National College of Ireland

Configuration Manual: An Approach to Classify Alzheimer's Disease using Vision Transformers

MSc Research Project
Data Analytics

Anitha Drewitt
Student ID: X21122954

School of Computing
National College of Ireland

Supervisor: Qurrat Ul Ain

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Anitha Drewitt
Student ID:	X21122954
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Qurrat Ul Ain
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual: An Approach to Classify Alzheimer's Disease using Vision Transformers
Word Count:	518
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	1st February 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual: An Approach to Classify Alzheimer's Disease using Vision Transformers

Anitha Drewitt
X21122954

1 Introduction

"An Approach to Classify Alzheimer's Disease using Vision Transformers" research can be recreated following this configuration guide. The prerequisites that must be met in order to successfully set up, create, run, and test this research using the suggested framework are covered in depth in this setup manual. Details on the environment setup and the libraries required to implement this project are provided. All the information about the dataset, implementation and evaluation has been provided in this configuration manual.

2 Hardware Specification

- Operating System: Windows 11 Home Single Language (11.0, Build 22000.1219)
- Processor: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
- Installed RAM: 16.0 GB
- System Type: 64-bit operating system, x64-based processor

2.1 Software Specification

- Python
- Windows 11

2.2 Cloud Storage

- Google Drive
- Google Colab

3 Libraries required

The commands that can be used to import each library that was needed for this research project are listed below.

Library	Command
numpy	import numpy as np
pandas	import pandas as pd
os	import os
tensorflow	import tensorflow as tf
	import keras
	from tensorflow.keras.preprocessing.image import ImageDataGenerator
	from tensorflow.keras.preprocessing.image import load_img, img_to_array
	from keras.applications.vgg16 import VGG16,preprocess_input
	from keras.utils import np_utils
matplot	import matplotlib.pylab as plt
seaborn	import seaborn as sns
shap	import shap
sklearn	from sklearn.utils import shuffle
	from sklearn.metrics import confusion_matrix
	from sklearn.model_selection import train_test_split
	from sklearn.metrics import classification_report

Figure 1: Library and Command

4 Dataset

The dataset for this research is taken from Kaggle which is completely available on the public domain named Kaggle. This dataset has a total of 6400 images of four stages namely Mild Demented (896 images), Moderate Demented (64 images), Non Demented (3200 images) and Very Mild Demented (2240 images).

4.1 Data Preparation

The dataset is split for training, validating, and testing the model and divided into 80% training and 20% testing. The training data is further divided into 20% validation and 80% training.

4.2 Data Preprocessing

Data Augmentation is used in preprocessing the data. It is a technique which is used to increase the amount of data and it helps in avoiding the issue of overfitting.

4.3 Loading the dataset

The augmented data is then passed through the pre-trained ViT architecture.

5 Classification Model Implementation and Evaluation

For ViT, the implementation process is started from scratch. Below code demonstrates how the libraries are exported and the dataset has been prepared, preprocessed and used vision transformer to classify the four stages and how the model has been evaluated.

```
#importing the libraries
import numpy as np
import pandas as pd

import os

import copy
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
import cv2
import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns
!pip install --quiet shap
import shap
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from keras.applications.vgg16 import VGG16,preprocess_input
```

```
     |████████████████████████████████| 575 kB 7.3 MB/s
```

Figure 2: Importing the libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] import os
    os.chdir('/content/drive/MyDrive/Alzheimer disease Classification')
```

Figure 3: Mounting the drive with google drive

```
W = 128 # The default size is 224 but resize to .5 to save memory size
H = 128 # The default size is 224 but resize to .5 to save memory size

label_to_class = {
    'Mild_Demented': 0,
    'Very_Mild_Demented': 1,
    'Non_Demented': 2,
    'Moderate_Demented':3,
}
class_to_label = {v: k for k, v in label_to_class.items()}
n_classes = len(label_to_class)

def get_images(dir_name='dataset', label_to_class=label_to_class):
    """read images / labels from directory"""

    Images = []
    Classes = []

    for j in ['/Dataset']:
        for label_name in os.listdir(dir_name+str(j)):
            cls = label_to_class[label_name]

            for img_name in os.listdir('/'.join([dir_name+str(j), label_name])):
                img = load_img('/'.join([dir_name+str(j), label_name, img_name]), target_size=(W, H))
                img = img_to_array(img)

                Images.append(img)
                Classes.append(cls)

    Images = np.array(Images, dtype=np.float32)
    Classes = np.array(Classes, dtype=np.float32)
    Images, Classes = shuffle(Images, Classes, random_state=0)

    return Images, Classes
```

Figure 4: Setting the width and height of the image, labeling the classes and setting the image in image array and class in class array

```
[ ] ## split train / test

    indices_train, indices_test = train_test_split(list(range(Images.shape[0])), train_size=0.8, test_size=0.2, shuffle=True)

    x_train = Images[indices_train]
    y_train = Classes[indices_train]
    x_test = Images[indices_test]
    y_test = Classes[indices_test]

    x_train.shape, y_train.shape, x_test.shape, y_test.shape

    ((5120, 128, 128, 3), (5120,), (1280, 128, 128, 3), (1280,))
```

Figure 5: Splittig the data to train and test set. 80% training set and 20% test set

```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 16
num_epochs = 200
image_size = 128  # We'll resize input images to this size
patch_size = 6  # Size of the patches to be extract from the input images
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim * 2,
    projection_dim,
]  # Size of the transformer layers
transformer_layers = 8
mlp_head_units = [2048, 1024]
```

Figure 6: Defining the parameters

```
[ ]  data_augmentation = keras.Sequential(
        [
            layers.Normalization(),
            layers.Resizing(image_size, image_size),
            layers.RandomFlip("horizontal"),
            layers.RandomRotation(factor=0.02),
            layers.RandomZoom(
                height_factor=0.2, width_factor=0.2
            ),
        ],
        name="data_augmentation",
    )
    # Compute the mean and the variance of the training data for normalization.
    data_augmentation.layers[0].adapt(x_train)
```

Figure 7: Data augmentation

Using the GELU activation function and Dropout layer which are used in Vit

```
[ ]  def mlp(x, hidden_units, dropout_rate):
         for units in hidden_units:
             x = layers.Dense(units, activation=tf.nn.gelu)(x) # GELU activation function
             x = layers.Dropout(dropout_rate)(x) # Dropout layer
         return x
```

Figure 8: Using the GELU activation function and Dropout layer which are used in ViT

```
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches
```

Figure 9: Converting the images into fixed size patches

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(4, 4))
image = x_train[np.random.choice(range(x_train.shape[0]))]
plt.imshow(image.astype("uint8"))
plt.axis("off")

resized_image = tf.image.resize(
    tf.convert_to_tensor([image]), size=(image_size, image_size)
)
patches = Patches(patch_size)(resized_image)
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}")

n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4, 4))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(patch_img.numpy().astype("uint8"))
    plt.axis("off")
```

Figure 10: To print the size and image of the patches

```python
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded
```

Figure 11: A position embedding is added to embed the patches

```python
def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    # Augment data.
    augmented = data_augmentation(inputs)
    # Create patches.
    patches = Patches(patch_size)(augmented)
    # Encode patches.
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    # Classify outputs.
    logits = layers.Dense(num_classes, activation="sigmoid")(features)
    # Create the Keras model.
    model = keras.Model(inputs=inputs, outputs=logits)
    return model
```

Figure 12: ViT classifier

```python
# compile the model
model = create_vit_classifier()
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss=keras.losses.CategoricalCrossentropy(),
    metrics=[keras.metrics.CategoricalAccuracy(name="accuracy")],
)
model.summary()
# train the model
history = model.fit(
    x_train, y_train, batch_size=16, epochs=200, validation_split=0.25
)

# evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test loss: {round(loss, 2)}")
print(f"Test accuracy: {round(accuracy * 100, 2)} %")
```

```
Epoch 142/200
240/240 [==============================] - 37s 154ms/step - loss: 0.5913 - accuracy: 0.7372 - val_loss: 0.5408 - val_accuracy: 0.7906
Epoch 143/200
240/240 [==============================] - 37s 154ms/step - loss: 0.5808 - accuracy: 0.7359 - val_loss: 0.5245 - val_accuracy: 0.7820
Epoch 144/200
240/240 [==============================] - 37s 154ms/step - loss: 0.5876 - accuracy: 0.7344 - val_loss: 0.5930 - val_accuracy: 0.7352
Epoch 145/200
240/240 [==============================] - 37s 155ms/step - loss: 0.5914 - accuracy: 0.7318 - val_loss: 0.5296 - val_accuracy: 0.7820
Epoch 146/200
240/240 [==============================] - 39s 162ms/step - loss: 0.5740 - accuracy: 0.7388 - val_loss: 0.5274 - val_accuracy: 0.7797
Epoch 147/200
240/240 [==============================] - 37s 154ms/step - loss: 0.5729 - accuracy: 0.7464 - val_loss: 0.5307 - val_accuracy: 0.8031
Epoch 148/200
240/240 [==============================] - 39s 163ms/step - loss: 0.5637 - accuracy: 0.7482 - val_loss: 0.5024 - val_accuracy: 0.8180
Epoch 149/200
240/240 [==============================] - 37s 154ms/step - loss: 0.5628 - accuracy: 0.7576 - val_loss: 0.5100 - val_accuracy: 0.8094
Epoch 150/200
```

Figure 13: Compile the model

```
Epoch 192/200
240/240 [==============================] - 39s 162ms/step - loss: 0.4476 - accuracy: 0.8094 - val_loss: 0.3812 - val_accuracy: 0.8617
Epoch 193/200
240/240 [==============================] - 39s 163ms/step - loss: 0.4353 - accuracy: 0.8141 - val_loss: 0.4583 - val_accuracy: 0.8078
Epoch 194/200
240/240 [==============================] - 39s 164ms/step - loss: 0.4380 - accuracy: 0.8148 - val_loss: 0.4461 - val_accuracy: 0.8281
Epoch 195/200
240/240 [==============================] - 37s 154ms/step - loss: 0.4414 - accuracy: 0.8135 - val_loss: 0.4341 - val_accuracy: 0.8281
Epoch 196/200
240/240 [==============================] - 39s 163ms/step - loss: 0.4168 - accuracy: 0.8169 - val_loss: 0.3448 - val_accuracy: 0.8703
Epoch 197/200
240/240 [==============================] - 39s 162ms/step - loss: 0.4344 - accuracy: 0.8180 - val_loss: 0.4286 - val_accuracy: 0.8336
Epoch 198/200
240/240 [==============================] - 37s 155ms/step - loss: 0.4469 - accuracy: 0.8078 - val_loss: 0.3863 - val_accuracy: 0.8531
Epoch 199/200
240/240 [==============================] - 39s 163ms/step - loss: 0.4137 - accuracy: 0.8247 - val_loss: 0.3892 - val_accuracy: 0.8586
Epoch 200/200
240/240 [==============================] - 39s 162ms/step - loss: 0.4085 - accuracy: 0.8279 - val_loss: 0.3486 - val_accuracy: 0.8734
40/40 [==============================] - 4s 86ms/step - loss: 0.3424 - accuracy: 0.8750
Test loss: 0.34
Test accuracy: 87.5 %
```

Figure 14: Accuracy and loss value

```
import matplotlib.pyplot as plt

# summarize history for accuracy
plt.plot(history.history['accuracy'],label="train_acc")
plt.plot(history.history['val_accuracy'],label="val_acc")
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig("/content/drive/MyDrive/Alzheimer disease Classification/accuracyplot.png")
plt.show()
# summarize history for loss
plt.plot(history.history['loss'],label="train_loss")
plt.plot(history.history['val_loss'],label="val_loss")
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig("/content/drive/MyDrive/Alzheimer disease Classification/lossplot.png")
plt.show()
```

Figure 15: Accuracy and loss

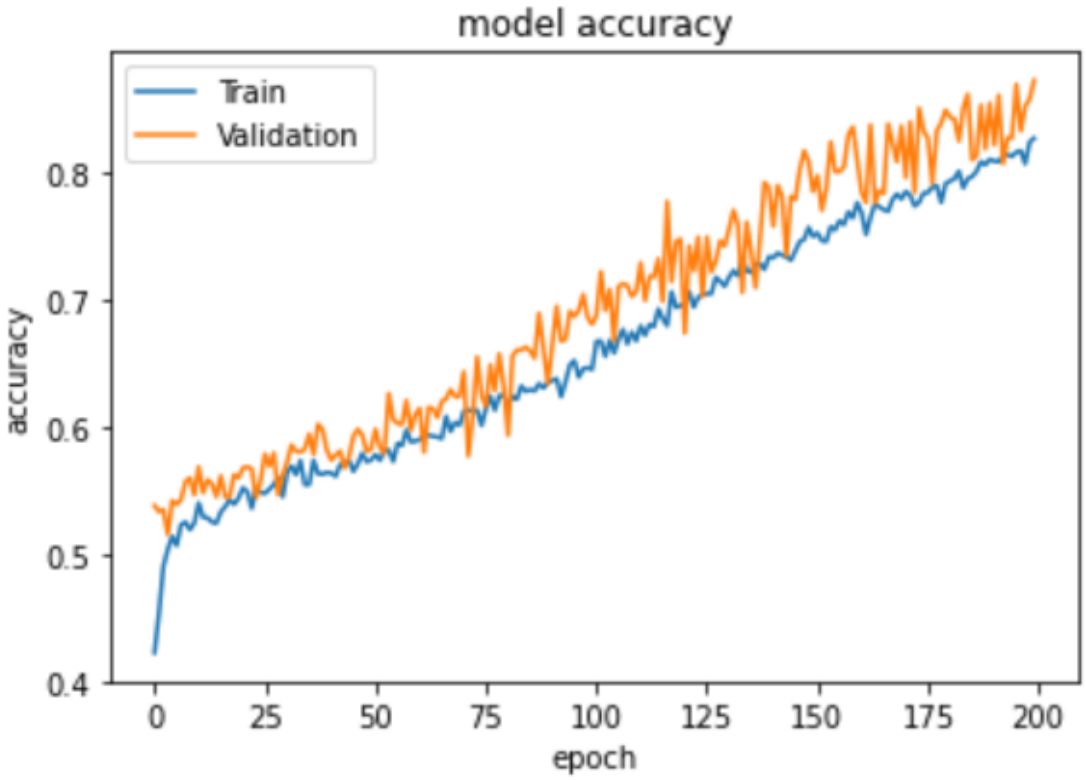

Figure 16: Accuracy graph

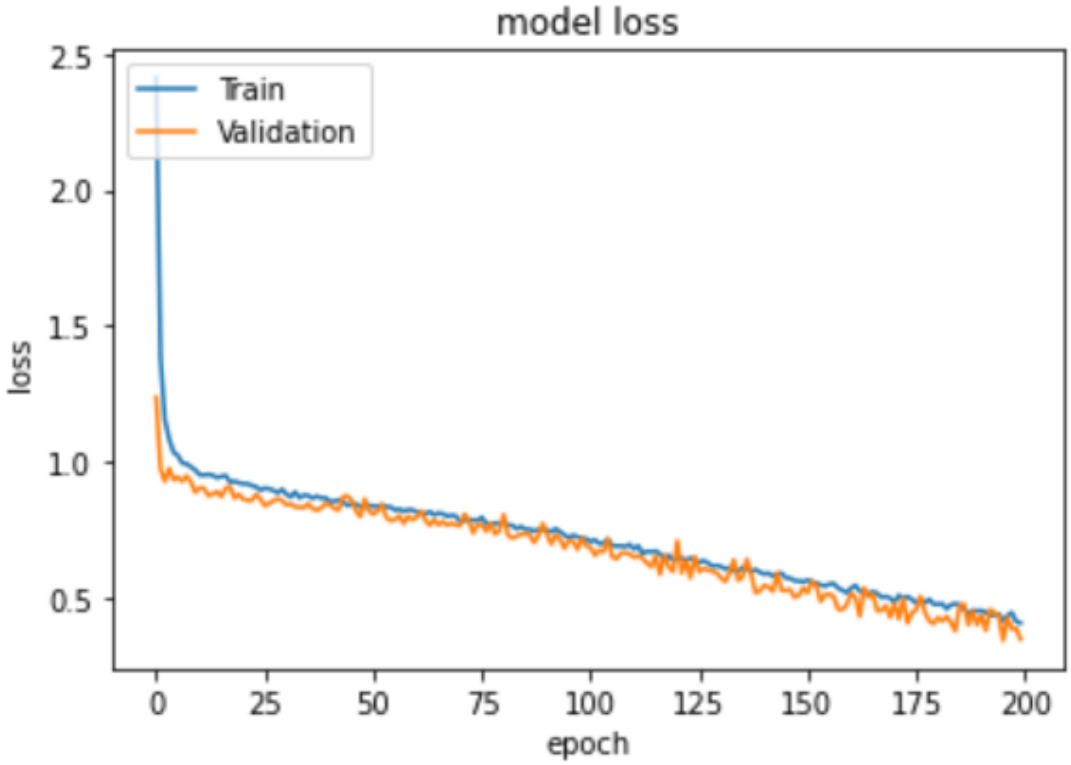

Figure 17: Loss graph

```
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='g')
plt.savefig("/content/drive/MyDrive/Alzheimer disease Classification/CM.png")
```

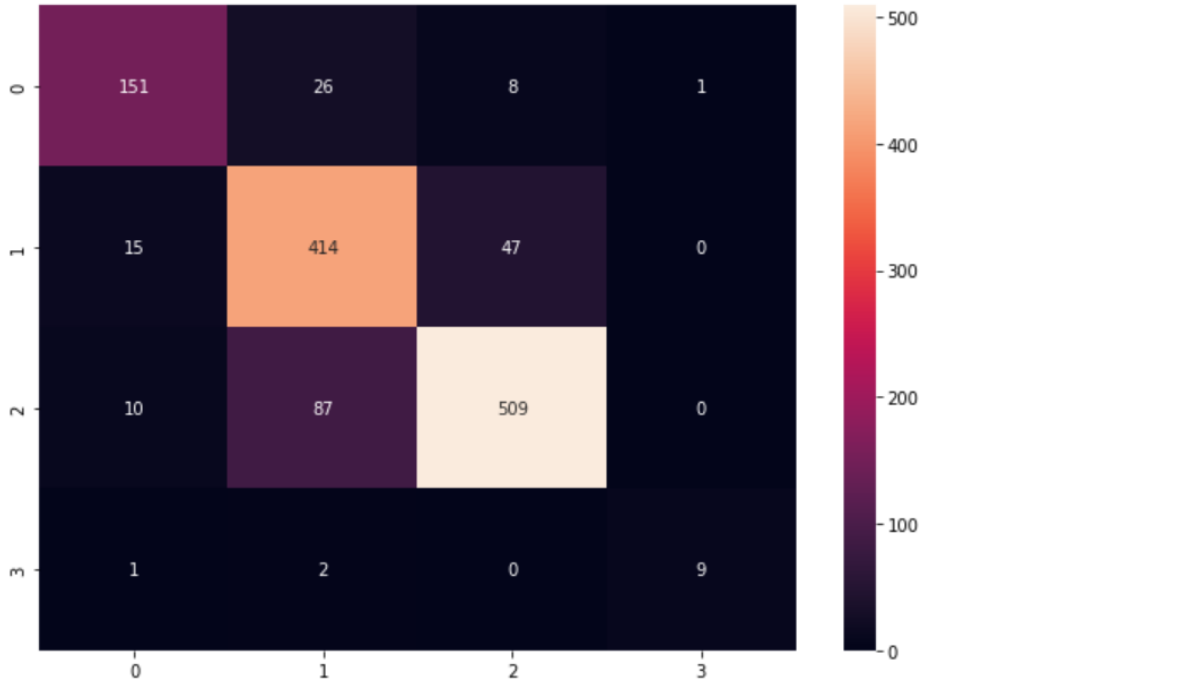

Figure 18: Confusion matrix

```
classes = ['Mild_Demented','Very_Mild_Demented','Non_Demented','Moderate_Demented']
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred,target_names=classes))
```

```
                    precision    recall  f1-score   support

     Mild_Demented       0.85      0.81      0.83       186
Very_Mild_Demented       0.78      0.87      0.82       476
      Non_Demented       0.90      0.84      0.87       606
 Moderate_Demented       0.90      0.75      0.82        12

          accuracy                           0.85      1280
         macro avg       0.86      0.82      0.84      1280
      weighted avg       0.85      0.85      0.85      1280
```

Figure 19: Classification report