Configuration Manual

MSc Research Project
MSc Data Analytics

Bhagya Vinod
Student ID: 22106111

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Bhagya Vinod
Student ID:	x22106111
Programme:	MSc Data Analytics **Year:** 2023
Module:	MSc Research Project
Lecturer:	Dr. Chrsitian Horn
Submission Due Date:	14/12/2023
Project Title:	Early Detection of Alzheimer's using Deep Learning Techniques

Word Count: **Page Count: 9**

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: Bhagya Vinod

Date: 14/12/2023

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

Bhagya Vinod
X22106111

1 Introduction

This configuration manual lists all the software, hardware and underlying code needed to carry out the research project "Early Detection of Alzheimer's using Deep Learning Techniques".

2 System Configuration

2.1 Hardware

Processor	:	12th Gen Intel(R) Core(TM) i7-1255U 1.70 GHz
Installed RAM	:	12.0 GB (11.7 GB usable)
Device ID	:	28C1A32F-4A19-4630-950E-8B6FB9ABB445
Product ID	:	00342-42315-95217-AAOEM
System type	:	64-bit operating system, x64-based processor

2.2 Software

Software Computing Tools Used: Python 3.9.13, Jupyter Notebook, Microsoft Word

3 Project Development

The figure depicts the overall steps followed to conduct the project

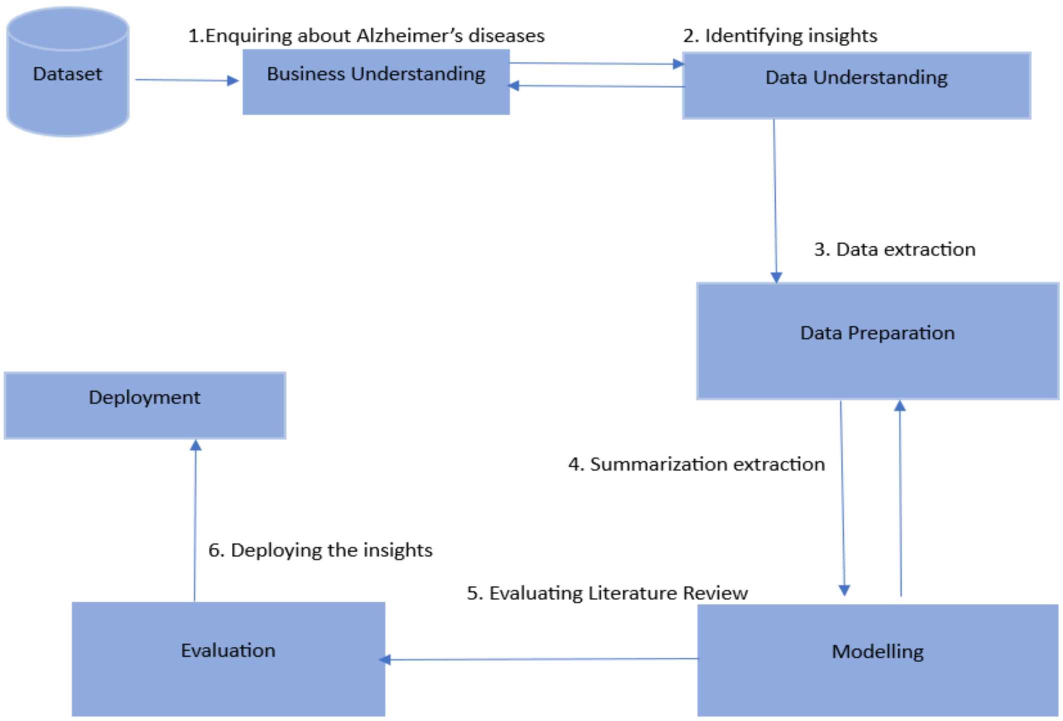

Figure1: Methods followed to conduct the study

4 Deep Learning Code

4.1 Importing Necessary Library

Pandas: To work with datasets, Python's Pandas package is utilized. It provides tools for exploring, cleaning, analysing and manipulating data.

TensorFlow: A platform that facilitates the application of optimal techniques for modelling, data processing, tracking performance and training models.

PIL: The Pillow library has every essential feature needed for image processing

```python
import numpy as np
import pandas as pd
import random
import os
from pathlib import Path
from PIL import Image
import tensorflow as tf
from random import randint
from tensorflow import keras
import tensorflow_addons as tfa
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
```

```python
from tensorflow.keras.layers import Conv2D, Flatten
from sklearn.model_selection import train_test_split
from sklearn.metrics import matthews_corrcoef as MCC
from distutils.dir_util import copy_tree, remove_tree
from sklearn.metrics import balanced_accuracy_score as BAS
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.applications.densenet import DenseNet169
from tensorflow.keras.applications.inception_v3 import InceptionV3
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications.efficientnet import EfficientNetB3
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
from tensorflow.keras.layers import SeparableConv2D, BatchNormalization, MaxPool2D, GlobalAveragePooling2D
```

Figure 2: Library Import

4.2 Hyperparameters and Image Settings for Neural Network Training

```python
EPOCHS = 100
BATCH_SIZE = 32
IMG_SIZE = 128
IMAGE_SIZE = [128, 128]
DIM = (IMG_SIZE, IMG_SIZE)
```

Figure 3: Settings for Training

The above code sets hyperparameters and image-related constants for training a neural network.

4.3 Loading Training Data File Paths and Labels

```python
train_dt = Path('./dataset/Combined Dataset/train')
filepaths = list(train_dt.glob(r'**/*.jpg'))
labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))
```

Figure 4: Training Data path

This code snippet utilizes Path module to create a relative path to the training dataset directory. The training dataset is stored at location ‘./dataset/Combined Dataset/train’. Following this, it makes use of this path to get a list of file paths for each JPG file found in the specified directory and all of its subdirectories.

```
train_df.head()
```

	Filepath	Label
0	dataset\Combined Dataset\train\Moderate Impair...	Moderate Impairment
1	dataset\Combined Dataset\train\No Impairment\N...	No Impairment
2	dataset\Combined Dataset\train\Mild Impairment...	Mild Impairment
3	dataset\Combined Dataset\train\Mild Impairment...	Mild Impairment
4	dataset\Combined Dataset\train\Moderate Impair...	Moderate Impairment

Figure 5: Subset of training Dataset

4.4 Loading Testing Data File Paths and Labels

```
test_dir = Path('./dataset/Combined Dataset/test')
```

```
filepaths = list(test_dir.glob(r'**/*.jpg'))
labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))
filepaths = pd.Series(filepaths, name='Filepath').astype(str)
labels = pd.Series(labels, name='Label')
```

```
test_df = pd.concat([filepaths, labels], axis=1)
test_df = test_df.sample(frac=1).reset_index(drop = True)
```

```
test_df.head()
```

	Filepath	Label
0	dataset\Combined Dataset\test\Very Mild Impair...	Very Mild Impairment
1	dataset\Combined Dataset\test\Very Mild Impair...	Very Mild Impairment
2	dataset\Combined Dataset\test\Very Mild Impair...	Very Mild Impairment
3	dataset\Combined Dataset\test\Very Mild Impair...	Very Mild Impairment
4	dataset\Combined Dataset\test\Very Mild Impair...	Very Mild Impairment

Figure 6: Testing Data path and subset of testing dataset

For the test dataset, a DataFrame called test_df is created in this code segment. Initially, file paths are gathered, and labels are extracted from the designated directory and all of its subdirectories. Two columns make up the resulting DataFrame: ‘Label’ for matching labels and ‘Filepath’ for image file paths.

4.5 Building a Convolutional Neural Network for Image Classification

```
Model: "cnn_model"
_________________________________________________________________________________________
 Layer (type)                               Output Shape              Param #
=========================================================================================
 conv2d (Conv2D)                            (None, 128, 128, 16)      448

 conv2d_1 (Conv2D)                          (None, 128, 128, 16)      2320

 max_pooling2d (MaxPooling2D)               (None, 64, 64, 16)        0

 conv2d_2 (Conv2D)                          (None, 64, 64, 32)        4640

 conv2d_3 (Conv2D)                          (None, 64, 64, 32)        9248

 batch_normalization (BatchNormalization)   (None, 64, 64, 32)        128

 max_pooling2d_1 (MaxPooling2D)             (None, 32, 32, 32)        0

 conv2d_4 (Conv2D)                          (None, 32, 32, 64)        18496

 conv2d_5 (Conv2D)                          (None, 32, 32, 64)        36928

 batch_normalization_1 (BatchNormalization) (None, 32, 32, 64)        256

 max_pooling2d_2 (MaxPooling2D)             (None, 16, 16, 64)        0

 conv2d_6 (Conv2D)                          (None, 16, 16, 128)       73856

 conv2d_7 (Conv2D)                          (None, 16, 16, 128)       147584

 batch_normalization_2 (BatchNormalization) (None, 16, 16, 128)       512

 max_pooling2d_3 (MaxPooling2D)             (None, 8, 8, 128)         0

 conv2d_8 (Conv2D)                          (None, 8, 8, 256)         295168

 last_conv_layer (Conv2D)                   (None, 8, 8, 256)         590080

 batch_normalization_3 (BatchNormalization) (None, 8, 8, 256)         1024

 max_pooling2d_4 (MaxPooling2D)             (None, 4, 4, 256)         0

 flatten (Flatten)                          (None, 4096)              0

 dropout (Dropout)                          (None, 4096)              0

 dense (Dense)                              (None, 512)               2097664

 batch_normalization_4 (BatchNormalization) (None, 512)               2048

 dropout_1 (Dropout)                        (None, 512)               0

 dense_1 (Dense)                            (None, 128)               65664

 batch_normalization_5 (BatchNormalization) (None, 128)               512

 dropout_2 (Dropout)                        (None, 128)               0

 dense_2 (Dense)                            (None, 64)                8256

 batch_normalization_6 (BatchNormalization) (None, 64)                256

 dropout_3 (Dropout)                        (None, 64)                0

 dense_3 (Dense)                            (None, 4)                 260

=========================================================================================
Total params: 3355348 (12.80 MB)
Trainable params: 3352980 (12.79 MB)
Non-trainable params: 2368 (9.25 KB)
```

Figure 7 : CNN Architecture

For the purpose of classifying images, the above code defines a Convolutional Neural Network(CNN) using the Keras Sequential API (Joseph, et al., 2021)The architecture consists of dense layers, max-pooling layers, batch normalization, dropout for regularization and multiple convolutional layers with rectified linear unit(ReLU) activation. With four output classes, the CNN is built for a multi-class classification task, and the softmax activation function is used in the final layer.

4.6 Compiling the Custom Convolutional Neural Network

```
OPT = tf.keras.optimizers.Adam(learning_rate=0.001)

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

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

Figure 8: Training process of CNN

This code snippet configures the training process for the previously defined custom CNN model. The following steps are performed:

- The learning rate of 0.001 is used to instantiate the Adam optimizer. During training, the optimizer is in charge of changing the model's weights based on the established gradients.
- During training, a set of metrics will be chosen to be monitored. The metrics chosen in this instance are F1 score, area under the curve (AUC), and categorical accuracy.
- Using the custom CNN model, the compile method is invoked. It details the metrics, loss function, and optimizer that will be applied during training.

4.7 Implementing Early Stopping and Model Checkpointing

```
In [19]: earlystopping = EarlyStopping(monitor = 'val_loss',
                                       mode = 'min',
                                       patience = 10,
                                       verbose = 1)

         filepath = './best_weights.hdf5'

         checkpoint    = ModelCheckpoint(filepath,
                                         monitor = 'val_loss',
                                         mode='min',
                                         save_best_only=True,
                                         verbose = 1)


         callback_list = [earlystopping, checkpoint]
```

Figure 9: Callback Configuration for Early Stopping and Model Checkpointing

The above snippet sets up callbacks for early stopping and model checkpointing during the training of a neural network. Neural networks are often trained using callbacks in order to avoid overfitting and maintain the optimal model weights depending on validation performance. Model checkpointing saves the model with the best validation performance for later use, while early stopping supports in ending training if the model's performance on the validation set stops improving.

4.8 Transfer Learning with ResNet50 for Image Classification

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(*IMAGE_SIZE, 3))
for layer in base_model.layers:
    layer.trainable = False
model_tl = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(1024, activation='relu'),
    Dropout(0.2),
    Dense(512, activation='relu'),
    Dropout(0.2),
    Dense(4, activation='softmax')
], name="transfer_learning_model")
model_tl.compile(optimizer=OPT, loss='categorical_crossentropy', metrics=METRICS)
```

Figure 10: Transfer Learning Model Configuration

The above-mentioned code demonstrates how to utilize transfer learning to image classification using the ResNet50 pre-trained model.

5. Result

5.1 Training Progress and Early Stopping

```
Epoch 24: val_loss did not improve from 0.16999
320/320 [==============================] - 175s 547ms/step - loss: 0.0329 - acc: 0.9899 - auc: 0.9994 - f1_score: 0.9899 - v
al_loss: 3.6875 - val_acc: 0.4464 - val_auc: 0.6629 - val_f1_score: 0.3120
Epoch 25/100
320/320 [==============================] - ETA: 0s - loss: 0.0472 - acc: 0.9846 - auc: 0.9989 - f1_score: 0.9846
Epoch 25: val_loss did not improve from 0.16999
320/320 [==============================] - 173s 541ms/step - loss: 0.0472 - acc: 0.9846 - auc: 0.9989 - f1_score: 0.9846 - v
al_loss: 1.3466 - val_acc: 0.7123 - val_auc: 0.8953 - val_f1_score: 0.6277
Epoch 26/100
320/320 [==============================] - ETA: 0s - loss: 0.0406 - acc: 0.9881 - auc: 0.9992 - f1_score: 0.9881
Epoch 26: val_loss did not improve from 0.16999
320/320 [==============================] - 173s 542ms/step - loss: 0.0406 - acc: 0.9881 - auc: 0.9992 - f1_score: 0.9881 - v
al_loss: 0.8556 - val_acc: 0.7936 - val_auc: 0.9380 - val_f1_score: 0.7727
Epoch 27/100
320/320 [==============================] - ETA: 0s - loss: 0.0534 - acc: 0.9818 - auc: 0.9989 - f1_score: 0.9818
Epoch 27: val_loss did not improve from 0.16999
320/320 [==============================] - 175s 547ms/step - loss: 0.0534 - acc: 0.9818 - auc: 0.9989 - f1_score: 0.9818 - v
al_loss: 0.9830 - val_acc: 0.7522 - val_auc: 0.9173 - val_f1_score: 0.8294
Epoch 27: early stopping
```

Figure 11: Training and Evaluation Summary

The training was terminated early at epoch 27 because validation loss did not improve further, avoiding the possibility of overfitting and preserving the optimal model weights for use at a later time.

5.2 Evaluating Model Performance on Test Data

```
test_scores = custom_model_combined.evaluate(test_images)
print("Testing Accuracy: %.2f%%"%(test_scores[1] * 100))
pred_labels = custom_model_combined.predict(test_images)

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)

pred = np.argmax(pred_labels,axis=1)

print(classification_report(test_images.classes,pred,target_names=CLASSES))
```

```
40/40 [==============================] - 8s 193ms/step - loss: 0.9830 - acc: 0.7522 - auc: 0.9173 - f1_score: 0.8294
Testing Accuracy: 75.22%
40/40 [==============================] - 8s 187ms/step
                      precision    recall  f1-score   support

     Mild Impairment       0.90      0.96      0.92       179
 Moderate Impairment       1.00      0.92      0.96        12
       No Impairment       0.99      0.53      0.69       640
Very Mild Impairment       0.60      0.98      0.74       448

            accuracy                           0.75      1279
           macro avg       0.87      0.85      0.83      1279
        weighted avg       0.84      0.75      0.75      1279
```

Figure 12: Model Evaluation and Classification Report

This code is crucial for determining how well the trained model performs across different categories and for evaluating how well it generalizes to new, unseen data.

5.3 Evaluation Metrics for Model Performance

The Balanced Accuracy Score and Matthew's Correlation Coefficient are two more evaluation metrics that are computed and printed by the snippet of code below

```
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: 84.62 %
Matthew's Correlation Coefficient: 66.93 %
```

Figure 13: Accuracy of CNN Model

By calculating the average sensitivity and specificity for each class, the Balanced Accuracy Score is a metric that accounts for dataset imbalances.

5.4 Transfer Learning Model Training and Evaluation

```
history_tl = model_tl.fit(
    train_images,
    validation_data=test_images,
    epochs=EPOCHS,
    callbacks=callback_list
)
test_scores_tl = model_tl.evaluate(test_images)
print("Transfer Learning Model Testing Accuracy: %.2f%%" % (test_scores_tl[1] * 100))

# Predicting and evaluating performance
pred_labels_tl = model_tl.predict(test_images)
pred_tl = np.argmax(pred_labels_tl, axis=1)

print(classification_report(test_images.classes, pred_tl, target_names=CLASSES))
print("Transfer Learning Balanced Accuracy Score: {} %".format(round(BAS(test_images.classes, pred_tl) * 100, 2)))
print("Transfer Learning Matthew's Correlation Coefficient: {} %".format(round(MCC(test_images.classes, pred_tl) * 100, 2)))
```

Figure 14: Transfer Learning Model Training and Evaluation

The previously defined ResNet50 based architecture is used to train a transfer learning model. The model's performance is evaluated on the test dataset and an in-depth report of the findings is given, along with testing accuracy, a balanced accuracy score, a classification report, and Matthew's correlation coefficient.

```
320/320 [==============================] - 335s 1s/step - loss: 0.7283 - acc: 0.6651 - auc: 0.9016 - f1_score: 0.6563 - val_
loss: 0.9488 - val_acc: 0.5152 - val_auc: 0.8200 - val_f1_score: 0.4008
Epoch 30: early stopping
40/40 [==============================] - 42s 1s/step - loss: 0.9488 - acc: 0.5152 - auc: 0.8200 - f1_score: 0.4008
Transfer Learning Model Testing Accuracy: 51.52%
40/40 [==============================] - 44s 1s/step
                      precision    recall  f1-score   support

     Mild Impairment       0.34      0.55      0.42       179
 Moderate Impairment       0.16      0.75      0.26        12
       No Impairment       0.63      0.75      0.68       640
Very Mild Impairment       0.42      0.17      0.24       448

            accuracy                           0.52      1279
           macro avg       0.39      0.55      0.40      1279
        weighted avg       0.51      0.52      0.49      1279

Transfer Learning Balanced Accuracy Score: 55.34 %
Transfer Learning Matthew's Correlation Coefficient: 23.51 %
```

Figure 15: Transfer Learning Model Evaluation Results

The above snippet shows the training process of a transfer learning model using the ResNet50 architecture with a focus on early stopping. The transfer learning model is trained over a predetermined number of epochs, showing metrics for training and validation such as accuracy, loss, AUC and F1-score at each epoch. In order to prevent overfitting, early stopping is used to monitor the validation loss and stop training if it does not improve after a predetermined number of epochs.

5 References

Joseph, F. J. J., Nonsiri, S. & Monsakul, A., 2021. *Keras and TensorFlow: A Hands-On Experience.* s.l.:s.n.