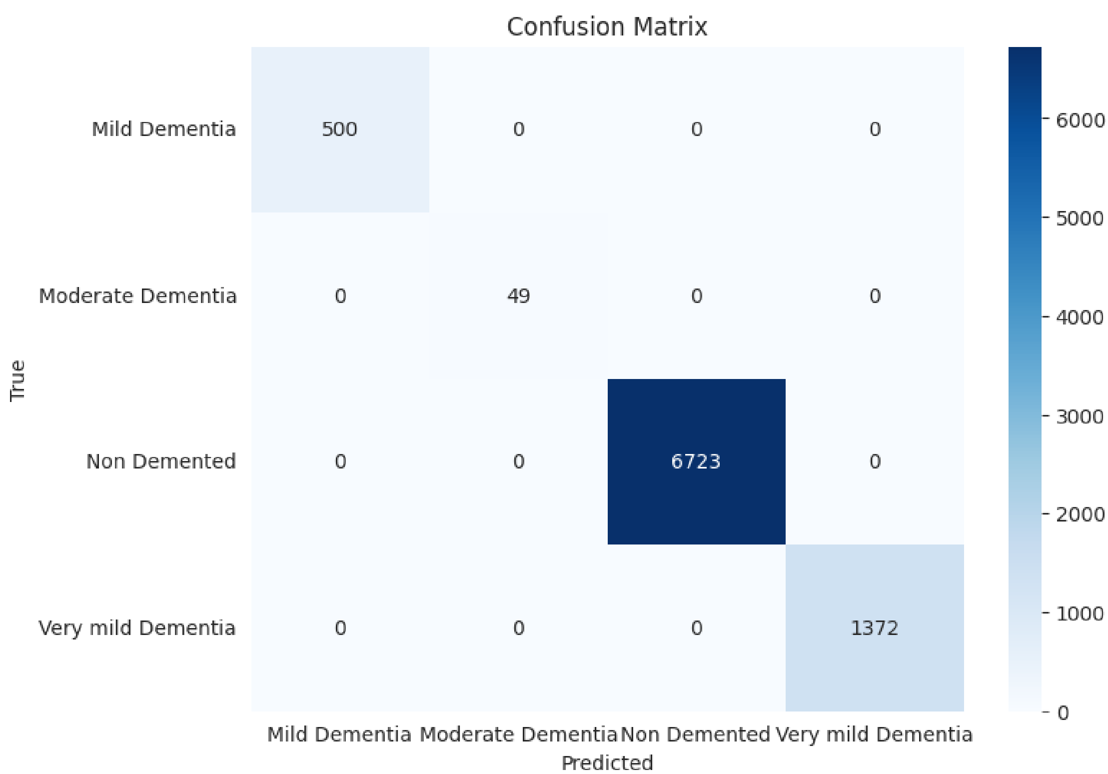
[19]:
```
# Classification report
target_names = list(test_gen.class_indices.keys())
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
                    precision    recall  f1-score   support

     Mild Dementia       1.00      1.00      1.00       500
 Moderate Dementia       1.00      1.00      1.00        49
      Non Demented       1.00      1.00      1.00      6723
Very mild Dementia       1.00      1.00      1.00      1372

          accuracy                           1.00      8644
         macro avg       1.00      1.00      1.00      8644
```

```
weighted avg       1.00      1.00      1.00      8644
```

[20]:

```
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=target_names,␣
 ↪yticklabels=target_names, cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

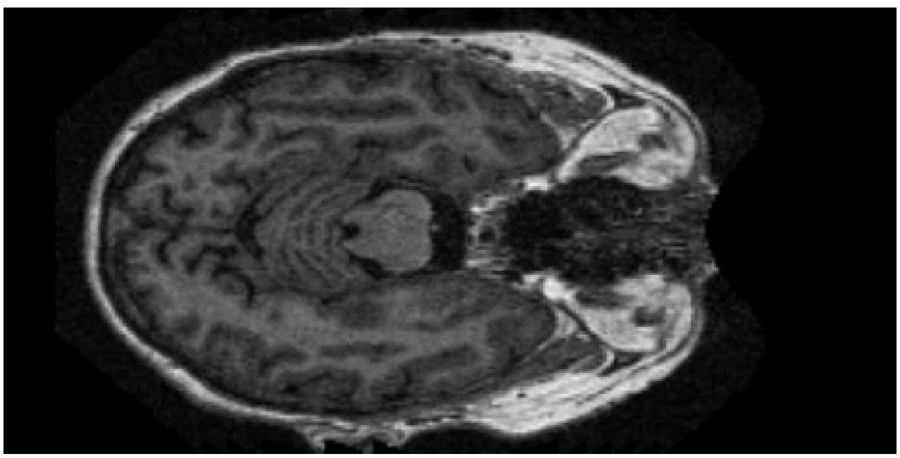


[37]:

```
img_path = '/kaggle/input/imagesoasis/Data/Non Demented/OAS1_0001_MR1_mpr-1_106.
 ↪jpg'
img = Image.open(img_path).convert('RGB')
img_resized = img.resize((224, 224))
x = np.array(img_resized) / 255.0
x = x.reshape(1, 224, 224, 3)

# Predict
res = model.predict_on_batch(x)[0]  # remove batch dimension
pred_class_idx = np.argmax(res)
```

```python
confidence = res[pred_class_idx] * 100

# Map index to class label
class_labels = list(test_gen.class_indices.keys())
predicted_class = class_labels[pred_class_idx]

# Display image and result
plt.figure(figsize=(6, 4))
plt.imshow(img)
plt.axis('off')
plt.title(f"{confidence:.2f}% Confidence This is {predicted_class}")
plt.show()

# Show probabilities for all classes
for i, (label, prob) in enumerate(zip(class_labels, res)):
    marker = "<-- Highest" if i == pred_class_idx else ""
    print(f"{label:<20}: {prob*100:.2f}% {marker}")
```

28.65% Confidence This is Non Demented

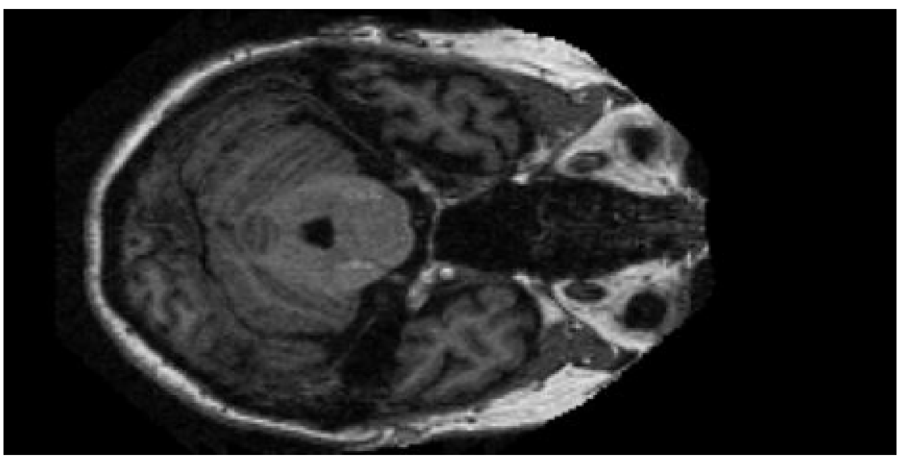


```
Mild Dementia       : 23.98%
Moderate Dementia   : 22.41%
Non Demented        : 28.65% <-- Highest
Very mild Dementia  : 24.96%
```

[36]:
```python
img_path = '/kaggle/input/imagesoasis/Data/Moderate Dementia/
↪OAS1_0308_MR1_mpr-1_116.jpg'
img = Image.open(img_path).convert('RGB')
img_resized = img.resize((224, 224))
x = np.array(img_resized) / 255.0
x = x.reshape(1, 224, 224, 3)
```

```python
# Predict
res = model.predict_on_batch(x)[0]  # remove batch dimension
pred_class_idx = np.argmax(res)
confidence = res[pred_class_idx] * 100

# Map index to class label
class_labels = list(test_gen.class_indices.keys())
predicted_class = class_labels[pred_class_idx]

# Display image and result
plt.figure(figsize=(6, 4))
plt.imshow(img)
plt.axis('off')
plt.title(f"{confidence:.2f}% Confidence This is {predicted_class}")
plt.show()

# Show probabilities for all classes
for i, (label, prob) in enumerate(zip(class_labels, res)):
    marker = "<-- Highest" if i == pred_class_idx else ""
    print(f"{label:<20}: {prob*100:.2f}% {marker}")
```

26.27% Confidence This is Moderate Dementia



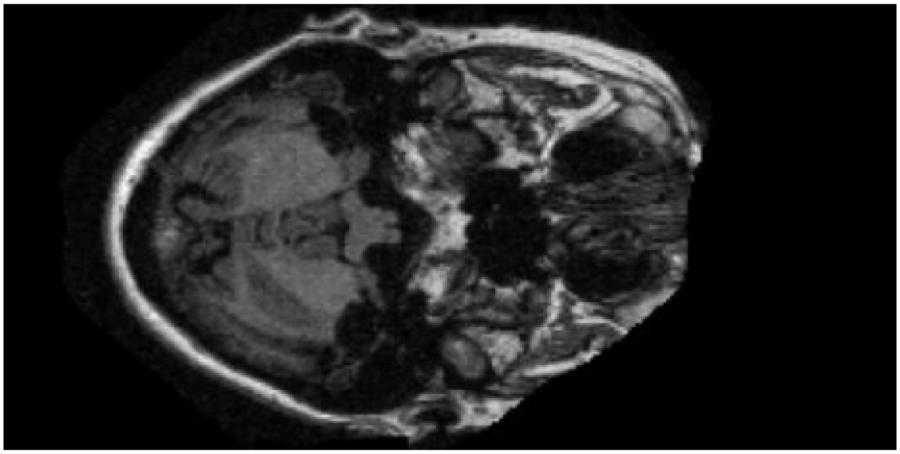
```
Mild Dementia        : 24.19%
Moderate Dementia    : 26.27% <-- Highest
Non Demented         : 25.90%
Very mild Dementia   : 23.64%
```

[32]:

```python
# Load and preprocess the image
img_path = '/kaggle/input/imagesoasis/Data/Very mild Dementia/
↪OAS1_0003_MR1_mpr-1_117.jpg'
```

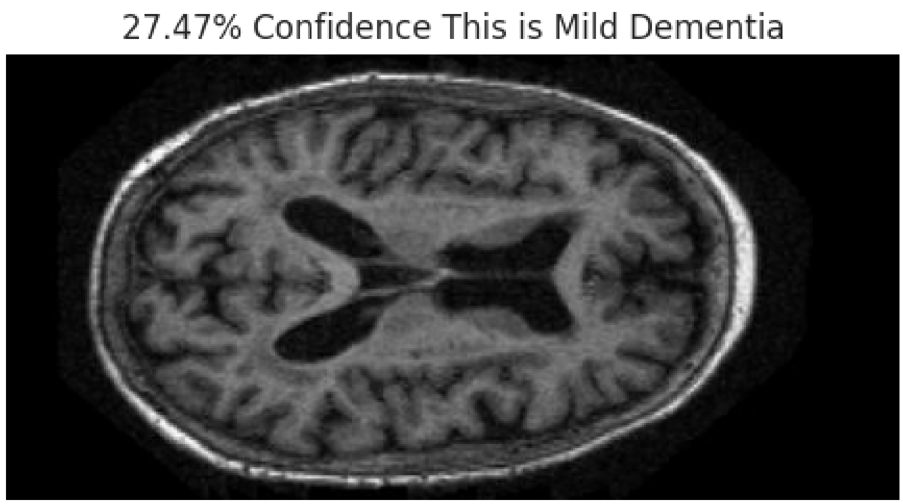
```python
img = Image.open(img_path).convert('RGB')
img_resized = img.resize((224, 224))
x = np.array(img_resized) / 255.0
x = x.reshape(1, 224, 224, 3)

# Predict
res = model.predict_on_batch(x)[0]  # remove batch dimension
pred_class_idx = np.argmax(res)
confidence = res[pred_class_idx] * 100

# Map index to class label
class_labels = list(test_gen.class_indices.keys())
predicted_class = class_labels[pred_class_idx]

# Display image and result
plt.figure(figsize=(6, 4))
plt.imshow(img)
plt.axis('off')
plt.title(f"{confidence:.2f}% Confidence This is {predicted_class}")
plt.show()

# Show probabilities for all classes
for i, (label, prob) in enumerate(zip(class_labels, res)):
    marker = "<-- Highest" if i == pred_class_idx else ""
    print(f"{label:<20}: {prob*100:.2f}% {marker}")
```

27.86% Confidence This is Very mild Dementia



```
Mild Dementia        : 24.08%
Moderate Dementia    : 20.65%
Non Demented         : 27.42%
Very mild Dementia   : 27.86% <-- Highest
```

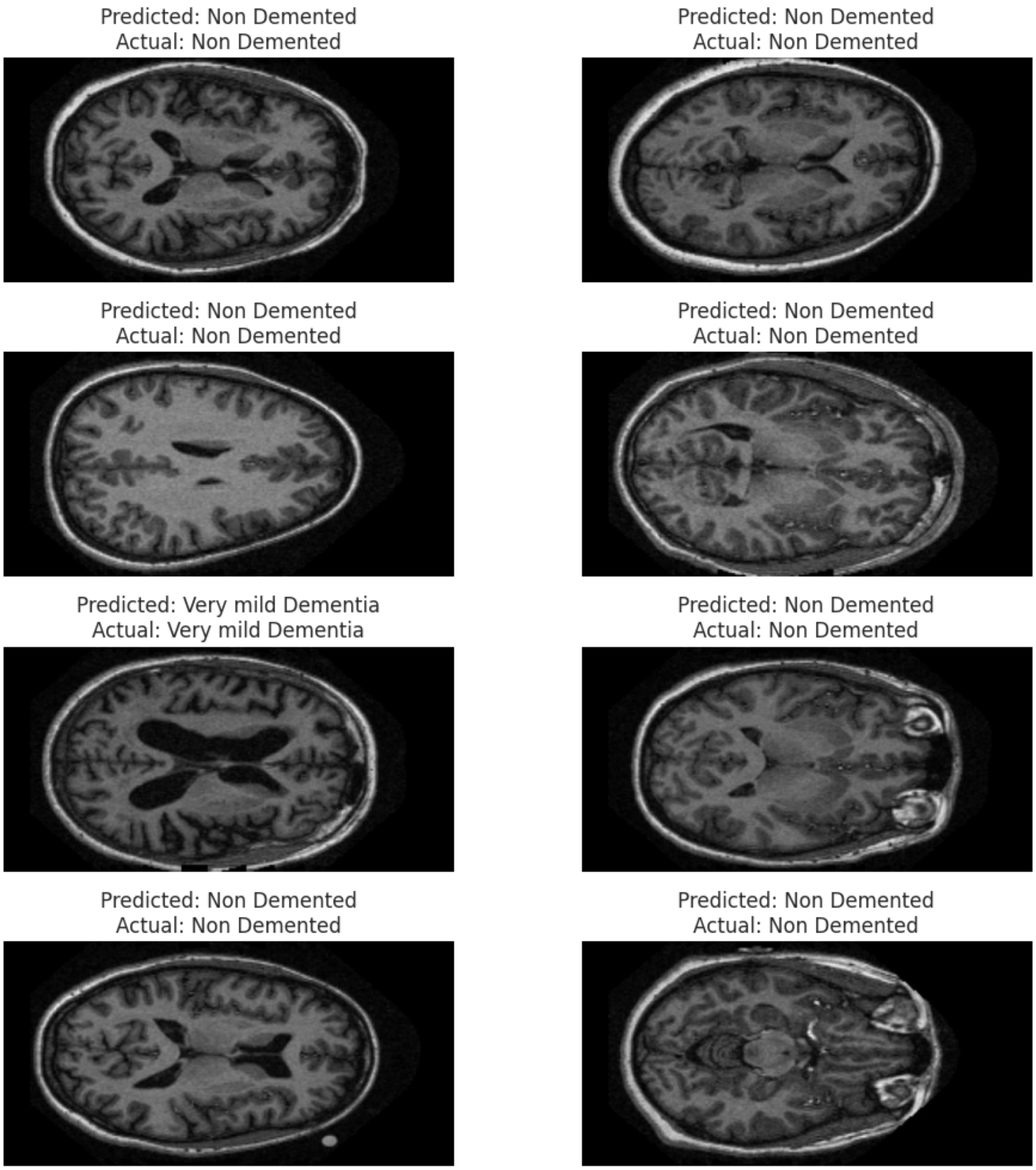
[31]:
```python
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

# Load and preprocess the image
img_path = '/kaggle/input/imagesoasis/Data/Mild Dementia/
↪OAS1_0028_MR1_mpr-1_145.jpg'
img = Image.open(img_path).convert('RGB')
img_resized = img.resize((224, 224))
x = np.array(img_resized) / 255.0
x = x.reshape(1, 224, 224, 3)

# Predict
res = model.predict_on_batch(x)[0]  # remove batch dimension
pred_class_idx = np.argmax(res)
confidence = res[pred_class_idx] * 100

# Map index to class label
class_labels = list(test_gen.class_indices.keys())
predicted_class = class_labels[pred_class_idx]

# Display image and result
plt.figure(figsize=(6, 4))
plt.imshow(img)
plt.axis('off')
plt.title(f"{confidence:.2f}% Confidence This is {predicted_class}")
plt.show()

# Show probabilities for all classes
for i, (label, prob) in enumerate(zip(class_labels, res)):
    marker = "<-- Highest" if i == pred_class_idx else ""
    print(f"{label:<20}: {prob*100:.2f}% {marker}")
```



```
Mild Dementia        : 27.47% <-- Highest
Moderate Dementia    : 22.67%
Non Demented         : 25.71%
Very mild Dementia   : 24.15%
```

[26]:

```
class_labels = list(test_gen.class_indices.keys())

# Get image filepaths (to load and show images)
filepaths = test_gen.filepaths

# Show 8 predictions with actual labels
count = 0
fig, ax = plt.subplots(4, 2)
fig.set_size_inches(10, 10)

for i in range(4):
    for j in range(2):
        index = count
        img = plt.imread(filepaths[index])

        predicted_label = class_labels[y_pred[index]]
        actual_label = class_labels[y_true[index]]

        ax[i, j].imshow(img)
        ax[i, j].set_title(f"Predicted: {predicted_label}\nActual: 
 ↪{actual_label}")
        ax[i, j].axis('off')

        count += 1

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


Predicted: Non Demented
Actual: Non Demented
Predicted: Non Demented
Actual: Non Demented
Predicted: Non Demented
Actual: Non Demented
Predicted: Non Demented
Actual: Non Demented
Predicted: Very mild Dementia
Actual: Very mild Dementia
Predicted: Non Demented
Actual: Non Demented
Predicted: Non Demented
Actual: Non Demented
Predicted: Non Demented
Actual: Non Demented

[ ]: