isease-classification-sequential-2

March 28, 2024

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt

import os
from distutils.dir_util import copy_tree, remove_tree

from PIL import Image
from random import randint

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import matthews_corrcoef as MCC
from sklearn.metrics import balanced_accuracy_score as BAS
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow_addons as tfa
from keras.utils.vis_utils import plot_model
from tensorflow.keras import Sequential, Input
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import Conv2D, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
from tensorflow.keras.layers import SeparableConv2D, BatchNormalization,␣
↪MaxPool2D



print("TensorFlow Version:", tf.__version__)
```

```
TensorFlow Version: 2.4.1
```

[2]:
```
base_dir = "/kaggle/input/alzheimers-dataset-4-class-of-images/Alzheimer_s␣
↪Dataset/"
```

```
root_dir = "./"
test_dir = base_dir + "test/"
train_dir = base_dir + "train/"
work_dir = root_dir + "dataset/"

if os.path.exists(work_dir):
    remove_tree(work_dir)


os.mkdir(work_dir)
copy_tree(train_dir, work_dir)
copy_tree(test_dir, work_dir)
print("Working Directory Contents:", os.listdir(work_dir))
```

```
Working Directory Contents: ['MildDemented', 'NonDemented', 'ModerateDemented',
'VeryMildDemented']
```

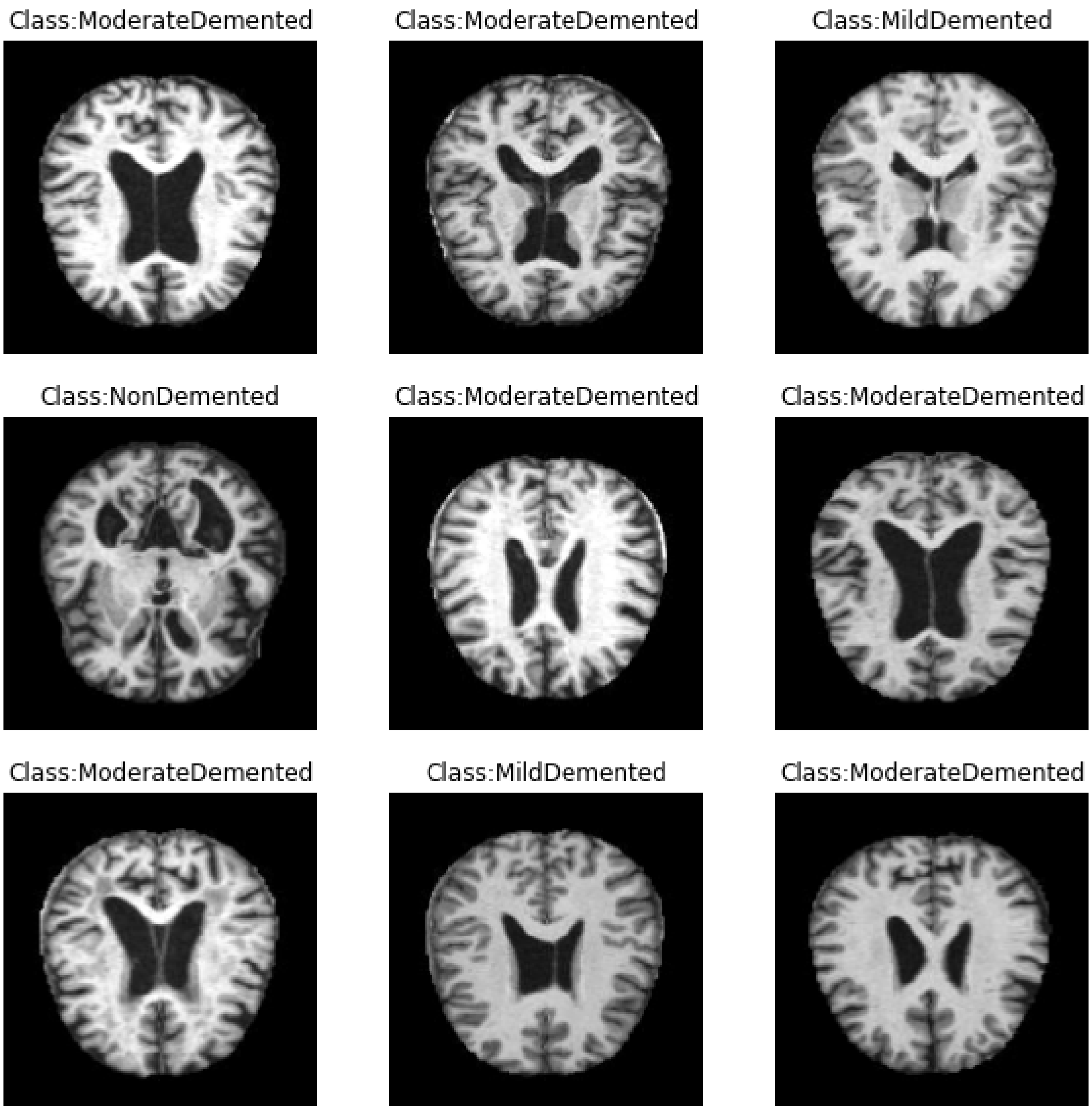
[3]:
```
WORK_DIR = './dataset/'

CLASSES = [ 'NonDemented',
            'VeryMildDemented',
            'MildDemented',
            'ModerateDemented']

IMG_SIZE = 176
IMAGE_SIZE = [176, 176]
DIM = (IMG_SIZE, IMG_SIZE)
```

[4]:
```
#Performing Image Augmentation to have more data samples

ZOOM = [.99, 1.01]
BRIGHT_RANGE = [0.8, 1.2]
HORZ_FLIP = True
FILL_MODE = "constant"
DATA_FORMAT = "channels_last"

work_dr = IDG(rescale = 1./255, brightness_range=BRIGHT_RANGE, zoom_range=ZOOM,
 ↪data_format=DATA_FORMAT, fill_mode=FILL_MODE, horizontal_flip=HORZ_FLIP)

train_data_gen = work_dr.flow_from_directory(directory=WORK_DIR,
 ↪target_size=DIM, batch_size=6500, shuffle=False)
```

```
Found 6400 images belonging to 4 classes.
```

[5]:
```
def show_images(generator,y_pred=None):
    """
    Input: An image generator,predicted labels (optional)
```

```
    Output: Displays a grid of 9 images with lables
    """

    # get image lables
    labels =dict(zip([0,1,2,3], CLASSES))

    # get a batch of images
    x,y = generator.next()

    # display a grid of 9 images
    plt.figure(figsize=(10, 10))
    if y_pred is None:
        for i in range(9):
            ax = plt.subplot(3, 3, i + 1)
            idx = randint(0, 6400)
            plt.imshow(x[idx])
            plt.axis("off")
            plt.title("Class:{}".format(labels[np.argmax(y[idx])]))

    else:
        for i in range(9):
            ax = plt.subplot(3, 3, i + 1)
            plt.imshow(x[i])
            plt.axis("off")
            plt.title("Actual:{} \nPredicted:{}".format(labels[np.
 ↪argmax(y[i])],labels[y_pred[i]]))

# Display Train Images
show_images(train_data_gen)
```



[6]:
```
#Retrieving the data from the ImageDataGenerator iterator

train_data, train_labels = train_data_gen.next()
```

[7]:
```
#Getting to know the dimensions of our dataset

print(train_data.shape, train_labels.shape)
```

```
(6400, 176, 176, 3) (6400, 4)
```

[8]:
```
#Performing over-sampling of the data, since the classes are imbalanced

sm = SMOTE(random_state=42)
```

```
train_data, train_labels = sm.fit_resample(train_data.reshape(-1, IMG_SIZE *␣
↪IMG_SIZE * 3), train_labels)

train_data = train_data.reshape(-1, IMG_SIZE, IMG_SIZE, 3)

print(train_data.shape, train_labels.shape)
```

```
(12800, 176, 176, 3) (12800, 4)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72:
FutureWarning: Pass classes=[0 1 2 3] as keyword args. From version 1.0
(renaming of 0.25) passing these as positional arguments will result in an error
  "will result in an error", FutureWarning)
```

[9]:
```
#Splitting the data into train, test, and validation sets

train_data, test_data, train_labels, test_labels = train_test_split(train_data,␣
↪train_labels, test_size = 0.2, random_state=42)
train_data, val_data, train_labels, val_labels = train_test_split(train_data,␣
↪train_labels, test_size = 0.2, random_state=42)
```

[10]:
```
def conv_block(filters, act='relu'):
    """Defining a Convolutional NN block for a Sequential CNN model. """

    block = Sequential()
    block.add(Conv2D(filters, 3, activation=act, padding='same'))
    block.add(Conv2D(filters, 3, activation=act, padding='same'))
    block.add(BatchNormalization())
    block.add(MaxPool2D())

    return block
```

[11]:
```
def dense_block(units, dropout_rate, act='relu'):
    """Defining a Dense NN block for a Sequential CNN model. """

    block = Sequential()
    block.add(Dense(units, activation=act))
    block.add(BatchNormalization())
    block.add(Dropout(dropout_rate))

    return block
```

[12]:
```
def construct_model(act='relu'):
    """Constructing a Sequential CNN architecture for performing the␣
↪classification task. """
```

```
    model = Sequential([
        Input(shape=(*IMAGE_SIZE, 3)),
        Conv2D(16, 3, activation=act, padding='same'),
        Conv2D(16, 3, activation=act, padding='same'),
        MaxPool2D(),
        conv_block(32),
        conv_block(64),
        conv_block(128),
        Dropout(0.2),
        conv_block(256),
        Dropout(0.2),
        Flatten(),
        dense_block(512, 0.7),
        dense_block(128, 0.5),
        dense_block(64, 0.3),
        Dense(4, activation='softmax')
    ], name = "cnn_model")

    return model
```

[13]:
```
#Defining a custom callback function to stop training our model when accuracy␣
↪goes above 99%

class MyCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('val_acc') > 0.99:
            print("\nReached accuracy threshold! Terminating training.")
            self.model.stop_training = True

my_callback = MyCallback()

#EarlyStopping callback to make sure model is always learning
early_stopping = EarlyStopping(monitor='val_loss', patience=2)
```

[14]:
```
#Defining other parameters for our CNN model

model = construct_model()

METRICS = [tf.keras.metrics.CategoricalAccuracy(name='acc'),
           tf.keras.metrics.AUC(name='auc'),
           tfa.metrics.F1Score(num_classes=4)]

CALLBACKS = [my_callback]

model.compile(optimizer='adam',
              loss=tf.losses.CategoricalCrossentropy(),
              metrics=METRICS)
```

```
model.summary()
```

```
Model: "cnn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 176, 176, 16)      448
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 176, 176, 16)      2320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 88, 88, 16)        0
_________________________________________________________________
sequential (Sequential)      (None, 44, 44, 32)        14016
_________________________________________________________________
sequential_1 (Sequential)    (None, 22, 22, 64)        55680
_________________________________________________________________
sequential_2 (Sequential)    (None, 11, 11, 128)       221952
_________________________________________________________________
dropout (Dropout)            (None, 11, 11, 128)       0
_________________________________________________________________
sequential_3 (Sequential)    (None, 5, 5, 256)         886272
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
sequential_4 (Sequential)    (None, 512)               3279360
_________________________________________________________________
sequential_5 (Sequential)    (None, 128)               66176
_________________________________________________________________
sequential_6 (Sequential)    (None, 64)                8512
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 260
=================================================================
Total params: 4,534,996
Trainable params: 4,532,628
Non-trainable params: 2,368
_________________________________________________________________
```

[15]:

```
#Fit the training data to the model and validate it using the validation data
EPOCHS = 100

history = model.fit(train_data, train_labels, validation_data=(val_data,␣
↪val_labels), callbacks=CALLBACKS, epochs=EPOCHS)
```

```
Epoch 1/100
256/256 [==============================] - 23s 55ms/step - loss: 1.8004 - acc:
```

```
0.2737 - auc: 0.5321 - f1_score: 0.2737 - val_loss: 1.6713 - val_acc: 0.2471 -
val_auc: 0.4970 - val_f1_score: 0.1213
Epoch 2/100
256/256 [==============================] - 13s 49ms/step - loss: 1.4250 - acc:
0.3321 - auc: 0.5971 - f1_score: 0.3307 - val_loss: 2.7988 - val_acc: 0.2568 -
val_auc: 0.5768 - val_f1_score: 0.1027
Epoch 3/100
256/256 [==============================] - 13s 49ms/step - loss: 1.0525 - acc:
0.5263 - auc: 0.7979 - f1_score: 0.5210 - val_loss: 1.6007 - val_acc: 0.3398 -
val_auc: 0.6151 - val_f1_score: 0.2682
Epoch 4/100
256/256 [==============================] - 13s 50ms/step - loss: 0.8075 - acc:
0.6314 - auc: 0.8812 - f1_score: 0.6164 - val_loss: 0.8203 - val_acc: 0.6265 -
val_auc: 0.8751 - val_f1_score: 0.6104
Epoch 5/100
256/256 [==============================] - 13s 50ms/step - loss: 0.7167 - acc:
0.6691 - auc: 0.9052 - f1_score: 0.6555 - val_loss: 2.2500 - val_acc: 0.3145 -
val_auc: 0.5723 - val_f1_score: 0.2327
Epoch 6/100
256/256 [==============================] - 13s 51ms/step - loss: 0.6619 - acc:
0.7046 - auc: 0.9192 - f1_score: 0.7015 - val_loss: 0.5792 - val_acc: 0.7500 -
val_auc: 0.9401 - val_f1_score: 0.7422
Epoch 7/100
256/256 [==============================] - 13s 51ms/step - loss: 0.5958 - acc:
0.7363 - auc: 0.9350 - f1_score: 0.7324 - val_loss: 1.0391 - val_acc: 0.5815 -
val_auc: 0.8315 - val_f1_score: 0.5865
Epoch 8/100
256/256 [==============================] - 13s 52ms/step - loss: 0.5959 - acc:
0.7432 - auc: 0.9368 - f1_score: 0.7426 - val_loss: 1.0476 - val_acc: 0.5464 -
val_auc: 0.8276 - val_f1_score: 0.5357
Epoch 9/100
256/256 [==============================] - 13s 52ms/step - loss: 0.5397 - acc:
0.7667 - auc: 0.9471 - f1_score: 0.7654 - val_loss: 0.4883 - val_acc: 0.7812 -
val_auc: 0.9561 - val_f1_score: 0.7794
Epoch 10/100
256/256 [==============================] - 13s 52ms/step - loss: 0.4880 - acc:
0.7864 - auc: 0.9559 - f1_score: 0.7867 - val_loss: 1.6974 - val_acc: 0.5093 -
val_auc: 0.7485 - val_f1_score: 0.4413
Epoch 11/100
256/256 [==============================] - 13s 51ms/step - loss: 0.4846 - acc:
0.7894 - auc: 0.9567 - f1_score: 0.7872 - val_loss: 1.1824 - val_acc: 0.6421 -
val_auc: 0.8877 - val_f1_score: 0.6106
Epoch 12/100
256/256 [==============================] - 13s 51ms/step - loss: 0.4496 - acc:
0.8060 - auc: 0.9625 - f1_score: 0.8044 - val_loss: 0.6092 - val_acc: 0.7666 -
val_auc: 0.9472 - val_f1_score: 0.7510
Epoch 13/100
256/256 [==============================] - 13s 51ms/step - loss: 0.4040 - acc:
```

```
0.8319 - auc: 0.9707 - f1_score: 0.8310 - val_loss: 0.4064 - val_acc: 0.8047 -
val_auc: 0.9689 - val_f1_score: 0.7931
Epoch 14/100
256/256 [==============================] - 13s 51ms/step - loss: 0.3385 - acc:
0.8616 - auc: 0.9787 - f1_score: 0.8591 - val_loss: 0.5648 - val_acc: 0.7700 -
val_auc: 0.9452 - val_f1_score: 0.7704
Epoch 15/100
256/256 [==============================] - 13s 51ms/step - loss: 0.3226 - acc:
0.8676 - auc: 0.9806 - f1_score: 0.8663 - val_loss: 1.9555 - val_acc: 0.5234 -
val_auc: 0.8089 - val_f1_score: 0.4310
Epoch 16/100
256/256 [==============================] - 13s 51ms/step - loss: 0.3050 - acc:
0.8781 - auc: 0.9823 - f1_score: 0.8769 - val_loss: 0.8674 - val_acc: 0.6982 -
val_auc: 0.9293 - val_f1_score: 0.6433
Epoch 17/100
256/256 [==============================] - 13s 51ms/step - loss: 0.2490 - acc:
0.9027 - auc: 0.9881 - f1_score: 0.9027 - val_loss: 0.3560 - val_acc: 0.8633 -
val_auc: 0.9820 - val_f1_score: 0.8584
Epoch 18/100
256/256 [==============================] - 13s 51ms/step - loss: 0.2023 - acc:
0.9263 - auc: 0.9917 - f1_score: 0.9254 - val_loss: 2.3461 - val_acc: 0.3975 -
val_auc: 0.7062 - val_f1_score: 0.3268
Epoch 19/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1911 - acc:
0.9330 - auc: 0.9926 - f1_score: 0.9332 - val_loss: 0.3677 - val_acc: 0.8608 -
val_auc: 0.9786 - val_f1_score: 0.8584
Epoch 20/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1843 - acc:
0.9342 - auc: 0.9934 - f1_score: 0.9334 - val_loss: 0.2853 - val_acc: 0.8979 -
val_auc: 0.9868 - val_f1_score: 0.8961
Epoch 21/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1652 - acc:
0.9411 - auc: 0.9942 - f1_score: 0.9410 - val_loss: 0.3495 - val_acc: 0.8828 -
val_auc: 0.9818 - val_f1_score: 0.8833
Epoch 22/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1322 - acc:
0.9534 - auc: 0.9961 - f1_score: 0.9534 - val_loss: 0.3302 - val_acc: 0.8877 -
val_auc: 0.9830 - val_f1_score: 0.8886
Epoch 23/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1199 - acc:
0.9572 - auc: 0.9966 - f1_score: 0.9571 - val_loss: 0.8247 - val_acc: 0.7896 -
val_auc: 0.9399 - val_f1_score: 0.7841
Epoch 24/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1231 - acc:
0.9586 - auc: 0.9961 - f1_score: 0.9584 - val_loss: 0.2141 - val_acc: 0.9233 -
val_auc: 0.9920 - val_f1_score: 0.9231
Epoch 25/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1011 - acc:
```

```
0.9662 - auc: 0.9977 - f1_score: 0.9660 - val_loss: 0.6412 - val_acc: 0.8159 -
val_auc: 0.9583 - val_f1_score: 0.8014
Epoch 26/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1094 - acc:
0.9643 - auc: 0.9971 - f1_score: 0.9643 - val_loss: 0.2493 - val_acc: 0.9170 -
val_auc: 0.9902 - val_f1_score: 0.9160
Epoch 27/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0896 - acc:
0.9669 - auc: 0.9982 - f1_score: 0.9668 - val_loss: 0.5295 - val_acc: 0.8286 -
val_auc: 0.9687 - val_f1_score: 0.8225
Epoch 28/100
256/256 [==============================] - 13s 51ms/step - loss: 0.1001 - acc:
0.9646 - auc: 0.9972 - f1_score: 0.9648 - val_loss: 0.3778 - val_acc: 0.8853 -
val_auc: 0.9805 - val_f1_score: 0.8820
Epoch 29/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0986 - acc:
0.9687 - auc: 0.9971 - f1_score: 0.9685 - val_loss: 0.3089 - val_acc: 0.9033 -
val_auc: 0.9845 - val_f1_score: 0.9013
Epoch 30/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0767 - acc:
0.9705 - auc: 0.9987 - f1_score: 0.9708 - val_loss: 0.5289 - val_acc: 0.8496 -
val_auc: 0.9686 - val_f1_score: 0.8403
Epoch 31/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0695 - acc:
0.9790 - auc: 0.9982 - f1_score: 0.9789 - val_loss: 0.2920 - val_acc: 0.9136 -
val_auc: 0.9864 - val_f1_score: 0.9119
Epoch 32/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0705 - acc:
0.9768 - auc: 0.9985 - f1_score: 0.9766 - val_loss: 0.3391 - val_acc: 0.9033 -
val_auc: 0.9828 - val_f1_score: 0.8998
Epoch 33/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0656 - acc:
0.9800 - auc: 0.9985 - f1_score: 0.9799 - val_loss: 1.2103 - val_acc: 0.7383 -
val_auc: 0.9036 - val_f1_score: 0.7300
Epoch 34/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0620 - acc:
0.9798 - auc: 0.9985 - f1_score: 0.9799 - val_loss: 0.7214 - val_acc: 0.8228 -
val_auc: 0.9495 - val_f1_score: 0.8189
Epoch 35/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0696 - acc:
0.9765 - auc: 0.9987 - f1_score: 0.9764 - val_loss: 0.4714 - val_acc: 0.8784 -
val_auc: 0.9711 - val_f1_score: 0.8720
Epoch 36/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0575 - acc:
0.9814 - auc: 0.9986 - f1_score: 0.9814 - val_loss: 0.7210 - val_acc: 0.8198 -
val_auc: 0.9489 - val_f1_score: 0.8091
Epoch 37/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0631 - acc:
```

```
0.9804 - auc: 0.9987 - f1_score: 0.9802 - val_loss: 0.2202 - val_acc: 0.9355 -
val_auc: 0.9900 - val_f1_score: 0.9351
Epoch 38/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0979 - acc:
0.9676 - auc: 0.9967 - f1_score: 0.9677 - val_loss: 0.2414 - val_acc: 0.9136 -
val_auc: 0.9898 - val_f1_score: 0.9123
Epoch 39/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0725 - acc:
0.9763 - auc: 0.9986 - f1_score: 0.9763 - val_loss: 0.2160 - val_acc: 0.9375 -
val_auc: 0.9886 - val_f1_score: 0.9375
Epoch 40/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0511 - acc:
0.9817 - auc: 0.9993 - f1_score: 0.9815 - val_loss: 0.2085 - val_acc: 0.9346 -
val_auc: 0.9907 - val_f1_score: 0.9347
Epoch 41/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0359 - acc:
0.9877 - auc: 0.9994 - f1_score: 0.9877 - val_loss: 0.5973 - val_acc: 0.8540 -
val_auc: 0.9605 - val_f1_score: 0.8555
Epoch 42/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0381 - acc:
0.9880 - auc: 0.9994 - f1_score: 0.9879 - val_loss: 0.2313 - val_acc: 0.9365 -
val_auc: 0.9895 - val_f1_score: 0.9367
Epoch 43/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0378 - acc:
0.9877 - auc: 0.9996 - f1_score: 0.9876 - val_loss: 0.4954 - val_acc: 0.8794 -
val_auc: 0.9677 - val_f1_score: 0.8761
Epoch 44/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0646 - acc:
0.9782 - auc: 0.9986 - f1_score: 0.9782 - val_loss: 0.2919 - val_acc: 0.9219 -
val_auc: 0.9850 - val_f1_score: 0.9207
Epoch 45/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0505 - acc:
0.9855 - auc: 0.9991 - f1_score: 0.9855 - val_loss: 0.2164 - val_acc: 0.9395 -
val_auc: 0.9895 - val_f1_score: 0.9395
Epoch 46/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0284 - acc:
0.9915 - auc: 0.9995 - f1_score: 0.9914 - val_loss: 0.5887 - val_acc: 0.8472 -
val_auc: 0.9596 - val_f1_score: 0.8423
Epoch 47/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0243 - acc:
0.9927 - auc: 0.9997 - f1_score: 0.9927 - val_loss: 0.3140 - val_acc: 0.9067 -
val_auc: 0.9842 - val_f1_score: 0.9048
Epoch 48/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0662 - acc:
0.9809 - auc: 0.9980 - f1_score: 0.9809 - val_loss: 0.2009 - val_acc: 0.9458 -
val_auc: 0.9890 - val_f1_score: 0.9458
Epoch 49/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0363 - acc:
```

```
0.9877 - auc: 0.9997 - f1_score: 0.9877 - val_loss: 0.2108 - val_acc: 0.9375 -
val_auc: 0.9904 - val_f1_score: 0.9377
Epoch 50/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0276 - acc:
0.9918 - auc: 0.9993 - f1_score: 0.9918 - val_loss: 0.9958 - val_acc: 0.7876 -
val_auc: 0.9227 - val_f1_score: 0.7801
Epoch 51/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0376 - acc:
0.9888 - auc: 0.9996 - f1_score: 0.9888 - val_loss: 0.2587 - val_acc: 0.9297 -
val_auc: 0.9867 - val_f1_score: 0.9287
Epoch 52/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0343 - acc:
0.9892 - auc: 0.9993 - f1_score: 0.9893 - val_loss: 0.7595 - val_acc: 0.8032 -
val_auc: 0.9456 - val_f1_score: 0.7983
Epoch 53/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0711 - acc:
0.9789 - auc: 0.9980 - f1_score: 0.9788 - val_loss: 0.2659 - val_acc: 0.9253 -
val_auc: 0.9849 - val_f1_score: 0.9245
Epoch 54/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0433 - acc:
0.9858 - auc: 0.9995 - f1_score: 0.9859 - val_loss: 2.8259 - val_acc: 0.5425 -
val_auc: 0.7920 - val_f1_score: 0.4884
Epoch 55/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0412 - acc:
0.9882 - auc: 0.9993 - f1_score: 0.9883 - val_loss: 0.2718 - val_acc: 0.9307 -
val_auc: 0.9849 - val_f1_score: 0.9303
Epoch 56/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0406 - acc:
0.9875 - auc: 0.9987 - f1_score: 0.9875 - val_loss: 0.1802 - val_acc: 0.9482 -
val_auc: 0.9901 - val_f1_score: 0.9479
Epoch 57/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0243 - acc:
0.9920 - auc: 0.9996 - f1_score: 0.9920 - val_loss: 0.5178 - val_acc: 0.8696 -
val_auc: 0.9680 - val_f1_score: 0.8671
Epoch 58/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0389 - acc:
0.9879 - auc: 0.9989 - f1_score: 0.9880 - val_loss: 0.4408 - val_acc: 0.8872 -
val_auc: 0.9758 - val_f1_score: 0.8839
Epoch 59/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0432 - acc:
0.9858 - auc: 0.9992 - f1_score: 0.9858 - val_loss: 0.5958 - val_acc: 0.8691 -
val_auc: 0.9607 - val_f1_score: 0.8609
Epoch 60/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0356 - acc:
0.9865 - auc: 0.9997 - f1_score: 0.9864 - val_loss: 0.4801 - val_acc: 0.8887 -
val_auc: 0.9693 - val_f1_score: 0.8860
Epoch 61/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0286 - acc:
```

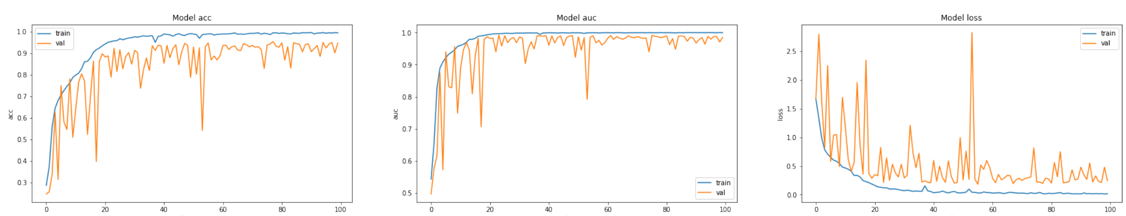
```
0.9883 - auc: 0.9998 - f1_score: 0.9884 - val_loss: 0.2878 - val_acc: 0.9365 -
val_auc: 0.9827 - val_f1_score: 0.9364
Epoch 62/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0309 - acc:
0.9900 - auc: 0.9993 - f1_score: 0.9899 - val_loss: 0.2086 - val_acc: 0.9380 -
val_auc: 0.9908 - val_f1_score: 0.9373
Epoch 63/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0283 - acc:
0.9914 - auc: 0.9996 - f1_score: 0.9914 - val_loss: 0.3530 - val_acc: 0.9175 -
val_auc: 0.9787 - val_f1_score: 0.9157
Epoch 64/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0310 - acc:
0.9906 - auc: 0.9995 - f1_score: 0.9908 - val_loss: 0.2598 - val_acc: 0.9302 -
val_auc: 0.9873 - val_f1_score: 0.9300
Epoch 65/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0160 - acc:
0.9951 - auc: 0.9999 - f1_score: 0.9951 - val_loss: 0.2890 - val_acc: 0.9341 -
val_auc: 0.9827 - val_f1_score: 0.9331
Epoch 66/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0298 - acc:
0.9901 - auc: 0.9997 - f1_score: 0.9901 - val_loss: 0.3373 - val_acc: 0.9155 -
val_auc: 0.9793 - val_f1_score: 0.9138
Epoch 67/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0288 - acc:
0.9928 - auc: 0.9994 - f1_score: 0.9928 - val_loss: 0.3290 - val_acc: 0.9126 -
val_auc: 0.9827 - val_f1_score: 0.9128
Epoch 68/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0319 - acc:
0.9899 - auc: 0.9992 - f1_score: 0.9899 - val_loss: 0.1925 - val_acc: 0.9453 -
val_auc: 0.9900 - val_f1_score: 0.9451
Epoch 69/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0307 - acc:
0.9897 - auc: 0.9994 - f1_score: 0.9896 - val_loss: 0.2590 - val_acc: 0.9395 -
val_auc: 0.9855 - val_f1_score: 0.9386
Epoch 70/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0274 - acc:
0.9915 - auc: 0.9998 - f1_score: 0.9916 - val_loss: 0.2854 - val_acc: 0.9297 -
val_auc: 0.9841 - val_f1_score: 0.9298
Epoch 71/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0233 - acc:
0.9929 - auc: 0.9998 - f1_score: 0.9928 - val_loss: 0.2493 - val_acc: 0.9365 -
val_auc: 0.9872 - val_f1_score: 0.9363
Epoch 72/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0320 - acc:
0.9921 - auc: 0.9991 - f1_score: 0.9921 - val_loss: 0.2800 - val_acc: 0.9282 -
val_auc: 0.9846 - val_f1_score: 0.9270
Epoch 73/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0157 - acc:
```

```
0.9950 - auc: 0.9998 - f1_score: 0.9951 - val_loss: 0.2947 - val_acc: 0.9307 -
val_auc: 0.9817 - val_f1_score: 0.9309
Epoch 74/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0263 - acc:
0.9896 - auc: 0.9995 - f1_score: 0.9893 - val_loss: 0.3077 - val_acc: 0.9175 -
val_auc: 0.9836 - val_f1_score: 0.9160
Epoch 75/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0262 - acc:
0.9919 - auc: 0.9997 - f1_score: 0.9919 - val_loss: 0.8139 - val_acc: 0.8296 -
val_auc: 0.9400 - val_f1_score: 0.8270
Epoch 76/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0192 - acc:
0.9925 - auc: 0.9998 - f1_score: 0.9925 - val_loss: 0.2215 - val_acc: 0.9385 -
val_auc: 0.9912 - val_f1_score: 0.9380
Epoch 77/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0422 - acc:
0.9855 - auc: 0.9994 - f1_score: 0.9856 - val_loss: 0.2192 - val_acc: 0.9429 -
val_auc: 0.9885 - val_f1_score: 0.9424
Epoch 78/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0215 - acc:
0.9942 - auc: 0.9996 - f1_score: 0.9941 - val_loss: 0.1965 - val_acc: 0.9546 -
val_auc: 0.9870 - val_f1_score: 0.9545
Epoch 79/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0124 - acc:
0.9960 - auc: 0.9998 - f1_score: 0.9960 - val_loss: 0.2881 - val_acc: 0.9341 -
val_auc: 0.9841 - val_f1_score: 0.9329
Epoch 80/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0209 - acc:
0.9926 - auc: 0.9997 - f1_score: 0.9926 - val_loss: 0.2738 - val_acc: 0.9277 -
val_auc: 0.9863 - val_f1_score: 0.9268
Epoch 81/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0163 - acc:
0.9952 - auc: 0.9998 - f1_score: 0.9951 - val_loss: 0.2016 - val_acc: 0.9487 -
val_auc: 0.9896 - val_f1_score: 0.9484
Epoch 82/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0168 - acc:
0.9947 - auc: 0.9998 - f1_score: 0.9947 - val_loss: 0.5604 - val_acc: 0.8687 -
val_auc: 0.9623 - val_f1_score: 0.8674
Epoch 83/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0282 - acc:
0.9906 - auc: 0.9994 - f1_score: 0.9906 - val_loss: 0.3112 - val_acc: 0.9312 -
val_auc: 0.9812 - val_f1_score: 0.9299
Epoch 84/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0322 - acc:
0.9917 - auc: 0.9990 - f1_score: 0.9917 - val_loss: 0.7490 - val_acc: 0.8315 -
val_auc: 0.9482 - val_f1_score: 0.8284
Epoch 85/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0188 - acc:
```

```
0.9945 - auc: 0.9997 - f1_score: 0.9945 - val_loss: 0.2061 - val_acc: 0.9492 -
val_auc: 0.9887 - val_f1_score: 0.9493
Epoch 86/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0163 - acc:
0.9950 - auc: 0.9998 - f1_score: 0.9950 - val_loss: 0.2150 - val_acc: 0.9443 -
val_auc: 0.9896 - val_f1_score: 0.9438
Epoch 87/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0165 - acc:
0.9943 - auc: 0.9998 - f1_score: 0.9940 - val_loss: 0.2274 - val_acc: 0.9414 -
val_auc: 0.9875 - val_f1_score: 0.9411
Epoch 88/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0146 - acc:
0.9956 - auc: 0.9999 - f1_score: 0.9956 - val_loss: 0.4331 - val_acc: 0.9067 -
val_auc: 0.9733 - val_f1_score: 0.9058
Epoch 89/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0125 - acc:
0.9958 - auc: 0.9999 - f1_score: 0.9959 - val_loss: 0.2582 - val_acc: 0.9409 -
val_auc: 0.9850 - val_f1_score: 0.9413
Epoch 90/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0201 - acc:
0.9944 - auc: 0.9998 - f1_score: 0.9944 - val_loss: 0.2714 - val_acc: 0.9434 -
val_auc: 0.9831 - val_f1_score: 0.9425
Epoch 91/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0087 - acc:
0.9973 - auc: 0.9999 - f1_score: 0.9973 - val_loss: 0.4787 - val_acc: 0.9072 -
val_auc: 0.9672 - val_f1_score: 0.9059
Epoch 92/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0219 - acc:
0.9924 - auc: 0.9998 - f1_score: 0.9923 - val_loss: 0.3455 - val_acc: 0.9219 -
val_auc: 0.9780 - val_f1_score: 0.9194
Epoch 93/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0209 - acc:
0.9927 - auc: 0.9999 - f1_score: 0.9927 - val_loss: 0.2662 - val_acc: 0.9365 -
val_auc: 0.9844 - val_f1_score: 0.9361
Epoch 94/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0228 - acc:
0.9933 - auc: 0.9995 - f1_score: 0.9933 - val_loss: 0.5512 - val_acc: 0.8857 -
val_auc: 0.9640 - val_f1_score: 0.8822
Epoch 95/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0133 - acc:
0.9969 - auc: 0.9996 - f1_score: 0.9969 - val_loss: 0.2214 - val_acc: 0.9502 -
val_auc: 0.9879 - val_f1_score: 0.9501
Epoch 96/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0151 - acc:
0.9948 - auc: 1.0000 - f1_score: 0.9948 - val_loss: 0.3272 - val_acc: 0.9248 -
val_auc: 0.9795 - val_f1_score: 0.9238
Epoch 97/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0110 - acc:
```

```
0.9957 - auc: 0.9999 - f1_score: 0.9956 - val_loss: 0.2312 - val_acc: 0.9429 -
val_auc: 0.9875 - val_f1_score: 0.9431
Epoch 98/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0132 - acc:
0.9955 - auc: 0.9996 - f1_score: 0.9955 - val_loss: 0.2145 - val_acc: 0.9502 -
val_auc: 0.9875 - val_f1_score: 0.9501
Epoch 99/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0092 - acc:
0.9972 - auc: 1.0000 - f1_score: 0.9972 - val_loss: 0.4792 - val_acc: 0.9019 -
val_auc: 0.9705 - val_f1_score: 0.9004
Epoch 100/100
256/256 [==============================] - 13s 51ms/step - loss: 0.0167 - acc:
0.9952 - auc: 0.9998 - f1_score: 0.9952 - val_loss: 0.2442 - val_acc: 0.9482 -
val_auc: 0.9849 - val_f1_score: 0.9481
```

[16]:
```
#Plotting the trend of the metrics during training

fig, ax = plt.subplots(1, 3, figsize = (30, 5))
ax = ax.ravel()

for i, metric in enumerate(["acc", "auc", "loss"]):
    ax[i].plot(history.history[metric])
    ax[i].plot(history.history["val_" + metric])
    ax[i].set_title("Model {}".format(metric))
    ax[i].set_xlabel("Epochs")
    ax[i].set_ylabel(metric)
    ax[i].legend(["train", "val"])
```



[17]:
```
#Evaluating the model on the data

#train_scores = model.evaluate(train_data, train_labels)
#val_scores = model.evaluate(val_data, val_labels)
test_scores = model.evaluate(test_data, test_labels)

#print("Training Accuracy: %.2f%%"%(train_scores[1] * 100))
#print("Validation Accuracy: %.2f%%"%(val_scores[1] * 100))
print("Testing Accuracy: %.2f%%"%(test_scores[1] * 100))
```

```
80/80 [==============================] - 1s 14ms/step - loss: 0.2458 - acc:
```

```
0.9488 - auc: 0.9850 - f1_score: 0.9493
Testing Accuracy: 94.88%
```

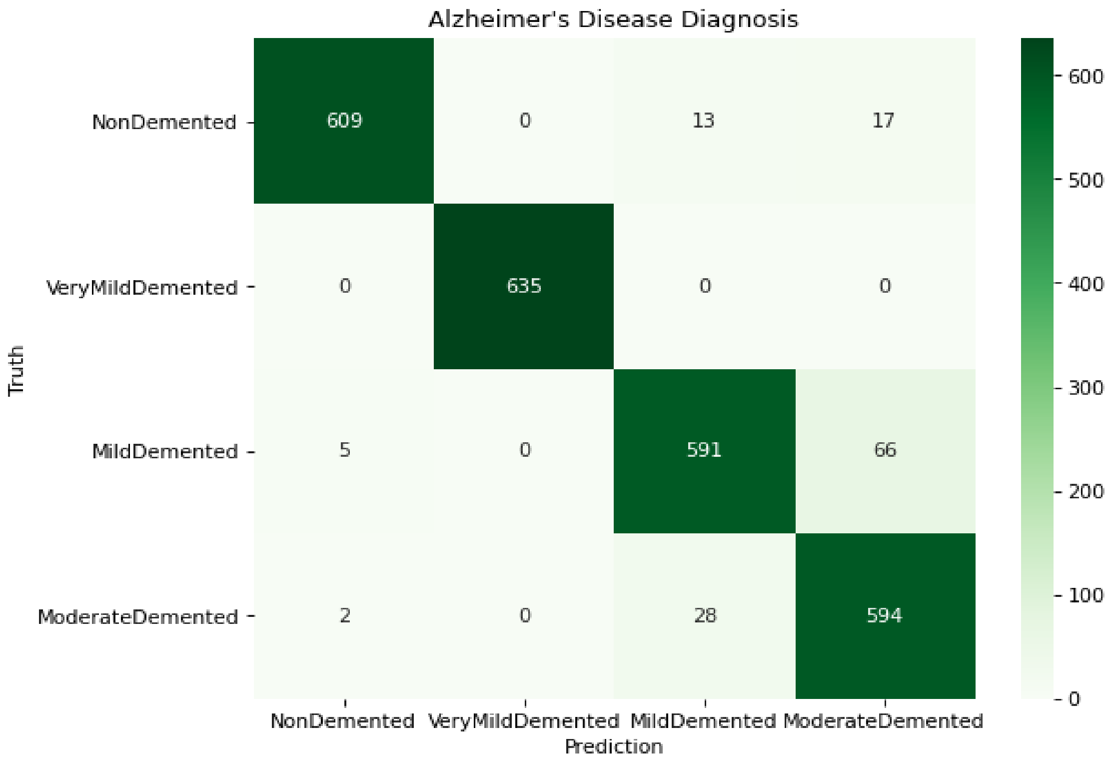
[18]:
```
#Predicting the test data

pred_labels = model.predict(test_data)
```

[19]:
```
#Print the classification report of the tested data

#Since the labels are softmax arrays, we need to roundoff to have it in the␣
↪form of 0s and 1s,
#similar to the test_labels
def roundoff(arr):
    """To round off according to the argmax of each predicted label array. """
    arr[np.argwhere(arr != arr.max())] = 0
    arr[np.argwhere(arr == arr.max())] = 1
    return arr

for labels in pred_labels:
    labels = roundoff(labels)

print(classification_report(test_labels, pred_labels, target_names=CLASSES))
```

```
                  precision    recall  f1-score   support

     NonDemented       0.99      0.95      0.97       639
VeryMildDemented       1.00      1.00      1.00       635
    MildDemented       0.94      0.89      0.91       662
ModerateDemented       0.88      0.95      0.91       624

       micro avg       0.95      0.95      0.95      2560
       macro avg       0.95      0.95      0.95      2560
    weighted avg       0.95      0.95      0.95      2560
     samples avg       0.95      0.95      0.95      2560
```

[20]:
```
#Plot the confusion matrix to understand the classification in detail

pred_ls = np.argmax(pred_labels, axis=1)
test_ls = np.argmax(test_labels, axis=1)

conf_arr = confusion_matrix(test_ls, pred_ls)

plt.figure(figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')

ax = sns.heatmap(conf_arr, cmap='Greens', annot=True, fmt='d',␣
↪xticklabels=CLASSES, yticklabels=CLASSES)
```

```
plt.title('Alzheimer\'s Disease Diagnosis')
plt.xlabel('Prediction')
plt.ylabel('Truth')
plt.show(ax)
```



[21]:

```
#Printing some other classification metrics

print("Balanced Accuracy Score: {} %".format(round(BAS(test_ls, pred_ls) * 100,␣
↪2)))
print("Matthew's Correlation Coefficient: {} %".format(round(MCC(test_ls,␣
↪pred_ls) * 100, 2)))
```

```
Balanced Accuracy Score: 94.94 %
Matthew's Correlation Coefficient: 93.22 %
```

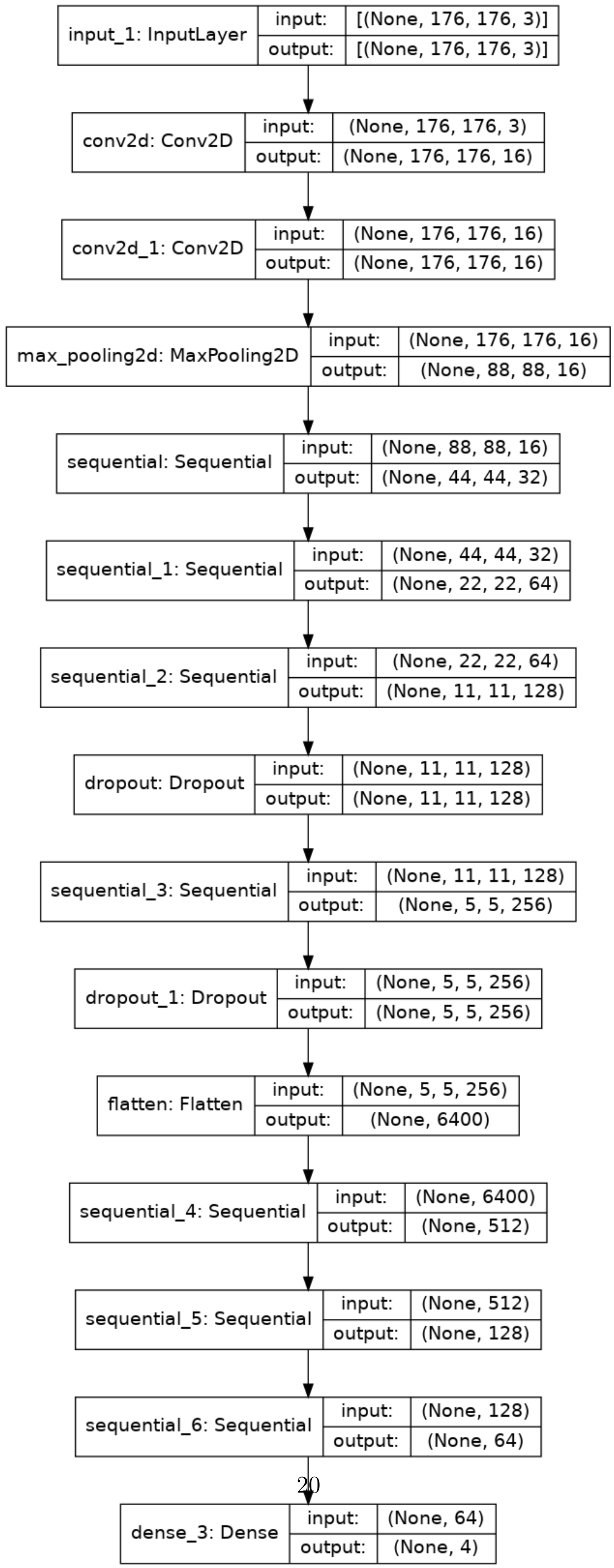
[22]:

```
#Saving the model for future use

model_dir = work_dir + "alzheimer_cnn_model"
model.save(model_dir, save_format='h5')
os.listdir(work_dir)
```

[22]:
```
['alzheimer_cnn_model',
 'MildDemented',
 'NonDemented',
 'ModerateDemented',
 'VeryMildDemented']
```

[23]:
```
pretrained_model = tf.keras.models.load_model(model_dir)

#Check its architecture
plot_model(pretrained_model, to_file=work_dir + "model_plot.png",␣
 ↪show_shapes=True, show_layer_names=True)
```

[23]:


input_1: InputLayer
input: [(None, 176, 176, 3)]
output: [(None, 176, 176, 3)]
conv2d: Conv2D
input: (None, 176, 176, 3)
output: (None, 176, 176, 16)
conv2d_1: Conv2D
input: (None, 176, 176, 16)
output: (None, 176, 176, 16)
max_pooling2d: MaxPooling2D
input: (None, 176, 176, 16)
output: (None, 88, 88, 16)
sequential: Sequential
input: (None, 88, 88, 16)
output: (None, 44, 44, 32)
sequential_1: Sequential
input: (None, 44, 44, 32)
output: (None, 22, 22, 64)
sequential_2: Sequential
input: (None, 22, 22, 64)
output: (None, 11, 11, 128)
dropout: Dropout
input: (None, 11, 11, 128)
output: (None, 11, 11, 128)
sequential_3: Sequential
input: (None, 11, 11, 128)
output: (None, 5, 5, 256)
dropout_1: Dropout
input: (None, 5, 5, 256)
output: (None, 5, 5, 256)
flatten: Flatten
input: (None, 5, 5, 256)
output: (None, 6400)
sequential_4: Sequential
input: (None, 6400)
output: (None, 512)
sequential_5: Sequential
input: (None, 512)
output: (None, 128)
sequential_6: Sequential
input: (None, 128)
output: (None, 64)
dense_3: Dense
input: (None, 64)
output: (None, 4)