disease-classification-inceptionv3

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 ReduceLROnPlateau
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
from tensorflow.keras.layers import SeparableConv2D, BatchNormalization,␣
 ↪GlobalAveragePooling2D


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', 'VeryMildDemented',
'ModerateDemented', 'NonDemented']
```

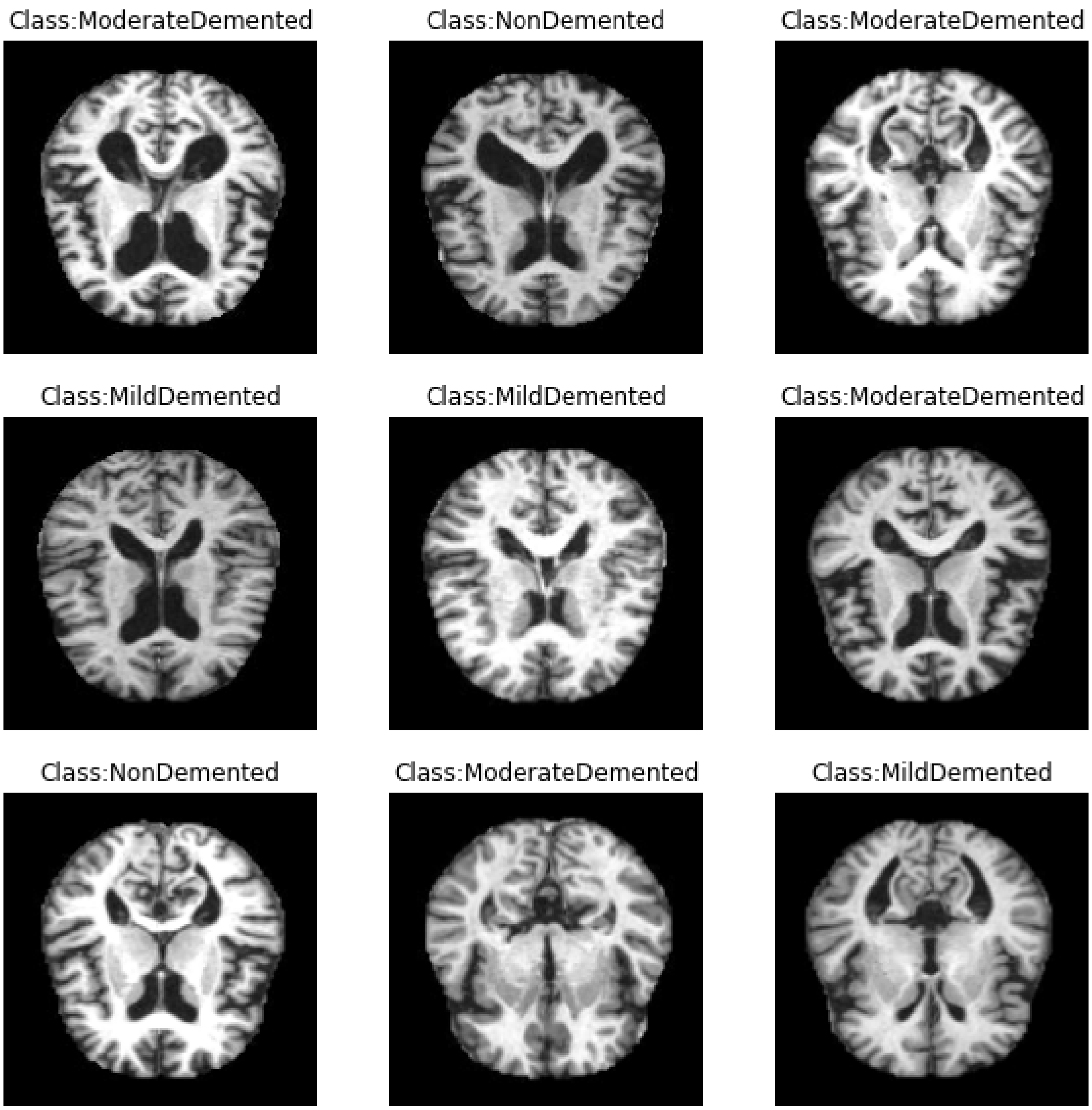
[3]:
```
WORK_DIR = './dataset/'

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
```

```python
    """

    # 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]:
```
inception_model = InceptionV3(input_shape=(176, 176, 3), include_top=False,␣
↪weights="imagenet")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 0s 0us/step
```

[11]:
```
for layer in inception_model.layers:
    layer.trainable=False
```

[12]:
```
custom_inception_model = Sequential([
        inception_model,
        Dropout(0.5),
        GlobalAveragePooling2D(),
        Flatten(),
        BatchNormalization(),
        Dense(512, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(256, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(128, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(64, activation='relu'),
```

```
        Dropout(0.5),
        BatchNormalization(),
        Dense(4, activation='softmax')
    ], name = "inception_cnn_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('acc') > 0.99:
            print("\nReached accuracy threshold! Terminating training.")
            self.model.stop_training = True

my_callback = MyCallback()

#ReduceLROnPlateau to stabilize the training process of the model
rop_callback = ReduceLROnPlateau(monitor="val_loss", patience=3)
```

[14]:
```
METRICS = [tf.keras.metrics.CategoricalAccuracy(name='acc'),
           tf.keras.metrics.AUC(name='auc'),
           tfa.metrics.F1Score(num_classes=4)]

CALLBACKS = [my_callback, rop_callback]

custom_inception_model.compile(optimizer='rmsprop',
                              loss=tf.losses.CategoricalCrossentropy(),
                              metrics=METRICS)

custom_inception_model.summary()
```

```
Model: "inception_cnn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 4, 4, 2048)        21802784
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 2048)        0
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
batch_normalization_94 (Batc (None, 2048)              8192
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
```

```
batch_normalization_95 (Batc (None, 512)              2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)              131328
_________________________________________________________________
batch_normalization_96 (Batc (None, 256)              1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)              32896
_________________________________________________________________
batch_normalization_97 (Batc (None, 128)              512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)              0
_________________________________________________________________
dense_3 (Dense)              (None, 64)               8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)               0
_________________________________________________________________
batch_normalization_98 (Batc (None, 64)               256
_________________________________________________________________
dense_4 (Dense)              (None, 4)                260
=================================================================
Total params: 23,036,644
Trainable params: 1,227,844
Non-trainable params: 21,808,800
_________________________________________________________________
```

[15]:

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

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

```
Epoch 1/100
256/256 [==============================] - 24s 58ms/step - loss: 1.5719 - acc:
0.3571 - auc: 0.6204 - f1_score: 0.3548 - val_loss: 0.7690 - val_acc: 0.6328 -
val_auc: 0.8886 - val_f1_score: 0.6269
Epoch 2/100
256/256 [==============================] - 12s 47ms/step - loss: 0.9683 - acc:
0.5569 - auc: 0.8289 - f1_score: 0.5512 - val_loss: 0.6706 - val_acc: 0.6855 -
val_auc: 0.9146 - val_f1_score: 0.6739
Epoch 3/100
256/256 [==============================] - 12s 48ms/step - loss: 0.8198 - acc:
0.6281 - auc: 0.8773 - f1_score: 0.6208 - val_loss: 0.6486 - val_acc: 0.6807 -
val_auc: 0.9185 - val_f1_score: 0.6634
Epoch 4/100
```

```
256/256 [==============================] - 12s 48ms/step - loss: 0.7384 - acc:
0.6651 - auc: 0.9002 - f1_score: 0.6647 - val_loss: 0.5888 - val_acc: 0.7236 -
val_auc: 0.9333 - val_f1_score: 0.7144
Epoch 5/100
256/256 [==============================] - 12s 48ms/step - loss: 0.6817 - acc:
0.6878 - auc: 0.9152 - f1_score: 0.6823 - val_loss: 0.5507 - val_acc: 0.7358 -
val_auc: 0.9420 - val_f1_score: 0.7282
Epoch 6/100
256/256 [==============================] - 12s 48ms/step - loss: 0.6642 - acc:
0.7008 - auc: 0.9193 - f1_score: 0.7013 - val_loss: 0.5433 - val_acc: 0.7427 -
val_auc: 0.9444 - val_f1_score: 0.7417
Epoch 7/100
256/256 [==============================] - 12s 47ms/step - loss: 0.6154 - acc:
0.7451 - auc: 0.9333 - f1_score: 0.7436 - val_loss: 0.5172 - val_acc: 0.7607 -
val_auc: 0.9498 - val_f1_score: 0.7588
Epoch 8/100
256/256 [==============================] - 12s 48ms/step - loss: 0.5910 - acc:
0.7449 - auc: 0.9378 - f1_score: 0.7446 - val_loss: 0.4915 - val_acc: 0.7866 -
val_auc: 0.9567 - val_f1_score: 0.7818
Epoch 9/100
256/256 [==============================] - 12s 47ms/step - loss: 0.5637 - acc:
0.7608 - auc: 0.9437 - f1_score: 0.7569 - val_loss: 0.4486 - val_acc: 0.7993 -
val_auc: 0.9633 - val_f1_score: 0.7964
Epoch 10/100
256/256 [==============================] - 12s 48ms/step - loss: 0.5363 - acc:
0.7695 - auc: 0.9486 - f1_score: 0.7665 - val_loss: 0.4358 - val_acc: 0.8115 -
val_auc: 0.9653 - val_f1_score: 0.8098
Epoch 11/100
256/256 [==============================] - 12s 48ms/step - loss: 0.5106 - acc:
0.7869 - auc: 0.9537 - f1_score: 0.7847 - val_loss: 0.4071 - val_acc: 0.8296 -
val_auc: 0.9699 - val_f1_score: 0.8290
Epoch 12/100
256/256 [==============================] - 12s 47ms/step - loss: 0.5009 - acc:
0.7984 - auc: 0.9552 - f1_score: 0.7975 - val_loss: 0.4066 - val_acc: 0.8228 -
val_auc: 0.9691 - val_f1_score: 0.8210
Epoch 13/100
256/256 [==============================] - 12s 48ms/step - loss: 0.4549 - acc:
0.8202 - auc: 0.9631 - f1_score: 0.8210 - val_loss: 0.3969 - val_acc: 0.8306 -
val_auc: 0.9708 - val_f1_score: 0.8293
Epoch 14/100
256/256 [==============================] - 12s 47ms/step - loss: 0.4585 - acc:
0.8194 - auc: 0.9627 - f1_score: 0.8184 - val_loss: 0.3753 - val_acc: 0.8389 -
val_auc: 0.9741 - val_f1_score: 0.8374
Epoch 15/100
256/256 [==============================] - 12s 47ms/step - loss: 0.4537 - acc:
0.8189 - auc: 0.9632 - f1_score: 0.8198 - val_loss: 0.3720 - val_acc: 0.8457 -
val_auc: 0.9741 - val_f1_score: 0.8447
Epoch 16/100
```

```
256/256 [==============================] - 12s 48ms/step - loss: 0.4307 - acc:
0.8317 - auc: 0.9670 - f1_score: 0.8299 - val_loss: 0.3618 - val_acc: 0.8506 -
val_auc: 0.9758 - val_f1_score: 0.8486
Epoch 17/100
256/256 [==============================] - 12s 47ms/step - loss: 0.4140 - acc:
0.8459 - auc: 0.9693 - f1_score: 0.8469 - val_loss: 0.3750 - val_acc: 0.8423 -
val_auc: 0.9743 - val_f1_score: 0.8391
Epoch 18/100
256/256 [==============================] - 12s 48ms/step - loss: 0.4016 - acc:
0.8519 - auc: 0.9712 - f1_score: 0.8503 - val_loss: 0.3475 - val_acc: 0.8608 -
val_auc: 0.9777 - val_f1_score: 0.8590
Epoch 19/100
256/256 [==============================] - 12s 48ms/step - loss: 0.3702 - acc:
0.8567 - auc: 0.9748 - f1_score: 0.8554 - val_loss: 0.3757 - val_acc: 0.8359 -
val_auc: 0.9739 - val_f1_score: 0.8371
Epoch 20/100
256/256 [==============================] - 12s 47ms/step - loss: 0.3788 - acc:
0.8611 - auc: 0.9741 - f1_score: 0.8612 - val_loss: 0.3454 - val_acc: 0.8599 -
val_auc: 0.9778 - val_f1_score: 0.8579
Epoch 21/100
256/256 [==============================] - 12s 47ms/step - loss: 0.3781 - acc:
0.8633 - auc: 0.9738 - f1_score: 0.8635 - val_loss: 0.3740 - val_acc: 0.8574 -
val_auc: 0.9752 - val_f1_score: 0.8549
Epoch 22/100
256/256 [==============================] - 12s 47ms/step - loss: 0.3551 - acc:
0.8694 - auc: 0.9771 - f1_score: 0.8677 - val_loss: 0.3317 - val_acc: 0.8638 -
val_auc: 0.9802 - val_f1_score: 0.8628
Epoch 23/100
256/256 [==============================] - 12s 48ms/step - loss: 0.3352 - acc:
0.8815 - auc: 0.9793 - f1_score: 0.8827 - val_loss: 0.3263 - val_acc: 0.8696 -
val_auc: 0.9804 - val_f1_score: 0.8676
Epoch 24/100
256/256 [==============================] - 12s 47ms/step - loss: 0.3131 - acc:
0.8861 - auc: 0.9818 - f1_score: 0.8851 - val_loss: 0.3254 - val_acc: 0.8682 -
val_auc: 0.9809 - val_f1_score: 0.8663
Epoch 25/100
256/256 [==============================] - 12s 47ms/step - loss: 0.3025 - acc:
0.8929 - auc: 0.9828 - f1_score: 0.8924 - val_loss: 0.3461 - val_acc: 0.8589 -
val_auc: 0.9785 - val_f1_score: 0.8577
Epoch 26/100
256/256 [==============================] - 12s 48ms/step - loss: 0.3247 - acc:
0.8799 - auc: 0.9801 - f1_score: 0.8794 - val_loss: 0.3268 - val_acc: 0.8672 -
val_auc: 0.9814 - val_f1_score: 0.8657
Epoch 27/100
256/256 [==============================] - 12s 47ms/step - loss: 0.3072 - acc:
0.8899 - auc: 0.9821 - f1_score: 0.8896 - val_loss: 0.3142 - val_acc: 0.8784 -
val_auc: 0.9825 - val_f1_score: 0.8771
Epoch 28/100
```

```
256/256 [==============================] - 12s 47ms/step - loss: 0.2817 - acc:
0.8988 - auc: 0.9851 - f1_score: 0.8980 - val_loss: 0.3369 - val_acc: 0.8633 -
val_auc: 0.9811 - val_f1_score: 0.8629
Epoch 29/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2859 - acc:
0.9001 - auc: 0.9839 - f1_score: 0.8999 - val_loss: 0.3036 - val_acc: 0.8838 -
val_auc: 0.9834 - val_f1_score: 0.8825
Epoch 30/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2897 - acc:
0.8927 - auc: 0.9842 - f1_score: 0.8915 - val_loss: 0.3020 - val_acc: 0.8789 -
val_auc: 0.9836 - val_f1_score: 0.8784
Epoch 31/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2617 - acc:
0.9078 - auc: 0.9863 - f1_score: 0.9085 - val_loss: 0.3057 - val_acc: 0.8828 -
val_auc: 0.9838 - val_f1_score: 0.8824
Epoch 32/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2816 - acc:
0.9011 - auc: 0.9848 - f1_score: 0.9007 - val_loss: 0.3001 - val_acc: 0.8848 -
val_auc: 0.9835 - val_f1_score: 0.8840
Epoch 33/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2673 - acc:
0.9061 - auc: 0.9859 - f1_score: 0.9054 - val_loss: 0.3083 - val_acc: 0.8857 -
val_auc: 0.9838 - val_f1_score: 0.8853
Epoch 34/100
256/256 [==============================] - 12s 48ms/step - loss: 0.2628 - acc:
0.9125 - auc: 0.9856 - f1_score: 0.9121 - val_loss: 0.3197 - val_acc: 0.8809 -
val_auc: 0.9818 - val_f1_score: 0.8800
Epoch 35/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2688 - acc:
0.9126 - auc: 0.9852 - f1_score: 0.9129 - val_loss: 0.3060 - val_acc: 0.8857 -
val_auc: 0.9839 - val_f1_score: 0.8853
Epoch 36/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2416 - acc:
0.9149 - auc: 0.9887 - f1_score: 0.9137 - val_loss: 0.2967 - val_acc: 0.8901 -
val_auc: 0.9846 - val_f1_score: 0.8896
Epoch 37/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2298 - acc:
0.9230 - auc: 0.9892 - f1_score: 0.9227 - val_loss: 0.2950 - val_acc: 0.8892 -
val_auc: 0.9851 - val_f1_score: 0.8886
Epoch 38/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2039 - acc:
0.9325 - auc: 0.9913 - f1_score: 0.9319 - val_loss: 0.2930 - val_acc: 0.8945 -
val_auc: 0.9854 - val_f1_score: 0.8939
Epoch 39/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2304 - acc:
0.9179 - auc: 0.9897 - f1_score: 0.9180 - val_loss: 0.2920 - val_acc: 0.8916 -
val_auc: 0.9855 - val_f1_score: 0.8907
Epoch 40/100
```

```
256/256 [==============================] - 12s 47ms/step - loss: 0.2080 - acc:
0.9280 - auc: 0.9909 - f1_score: 0.9278 - val_loss: 0.2888 - val_acc: 0.8940 -
val_auc: 0.9859 - val_f1_score: 0.8932
Epoch 41/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2090 - acc:
0.9317 - auc: 0.9905 - f1_score: 0.9313 - val_loss: 0.2913 - val_acc: 0.8931 -
val_auc: 0.9856 - val_f1_score: 0.8922
Epoch 42/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2060 - acc:
0.9382 - auc: 0.9904 - f1_score: 0.9380 - val_loss: 0.2885 - val_acc: 0.8931 -
val_auc: 0.9859 - val_f1_score: 0.8922
Epoch 43/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2181 - acc:
0.9309 - auc: 0.9896 - f1_score: 0.9310 - val_loss: 0.2873 - val_acc: 0.8984 -
val_auc: 0.9859 - val_f1_score: 0.8976
Epoch 44/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1872 - acc:
0.9371 - auc: 0.9924 - f1_score: 0.9373 - val_loss: 0.2855 - val_acc: 0.8994 -
val_auc: 0.9863 - val_f1_score: 0.8988
Epoch 45/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1993 - acc:
0.9352 - auc: 0.9912 - f1_score: 0.9350 - val_loss: 0.2885 - val_acc: 0.9004 -
val_auc: 0.9862 - val_f1_score: 0.8996
Epoch 46/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2115 - acc:
0.9267 - auc: 0.9906 - f1_score: 0.9272 - val_loss: 0.2845 - val_acc: 0.8989 -
val_auc: 0.9865 - val_f1_score: 0.8982
Epoch 47/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1781 - acc:
0.9369 - auc: 0.9932 - f1_score: 0.9372 - val_loss: 0.2870 - val_acc: 0.9004 -
val_auc: 0.9864 - val_f1_score: 0.8996
Epoch 48/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1713 - acc:
0.9461 - auc: 0.9935 - f1_score: 0.9464 - val_loss: 0.2870 - val_acc: 0.9004 -
val_auc: 0.9867 - val_f1_score: 0.8997
Epoch 49/100
256/256 [==============================] - 12s 48ms/step - loss: 0.1934 - acc:
0.9365 - auc: 0.9916 - f1_score: 0.9363 - val_loss: 0.2841 - val_acc: 0.9019 -
val_auc: 0.9870 - val_f1_score: 0.9012
Epoch 50/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1638 - acc:
0.9462 - auc: 0.9943 - f1_score: 0.9459 - val_loss: 0.2870 - val_acc: 0.9004 -
val_auc: 0.9868 - val_f1_score: 0.8998
Epoch 51/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1811 - acc:
0.9422 - auc: 0.9925 - f1_score: 0.9418 - val_loss: 0.2872 - val_acc: 0.8999 -
val_auc: 0.9865 - val_f1_score: 0.8994
Epoch 52/100
```

```
256/256 [==============================] - 12s 47ms/step - loss: 0.1694 - acc:
0.9386 - auc: 0.9946 - f1_score: 0.9387 - val_loss: 0.2885 - val_acc: 0.9004 -
val_auc: 0.9867 - val_f1_score: 0.8997
Epoch 53/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1510 - acc:
0.9471 - auc: 0.9952 - f1_score: 0.9472 - val_loss: 0.2866 - val_acc: 0.9019 -
val_auc: 0.9867 - val_f1_score: 0.9011
Epoch 54/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1744 - acc:
0.9443 - auc: 0.9932 - f1_score: 0.9443 - val_loss: 0.2866 - val_acc: 0.9009 -
val_auc: 0.9867 - val_f1_score: 0.9003
Epoch 55/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1616 - acc:
0.9477 - auc: 0.9934 - f1_score: 0.9474 - val_loss: 0.2861 - val_acc: 0.9043 -
val_auc: 0.9870 - val_f1_score: 0.9037
Epoch 56/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1784 - acc:
0.9384 - auc: 0.9932 - f1_score: 0.9384 - val_loss: 0.2853 - val_acc: 0.9028 -
val_auc: 0.9869 - val_f1_score: 0.9022
Epoch 57/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1686 - acc:
0.9443 - auc: 0.9934 - f1_score: 0.9450 - val_loss: 0.2868 - val_acc: 0.9019 -
val_auc: 0.9869 - val_f1_score: 0.9012
Epoch 58/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2032 - acc:
0.9313 - auc: 0.9907 - f1_score: 0.9315 - val_loss: 0.2864 - val_acc: 0.9009 -
val_auc: 0.9868 - val_f1_score: 0.9003
Epoch 59/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1660 - acc:
0.9431 - auc: 0.9937 - f1_score: 0.9431 - val_loss: 0.2875 - val_acc: 0.9023 -
val_auc: 0.9870 - val_f1_score: 0.9017
Epoch 60/100
256/256 [==============================] - 12s 47ms/step - loss: 0.2014 - acc:
0.9362 - auc: 0.9906 - f1_score: 0.9355 - val_loss: 0.2869 - val_acc: 0.9014 -
val_auc: 0.9869 - val_f1_score: 0.9007
Epoch 61/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1668 - acc:
0.9402 - auc: 0.9940 - f1_score: 0.9401 - val_loss: 0.2863 - val_acc: 0.9023 -
val_auc: 0.9867 - val_f1_score: 0.9015
Epoch 62/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1655 - acc:
0.9462 - auc: 0.9942 - f1_score: 0.9462 - val_loss: 0.2845 - val_acc: 0.9019 -
val_auc: 0.9871 - val_f1_score: 0.9011
Epoch 63/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1865 - acc:
0.9377 - auc: 0.9919 - f1_score: 0.9381 - val_loss: 0.2871 - val_acc: 0.9023 -
val_auc: 0.9869 - val_f1_score: 0.9018
Epoch 64/100
```

```
256/256 [==============================] - 12s 47ms/step - loss: 0.1573 - acc:
0.9453 - auc: 0.9949 - f1_score: 0.9451 - val_loss: 0.2874 - val_acc: 0.9014 -
val_auc: 0.9867 - val_f1_score: 0.9008
Epoch 65/100
256/256 [==============================] - 12s 48ms/step - loss: 0.1820 - acc:
0.9430 - auc: 0.9919 - f1_score: 0.9431 - val_loss: 0.2871 - val_acc: 0.9028 -
val_auc: 0.9867 - val_f1_score: 0.9022
Epoch 66/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1592 - acc:
0.9513 - auc: 0.9941 - f1_score: 0.9511 - val_loss: 0.2886 - val_acc: 0.9009 -
val_auc: 0.9865 - val_f1_score: 0.9001
Epoch 67/100
256/256 [==============================] - 12s 48ms/step - loss: 0.1779 - acc:
0.9367 - auc: 0.9930 - f1_score: 0.9359 - val_loss: 0.2878 - val_acc: 0.9004 -
val_auc: 0.9865 - val_f1_score: 0.8996
Epoch 68/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1730 - acc:
0.9425 - auc: 0.9930 - f1_score: 0.9421 - val_loss: 0.2852 - val_acc: 0.9019 -
val_auc: 0.9870 - val_f1_score: 0.9013
Epoch 69/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1824 - acc:
0.9405 - auc: 0.9922 - f1_score: 0.9398 - val_loss: 0.2875 - val_acc: 0.9023 -
val_auc: 0.9867 - val_f1_score: 0.9016
Epoch 70/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1545 - acc:
0.9508 - auc: 0.9947 - f1_score: 0.9509 - val_loss: 0.2848 - val_acc: 0.9009 -
val_auc: 0.9870 - val_f1_score: 0.9002
Epoch 71/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1932 - acc:
0.9351 - auc: 0.9917 - f1_score: 0.9350 - val_loss: 0.2871 - val_acc: 0.9023 -
val_auc: 0.9867 - val_f1_score: 0.9019
Epoch 72/100
256/256 [==============================] - 12s 48ms/step - loss: 0.1658 - acc:
0.9413 - auc: 0.9942 - f1_score: 0.9408 - val_loss: 0.2875 - val_acc: 0.9019 -
val_auc: 0.9867 - val_f1_score: 0.9014
Epoch 73/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1828 - acc:
0.9388 - auc: 0.9924 - f1_score: 0.9389 - val_loss: 0.2853 - val_acc: 0.9019 -
val_auc: 0.9869 - val_f1_score: 0.9012
Epoch 74/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1579 - acc:
0.9478 - auc: 0.9944 - f1_score: 0.9477 - val_loss: 0.2856 - val_acc: 0.9023 -
val_auc: 0.9869 - val_f1_score: 0.9015
Epoch 75/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1729 - acc:
0.9397 - auc: 0.9931 - f1_score: 0.9391 - val_loss: 0.2849 - val_acc: 0.9014 -
val_auc: 0.9869 - val_f1_score: 0.9007
Epoch 76/100
```

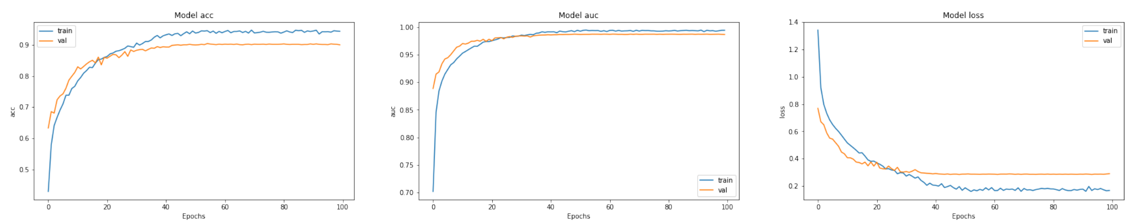
```
256/256 [==============================] - 12s 47ms/step - loss: 0.1845 - acc:
0.9394 - auc: 0.9927 - f1_score: 0.9390 - val_loss: 0.2863 - val_acc: 0.9028 -
val_auc: 0.9870 - val_f1_score: 0.9022
Epoch 77/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1800 - acc:
0.9443 - auc: 0.9922 - f1_score: 0.9442 - val_loss: 0.2866 - val_acc: 0.9019 -
val_auc: 0.9867 - val_f1_score: 0.9012
Epoch 78/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1773 - acc:
0.9432 - auc: 0.9924 - f1_score: 0.9428 - val_loss: 0.2855 - val_acc: 0.9019 -
val_auc: 0.9869 - val_f1_score: 0.9012
Epoch 79/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1857 - acc:
0.9429 - auc: 0.9921 - f1_score: 0.9428 - val_loss: 0.2874 - val_acc: 0.9019 -
val_auc: 0.9867 - val_f1_score: 0.9011
Epoch 80/100
256/256 [==============================] - 12s 48ms/step - loss: 0.1856 - acc:
0.9388 - auc: 0.9924 - f1_score: 0.9384 - val_loss: 0.2854 - val_acc: 0.9023 -
val_auc: 0.9869 - val_f1_score: 0.9016
Epoch 81/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1930 - acc:
0.9366 - auc: 0.9917 - f1_score: 0.9364 - val_loss: 0.2866 - val_acc: 0.9033 -
val_auc: 0.9869 - val_f1_score: 0.9027
Epoch 82/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1670 - acc:
0.9477 - auc: 0.9936 - f1_score: 0.9476 - val_loss: 0.2853 - val_acc: 0.9019 -
val_auc: 0.9870 - val_f1_score: 0.9011
Epoch 83/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1645 - acc:
0.9430 - auc: 0.9939 - f1_score: 0.9425 - val_loss: 0.2865 - val_acc: 0.9014 -
val_auc: 0.9869 - val_f1_score: 0.9005
Epoch 84/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1772 - acc:
0.9413 - auc: 0.9933 - f1_score: 0.9412 - val_loss: 0.2853 - val_acc: 0.9019 -
val_auc: 0.9868 - val_f1_score: 0.9012
Epoch 85/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1720 - acc:
0.9447 - auc: 0.9932 - f1_score: 0.9452 - val_loss: 0.2864 - val_acc: 0.9019 -
val_auc: 0.9867 - val_f1_score: 0.9012
Epoch 86/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1679 - acc:
0.9468 - auc: 0.9934 - f1_score: 0.9469 - val_loss: 0.2855 - val_acc: 0.9019 -
val_auc: 0.9869 - val_f1_score: 0.9012
Epoch 87/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1735 - acc:
0.9462 - auc: 0.9934 - f1_score: 0.9460 - val_loss: 0.2865 - val_acc: 0.9009 -
val_auc: 0.9868 - val_f1_score: 0.9003
Epoch 88/100
```

```
256/256 [==============================] - 12s 47ms/step - loss: 0.1705 - acc:
0.9410 - auc: 0.9937 - f1_score: 0.9409 - val_loss: 0.2856 - val_acc: 0.9014 -
val_auc: 0.9869 - val_f1_score: 0.9008
Epoch 89/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1653 - acc:
0.9457 - auc: 0.9938 - f1_score: 0.9455 - val_loss: 0.2851 - val_acc: 0.9014 -
val_auc: 0.9870 - val_f1_score: 0.9008
Epoch 90/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1720 - acc:
0.9431 - auc: 0.9936 - f1_score: 0.9433 - val_loss: 0.2866 - val_acc: 0.9033 -
val_auc: 0.9867 - val_f1_score: 0.9027
Epoch 91/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1818 - acc:
0.9427 - auc: 0.9925 - f1_score: 0.9427 - val_loss: 0.2859 - val_acc: 0.9019 -
val_auc: 0.9869 - val_f1_score: 0.9012
Epoch 92/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1556 - acc:
0.9478 - auc: 0.9950 - f1_score: 0.9470 - val_loss: 0.2878 - val_acc: 0.9033 -
val_auc: 0.9868 - val_f1_score: 0.9026
Epoch 93/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1821 - acc:
0.9408 - auc: 0.9923 - f1_score: 0.9402 - val_loss: 0.2865 - val_acc: 0.9019 -
val_auc: 0.9869 - val_f1_score: 0.9012
Epoch 94/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1747 - acc:
0.9406 - auc: 0.9935 - f1_score: 0.9409 - val_loss: 0.2842 - val_acc: 0.9014 -
val_auc: 0.9870 - val_f1_score: 0.9008
Epoch 95/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1801 - acc:
0.9446 - auc: 0.9929 - f1_score: 0.9443 - val_loss: 0.2869 - val_acc: 0.9014 -
val_auc: 0.9867 - val_f1_score: 0.9009
Epoch 96/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1780 - acc:
0.9415 - auc: 0.9933 - f1_score: 0.9416 - val_loss: 0.2863 - val_acc: 0.9009 -
val_auc: 0.9868 - val_f1_score: 0.9002
Epoch 97/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1861 - acc:
0.9420 - auc: 0.9913 - f1_score: 0.9420 - val_loss: 0.2869 - val_acc: 0.9033 -
val_auc: 0.9869 - val_f1_score: 0.9027
Epoch 98/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1704 - acc:
0.9468 - auc: 0.9932 - f1_score: 0.9468 - val_loss: 0.2858 - val_acc: 0.9023 -
val_auc: 0.9870 - val_f1_score: 0.9018
Epoch 99/100
256/256 [==============================] - 12s 47ms/step - loss: 0.1705 - acc:
0.9412 - auc: 0.9936 - f1_score: 0.9401 - val_loss: 0.2882 - val_acc: 0.9019 -
val_auc: 0.9866 - val_f1_score: 0.9010
Epoch 100/100
```

```
256/256 [==============================] - 12s 47ms/step - loss: 0.1721 - acc:
0.9432 - auc: 0.9937 - f1_score: 0.9428 - val_loss: 0.2899 - val_acc: 0.9004 -
val_auc: 0.9864 - val_f1_score: 0.8996
```

[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 = custom_inception_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 [==============================] - 3s 37ms/step - loss: 0.2765 - acc:
0.9055 - auc: 0.9875 - f1_score: 0.9055
Testing Accuracy: 90.55%
```

[18]:

```
#Predicting the test data

pred_labels = custom_inception_model.predict(test_data)
```

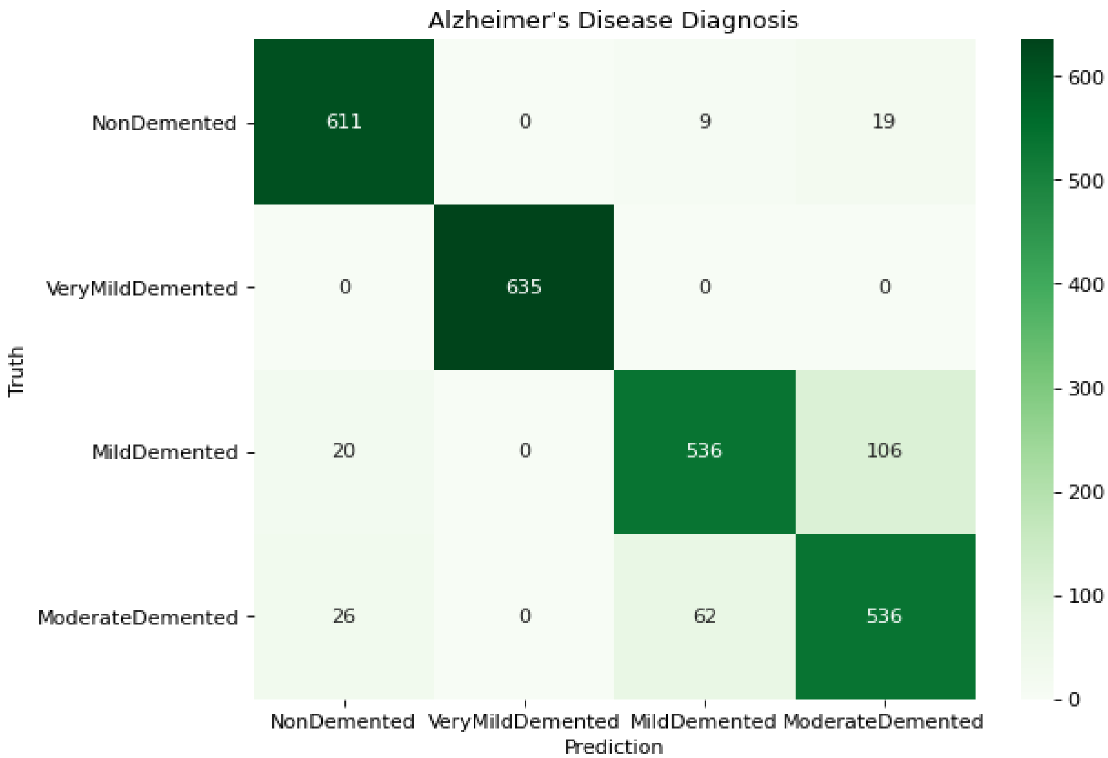
[19]:

```
#Print the classification report of the tested data
```

```python
#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       0.93      0.96      0.94       639
VeryMildDemented       1.00      1.00      1.00       635
    MildDemented       0.88      0.81      0.84       662
ModerateDemented       0.81      0.86      0.83       624

       micro avg       0.91      0.91      0.91      2560
       macro avg       0.91      0.91      0.91      2560
    weighted avg       0.91      0.91      0.91      2560
     samples avg       0.91      0.91      0.91      2560
```

[20]:
```python
#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: 90.62 %
Matthew's Correlation Coefficient: 87.44 %
```

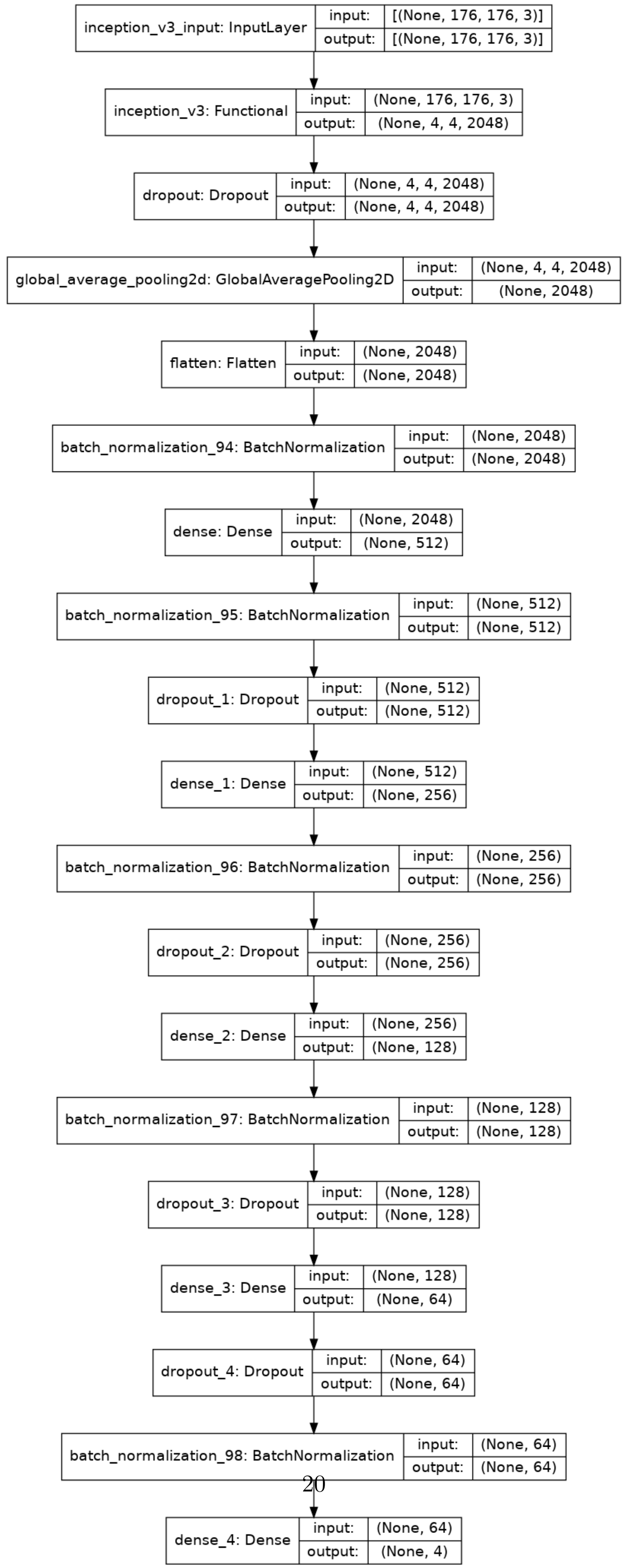
[22]:
```
#Saving the model for future use

custom_inception_model_dir = work_dir + "alzheimer_inception_cnn_model"
custom_inception_model.save(custom_inception_model_dir, save_format='h5')
os.listdir(work_dir)
```

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

[23]:

```
pretrained_model = tf.keras.models.load_model(custom_inception_model_dir)

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

[23]:


inception_v3_input: InputLayer
input: [(None, 176, 176, 3)]
output: [(None, 176, 176, 3)]
inception_v3: Functional
input: (None, 176, 176, 3)
output: (None, 4, 4, 2048)
dropout: Dropout
input: (None, 4, 4, 2048)
output: (None, 4, 4, 2048)
global_average_pooling2d: GlobalAveragePooling2D
input: (None, 4, 4, 2048)
output: (None, 2048)
flatten: Flatten
input: (None, 2048)
output: (None, 2048)
batch_normalization_94: BatchNormalization
input: (None, 2048)
output: (None, 2048)
dense: Dense
input: (None, 2048)
output: (None, 512)
batch_normalization_95: BatchNormalization
input: (None, 512)
output: (None, 512)
dropout_1: Dropout
input: (None, 512)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 256)
batch_normalization_96: BatchNormalization
input: (None, 256)
output: (None, 256)
dropout_2: Dropout
input: (None, 256)
output: (None, 256)
dense_2: Dense
input: (None, 256)
output: (None, 128)
batch_normalization_97: BatchNormalization
input: (None, 128)
output: (None, 128)
dropout_3: Dropout
input: (None, 128)
output: (None, 128)
dense_3: Dense
input: (None, 128)
output: (None, 64)
dropout_4: Dropout
input: (None, 64)
output: (None, 64)
batch_normalization_98: BatchNormalization
input: (None, 64)
output: (None, 64)
dense_4: Dense
input: (None, 64)
output: (None, 4)