Configuration Manual

MSc Research Project
Data Analytics

Animesh Kumar
Student ID: X18184731

School of Computing
National College of Ireland

Supervisor: Dr. Paul Stynes

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Animesh Kumar
Student ID:	X18184731
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Paul Stynes
Submission Due Date:	28/09/2020
Project Title:	Configuration Manual
Word Count:	931
Page Count:	18

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:	Animesh Kumar
Date:	27th September 2020

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

Animesh Kumar
X18184731

1 Introduction

This configuration manual frames the various software and hardware specifications and their versions used while implementing procedures of research topic "Brain Age Classification from Brain MRI using ConvCaps Framework". This work will help the future researchers to replicate the research work for further analysis and extension without any difficulties.

2 System Configuration

2.1 Hardware specification

This is the current system hardware configuration which facilitates an Intel i7-8550U processor with a max clock speed of 1.99GHz.Figure 1

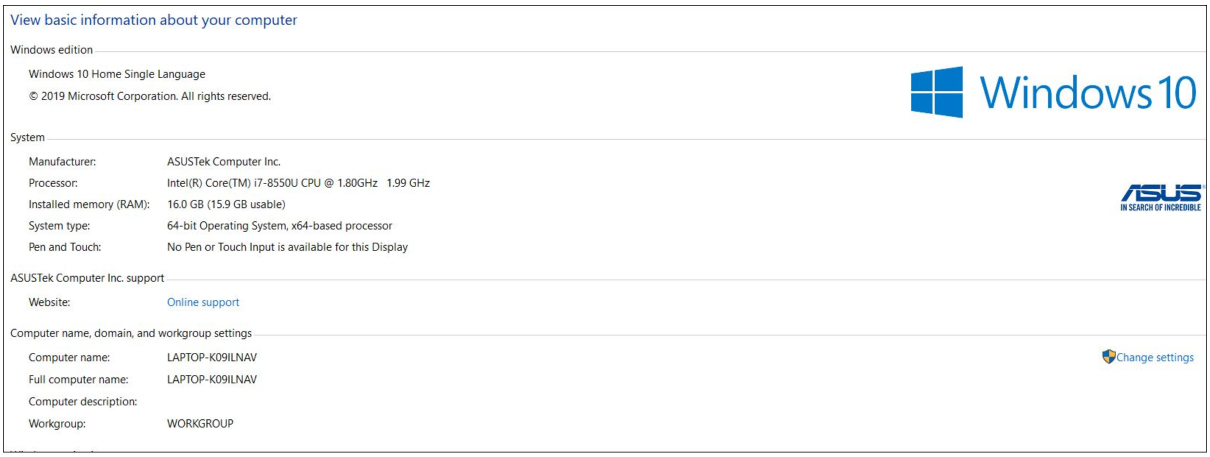

Figure 1: Hardware specification of the system

2.2 Software specification

Below are the software specification used while executing implementation procedures.

2.2.1 Python 3.7.3

Latest version of Python 3.7.3 has been used for this research.

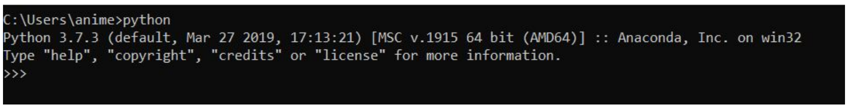

Figure 2: Python version

2.2.2 Google Colaboratory

All implementation are performed on Google Colab notebook platform. It is a cloud based platform, which provides set of GPU's and CPU's to process code faster and reduces computational time. For deep learning models GPU's are highly recommended as with increase in data size, model run time with rise. All data related to the project were uploaded in google drive for faster retrieval. Figure 3

Figure 3: Google colaboratory sign-in

2.2.3 Anaconda

Anaconda app suite is freely available platform for python application. It facilitates python notebook know as Jupyter. Jupyter notebook is a well verse python IDE (Integrated Development Suite). Some data pre-processing were performed on Jupyter, due to storage limitation of google drive.Figure 4

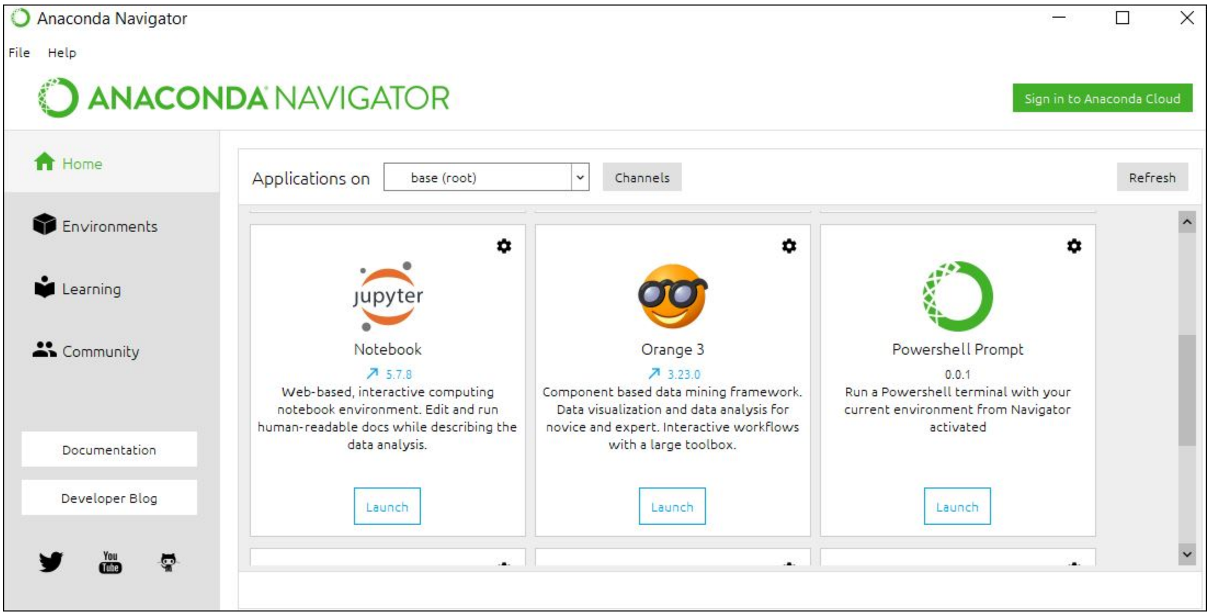

Figure 4: Anaconda app suite

2.2.4 Overleaf

All reporting and explanation of research were performed on Overleaf.Figure 5

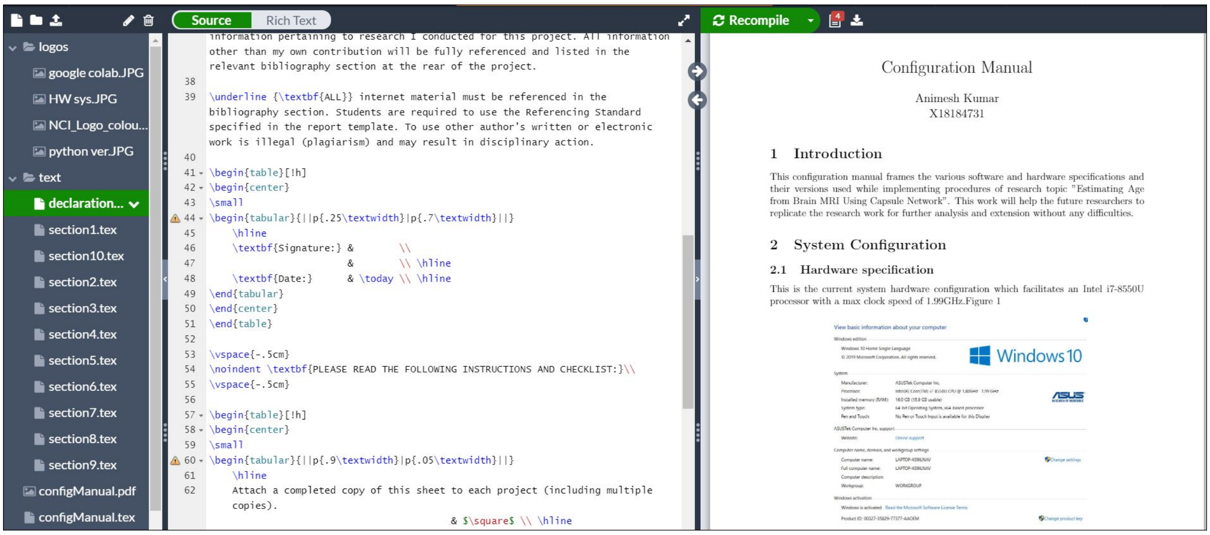

Figure 5: Overleaf GUI

2.2.5 Libraries

Python is well known for its libraries for any utility. This research comprises libraries like Tensorflow, Keras, CV2, Shutil and PIL to name the few.Figure 6

```python
import numpy as np
import os
import pandas as pd
from keras.preprocessing.image import ImageDataGenerator
from keras import callbacks
from keras.utils.vis_utils import plot_model
import cv2
import random
import shutil
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
import zipfile
import matplotlib.pyplot as plt
from tqdm import tqdm, tqdm_notebook
import warnings
from keras.utils import to_categorical

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
```

Figure 6: Python libraries

3 Data Collection and Preparation

Data is collected from OASIS (Open Access Series of Imaging Studies) Marcus et al. (2007) which provides free subscription for all datasets on registering to their website. The dataset contained images and CSV demographics with detail like age, dementia rating, gender and etc.The data was present in different folders out of which only images from FSLSEG and PREPROCESSED were collected for this research as shown in figure Figure 7.

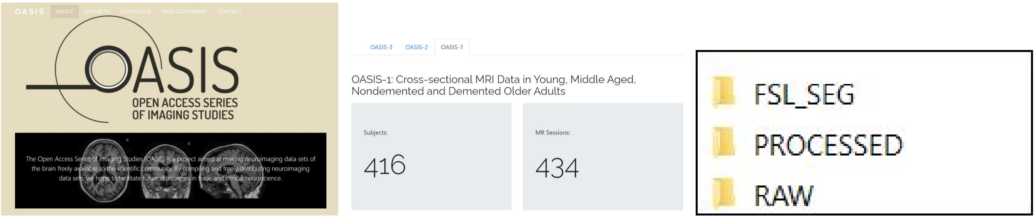

Figure 7: (a) OASIS (b) OASIS-1 Dataset (c) Folder

Data fetched from sub-folders FSLSEG and PREPROCESSSED as shown in Figure 8.

Data from below folders get extracted in a new folder [OASIS_data_image]

1. FSL_SEG: Contain GM, WM, CSF segmented image generated from the masked atlas image
2. PREPROCESSED : Contain folder (T88_111) which includes the atlas-registered field-corrected gain images and a brain-masked image version resampled to isotropic voxels of 1mm .

```
[ ] #Copying required files from original OASIS dataset to new folder

    def file_extract(src,dest):
      for file_name in os.listdir(src):
          sub_dir_path = directory + file_name
          fn = file_name[:-4]
          proc_dir_path = (sub_dir_path + '\\' + fn)
          file_copy(proc_dir_path,fn,dest)

    def file_copy(path,file,dest):

        for folder in os.listdir(path):
            # Copying files from preprocessed path
            preprocessed_path = glob.glob(path + '\\' + folder + '\PROCESSED\MPRAGE\\T88_111\\**anon_111_t88_masked_gfc_tra_90.gif') #constructing required path
            preprocessed_path1 = preprocessed_path[0]
            img_preprocessed = Image.open(preprocessed_path1)
            img_preprocessed.save (dest  + folder + 'pp.gif' ,'gif') # Saving to folder

            #Copying file from FSL_SEG path
            FSL_SEG_path = glob.glob(path + '\\' + folder + '\FSL_SEG\**anon_111_t88_masked_gfc_fseg_tra_90.gif') #constructing required path
            FSL_SEG_path1 = FSL_SEG_path[0]
            img_mask = Image.open(FSL_SEG_path1)
            img_mask.save (dest + folder + 'fsl.gif','gif') # Saving to folder
```

Figure 8: Loaded data in (.gif) format

3.1 Data Storage

Images were stored in GIPHY format first to the local system as shown in Figure 9. The stored data were converted into PNG format which reduced the size of images to 1/4th of original data as shown in Figure 11.

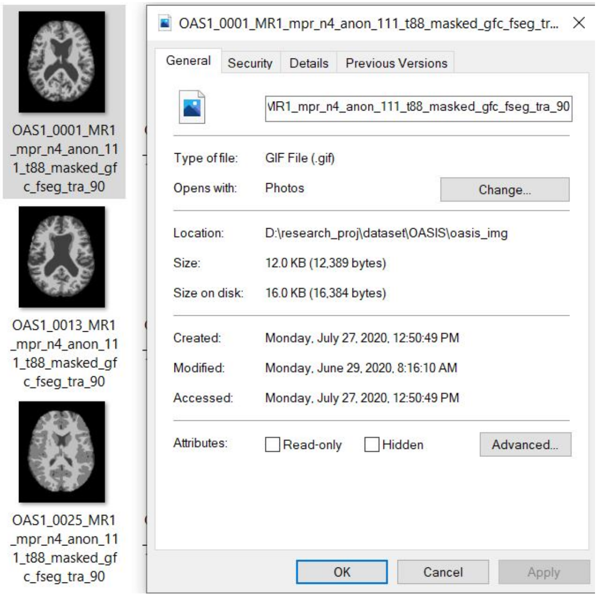

Figure 9: Loaded data in (.gif) format

3.2 Outlier check and removal

Images were checked for outliers like blank images. Code is shown in Figure 10.

```
#removing outlier with blank image of shape (208,176) and removing corresponding rows from dataframe

import cv2
import os

#Refrence blank image for comparison
blank_image = np.zeros((208,176), np.uint8)
blank_image.fill(255)
blank_image.shape

src = 'D:/research_proj/dataset/oasis_1_6/data_oasis_copy/' #source folder

for name in os.listdir(src):
    file_name = src + name # file path
    file_load = cv2.imread(file_name, cv2.IMREAD_UNCHANGED) # load file

    if file_load.shape == blank_image.shape: # comparing with blank image

        oasis_df[oasis_df.ID != name[0:13]] # remove image id row from dataframe
        os.remove(file_name) # remove file from folder
```

Figure 10: Outlier check

3.3 Data Conversion

Images are stored in GIPHY format first to the local system as shown in Figure 9.

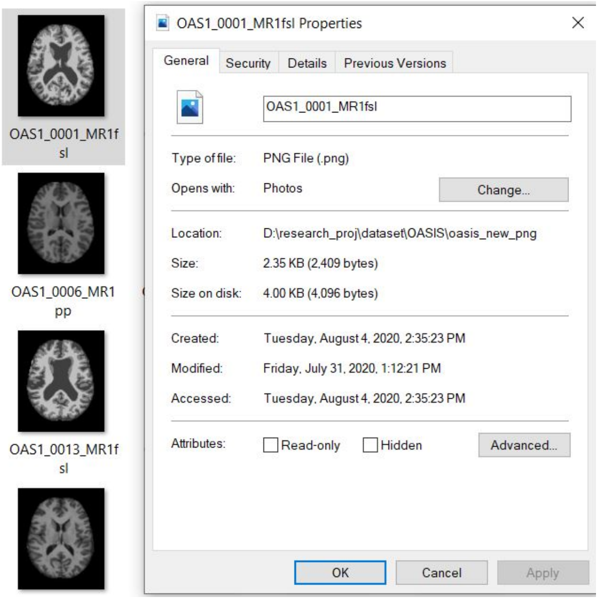

Figure 11: Data conversion in (.png) format

3.4 Demographic Storage and Conversion

Data demographics was present with several details as mentioned above however, only image ID and Age were taken for this study as shownFigure 12.The data were further classified into 6 classes as shown in Figure 13

```
df_oasis = pd.read_csv("/content/oasis_cross-sectional.csv")
df_oasis.head()
```

	ID	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF	Delay
0	OAS1_0001_MR1	F	R	74	2.0	3.0	29.0	0.0	1344	0.743	1.306	NaN
1	OAS1_0002_MR1	F	R	55	4.0	1.0	29.0	0.0	1147	0.810	1.531	NaN
2	OAS1_0003_MR1	F	R	73	4.0	3.0	27.0	0.5	1454	0.708	1.207	NaN
3	OAS1_0004_MR1	M	R	28	NaN	NaN	NaN	NaN	1588	0.803	1.105	NaN
4	OAS1_0005_MR1	M	R	18	NaN	NaN	NaN	NaN	1737	0.848	1.010	NaN

Figure 12: Loaded data demographic

```
df_oasis_cls = pd.read_csv("/content/drive/My Drive/(
df_oasis_cls.head()
```

	ID	category
0	OAS1_0001_MR1	5
1	OAS1_0002_MR1	3
2	OAS1_0003_MR1	5
3	OAS1_0004_MR1	2
4	OAS1_0005_MR1	1

Figure 13: Data demographic converted into classes from 1 to 6

3.5 Test Train Split

Test data is split from train data before performing augmentation so as to get real testing accuracy of the model. The split ratio was 0.2 Figure 14.

Test_Train Split

Test data is taken out from entire dataset before augmentation

Testing ratio = 0.20.

```
[ ] from sklearn.model_selection import train_test_split

    train, test = train_test_split(oasis_data,test_size=0.20, random_state=42,shuffle=True)
```

Figure 14: Test train split

3.6 Data Augmentation

Images were augmented in 12 different filters Mikolajczyk and Grochowski (2018) as shown in Figure 15Figure 16.

```
dest = "/content/drive/My Drive/Augmentation/oasis_augmentation/" # Destination augmentation folder


def augmentation(path):

  for image in os.listdir(path):

    img_path = os.path.join(path + image)
    img_load = cv2.imread(img_path)

    #img_load = tf.convert_to_tensor(img_inp, dtype=None, dtype_hint=None, name=None)

    #flipping right to left
    flippedrl = tf.image.flip_left_right(img_load)
    fliprl = np.asarray(flippedrl)
    cv2.imwrite(dest + image[:-4] + '_fliprl' + '.png', fliprl)

    #rotating by 90 degree
    rotated = tf.image.rot90(img_load)
    rot = np.asarray(rotated)
    cv2.imwrite(dest + image[:-4] + '_rotate90' + '.png', rot)

    #flipping up to down
    flippedud = tf.image.flip_up_down(img_load)
    flipud = np.asarray(flippedud)
    cv2.imwrite(dest + image[:-4] + '_flipud' + '.png', flipud)

    #cropping by 0.8 feaction
    cropped = tf.image.central_crop(img_load, central_fraction=0.8)
    crop = np.asarray(cropped)
```

Figure 15: Augmentation code

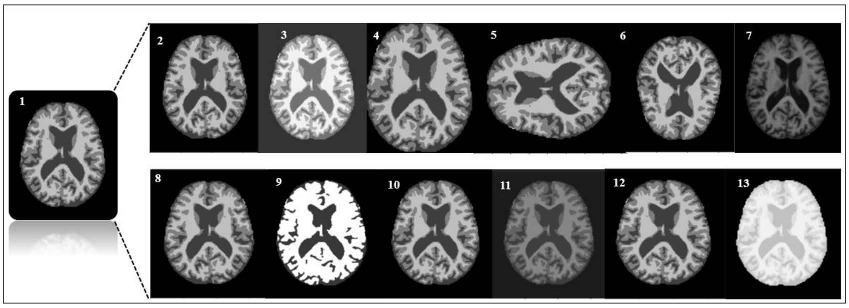

Figure 16: Data augmentation: - 1) Original Data 2) Left-Right Flip 3) Brightness(0.2) 4) Center Cropping (0.8) 5) Rotation 90 6) Upside Down 7) Random Contrast 8) Saturation (10) 9) Adjust Contrast (8), 10) Random Hue 11) Segmented 12) Random Gamma 13) Random Saturation

Augmented images are further divided into different classes from class 1 to class 6 in respective folders Figure 17.

Figure 17: (a) Class division code (b) Different class folders

Class folders are divided into train and validation folder as shown in which was fed as input for InceptionV3 and DenseNet architectures Figure 18.

Figure 18: (a) Train validation split code (b) Train and validation folders

4 Model Implementation

There are four models implemented under this project. The state-of-the-art is the baseline model which was replicated under baseline implementation 1 followed by the novel architecture proposed in this research as Convolutional Capsule Network. The pre-trained models like InceptionV3 and DenseNet were used for model analysis and comparison.

4.1 Baseline model:Alexnet-CNN (State-of-the-art)

The model consist of convolutional layer block inspired from alexnet model as shown in Figure 19. And model run is shown in Figure 20.

```
def CNN_model():
  model = Sequential()
  model.add(Conv2D(filters = 16, kernel_size = 3, padding = 'same', activation = 'relu', input_shape = (224, 224, 3)))
  model.add(Dropout(0.3))
  model.add(MaxPooling2D(pool_size = 3))

  model.add(Conv2D(filters = 32, kernel_size = 3, padding = 'same', activation = 'relu'))
  model.add(Dropout(0.3))
  model.add(MaxPooling2D(pool_size = 3))

  model.add(Conv2D(filters = 64, kernel_size = 3, padding = 'same', activation = 'relu'))
  model.add(Dropout(0.3))
  model.add(MaxPooling2D(pool_size = 3))

  model.add(Conv2D(filters = 128, kernel_size = 3, padding = 'same', activation = 'relu'))
  model.add(Dropout(0.3))
  model.add(MaxPooling2D(pool_size = 3))

  model.add(Flatten())
  model.add(Dense(1024, activation='relu'))
  model.add(Dropout(0.3))
  model.add(Dense(512, activation='relu'))
  model.add(Dropout(0.3))
  model.add(Dense(9, activation = 'softmax'))
```

Figure 19: Alexnet-CNN modelling

```
model = CNN_model()

epochs = 50

STEP_SIZE_TRAIN=train.n//train.batch_size
STEP_SIZE_VALID=valid.n//valid.batch_size

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=12, restore_best_weights=Tru

time_callback = TimeHistory()

def scheduler(epoch):
# This function keeps the learning rate at 0.001 for the first ten epochs
# and decreases it exponentially after that.
  if epoch < 12:
    return 0.001
  else:
    return 0.001 * tf.math.exp(0.5 * (12 - epoch))

learning_rate_scheduler = tf.keras.callbacks.LearningRateScheduler(scheduler)

with tf.device('/GPU:0'):

# model training
  history = model.fit_generator(train,
                                steps_per_epoch=STEP_SIZE_TRAIN,
                                validation_data=valid,
                                validation_steps=STEP_SIZE_VALID,
                                epochs=100, callbacks= [time_callback])

WARNING:tensorflow:From <ipython-input-24-a6f8a0f90acd>:29: Model.fit_generator (from tensorflow.python.keras.e
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/50
103/103 [==============================] - 178s 2s/step - loss: 0.3297 - accuracy: 0.2362 - val_loss: 0.3336 -
Epoch 2/50
103/103 [==============================] - 173s 2s/step - loss: 0.3270 - accuracy: 0.2393 - val_loss: 0.3359 -
Epoch 3/50
103/103 [==============================] - 173s 2s/step - loss: 0.3242 - accuracy: 0.2661 - val_loss: 0.3331 -
Epoch 4/50
```

Figure 20: ALexnet-CNN model run

4.2 Proposed model:Convolutional Capsule Network

Model is the combination of Capsule network [1] and convolutional block. The CNN layers are at the starting for sub-sampling followed by CAPSNET for classification.Figure 26 denotes the modelling of ConvCaps with convolutional layers and capsule layer. The hyper-parameters considered for this model is shown in Figure 27. Also, in Figure 28 model run with steps per epoch is shown.

[1] https://github.com/XifengGuo/CapsNet-Keras

The proposed ConvCaps contains important class functions for individual working of the architecture as shown below in figure (21,22,23,24,25,26).

```python
class Length(layers.Layer):

    #Compute the length of vectors. This is used to compute a Tensor that has the same s

    def call(self, inputs, **kwargs):
        return K.sqrt(K.sum(K.square(inputs), -1))

    def compute_output_shape(self, input_shape):
        return input_shape[:-1]
```

Figure 21: Caps length formation layer

```python
class Mask(layers.Layer):
    """
    Mask a Tensor with shape=[None, d1, d2] by the max value in axis=1.
    Output shape: [None, d2]
    """
    def call(self, inputs, **kwargs):
        # use true label to select target capsule, shape=[batch_size, num_capsule]
        if type(inputs) is list:  # true label is provided with shape = [batch_size, n_clas
            assert len(inputs) == 2
            inputs, mask = inputs
        else:  # if no true label, mask by the max length of vectors of capsules
            x = inputs
            # Enlarge the range of values in x to make max(new_x)=1 and others < 0
            x = (x - K.max(x, 1, True)) / K.epsilon() + 1
            mask = K.clip(x, 0, 1)  # the max value in x clipped to 1 and other to 0

        # masked inputs, shape = [batch_size, dim_vector]
        inputs_masked = K.batch_dot(inputs, mask, [1, 1])
        return inputs_masked
```

Figure 22: Masking layer

```python
def squash(vectors, axis=-1):
    """
    The non-linear activation used in Capsule. It drives the length of a large vector to near 1
    :param vectors: some vectors to be squashed, N-dim tensor
    :param axis: the axis to squash
    :return: a Tensor with same shape as input vectors
    """
    s_squared_norm = K.sum(K.square(vectors), axis, keepdims=True)
    scale = s_squared_norm / (1 + s_squared_norm) / K.sqrt(s_squared_norm)
    return scale * vectors
```

Figure 23: Squashing layer

```python
class CapsuleLayer(layers.Layer):
    def __init__(self, num_capsule, dim_vector, num_routing=3,
                 kernel_initializer='glorot_uniform',
                 bias_initializer='zeros',
                 **kwargs):
        super(CapsuleLayer, self).__init__(**kwargs)
        self.num_capsule = num_capsule
        self.dim_vector = dim_vector
        self.num_routing = num_routing
        self.kernel_initializer = initializers.get(kernel_initializer)
        self.bias_initializer = initializers.get(bias_initializer)

    def build(self, input_shape):
        assert len(input_shape) >= 3, "The input Tensor should have shape=[None, input_num_capsule, input_dim_vector]"
        self.input_num_capsule = input_shape[1]
        self.input_dim_vector = input_shape[2]

        # Transform matrix
        self.W = self.add_weight(shape=[self.input_num_capsule, self.num_capsule, self.input_dim_vector, self.dim_vector],
                                 initializer=self.kernel_initializer,
                                 name='W')

        # Coupling coefficient. The redundant dimensions are just to facilitate subsequent matrix calculation.
        self.bias = self.add_weight(shape=[1, self.input_num_capsule, self.num_capsule, 1, 1],
                                    initializer=self.bias_initializer,
                                    name='bias',
                                    trainable=False)
        self.built = True
```

Figure 24: Main capsule layer

```python
def PrimaryCap(inputs, dim_vector, n_channels, kernel_size, strides, padding):
    """
    Apply Conv2D `n_channels` times and concatenate all capsules
    :param inputs: 4D tensor, shape=[None, width, height, channels]
    :param dim_vector: the dim of the output vector of capsule
    :param n_channels: the number of types of capsules
    :return: output tensor, shape=[None, num_capsule, dim_vector]
    """
    output = layers.Conv2D(filters=dim_vector*n_channels, kernel_size=kernel_size, strides=strides, padding=padding)(inputs)
    outputs = layers.Reshape(target_shape=[-1, dim_vector])(output)
    return layers.Lambda(squash)(outputs)
```

Figure 25: Primary caps layer

```python
from keras import layers, models
from keras import backend as K
from keras.utils import to_categorical
def CapsNet(input_shape, n_class, num_routing):
    """
    A Capsule Network on brain age data
    :param input_shape: data shape, 4d, [None, width, height, channels]
    :param n_class: number of classes
    :param num_routing: number of routing iterations
    :return: A Keras Model with 2 inputs and 2 outputs
    """
    x = layers.Input(shape=input_shape)

    # Conv1: Just a conventional Conv2D layer
    blk1_conv_64 = layers.Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu', name='conv_64')(x)
    blk1_conv_2_64 = layers.Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu', name='conv_2_64')(blk1_conv_64)
    max_pool_1_64 = layers.MaxPool2D(pool_size=(2,2),strides=(2,2),name='max_pool_64')(blk1_conv_2_64)

    # Conv2: For further sub sampling
    bkl2_conv_128 =layers.Conv2D(filters=128, kernel_size=3, strides=1, padding='same', activation='relu', name='conv_128')(max_pool_1_64)
    bkl2_conv_2_128 =layers.Conv2D(filters=128, kernel_size=3, strides=1, padding='same', activation='relu', name='conv_2_128')(bkl2_conv_128)
    max_pool_2_128 = layers.MaxPool2D(pool_size=(2,2),strides=(2,2), name='max_pool_128')(bkl2_conv_2_128)

    bkl3_conv_256 =layers.Conv2D(filters=256, kernel_size=3, strides=1, padding='same', activation='relu', name='conv_256')(max_pool_2_128)
    conv2_caps = layers.Conv2D(filters=256, kernel_size=9, strides=1, padding='valid', activation='relu', name='conv2')(bkl3_conv_256)#(max_pool_3_256)

    # Primary Caps layer: Conv2D layer with `squash` activation, then reshape to [None, num_capsule, dim_vector]
    primarycaps = PrimaryCap(conv2_caps, dim_vector=8, n_channels=32, kernel_size=9, strides=2, padding='valid')

    # Digit caps layer: Capsule layer. Routing algorithm works here.
    digitcaps = CapsuleLayer(num_capsule=n_class, dim_vector=16, num_routing=num_routing, name='digitcaps')(primarycaps)

    # Output caps layer: This is an auxiliary layer to replace each capsule with its length. Just to match the true label's shape.
    out_caps = Length(name='caps')(digitcaps)

    # Decoder network.
    y = layers.Input(shape=(n_class,))
    masked = Mask()([digitcaps, y])  # The true label is used to mask the output of capsule layer.

    x_recon = layers.Dense(512, activation='relu')(masked)
    x_recon = layers.Dense(1024, activation='relu')(x_recon)
    x_recon = layers.Dense(np.prod(input_shape), activation='softmax')(x_recon)
    x_recon = layers.Reshape(target_shape=input_shape, name='recon')(x_recon)

    # two-input-two-output keras Model
    return models.Model([x, y], [out_caps, x_recon])
```

Figure 26: ConvCaps block

```
# compile the model
model.compile(optimizer='adam',
              loss=[margin_loss, 'mse'],
              loss_weights=[1., 0.0005],
              metrics={'caps': 'accuracy'})
```

Figure 27: Hyper-parameter tuning

```
#time counter for model time capturing
import time
time.perf_counter()

#training model with data and created model
train_model(model=model, data=((x_train, y_train), (x_test, y_test)))
Epoch 1/100
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

246/246 [==============================] - 1239s 5s/step - loss: 1.1335 - caps_loss: 1.1334 -

Epoch 00001: val_loss improved from inf to 1.06416, saving model to weights-01.h5
Epoch 2/100
246/246 [==============================] - 1239s 5s/step - loss: 1.1318 - caps_loss: 1.1317 -

Epoch 00002: val_loss did not improve from 1.06416
```

Figure 28: ConvCaps model run

4.3 Supporting models: InceptionV3 and DenseNet

For pre-trained Transfer Learning models data were passed through image generator with real time augmentation. The data were stored in test. train and valid folders for processing Figure 29.

```
image_size = 175  #input image size


# real time augmentation fro training data
dataGenerator = ImageDataGenerator(horizontal_flip=True,
                                vertical_flip=True,
                                rotation_range=45,
                                width_shift_range=0.2,
                                height_shift_range=0.2,
                                zoom_range=.4,
                                fill_mode='nearest',
                                shear_range=0.1,
                                rescale=1/255,
                                )#validation_split=0.2

#testing data scaling
test_Data_Generator = ImageDataGenerator(rescale=1/255)

#train data load
trainGenerator = dataGenerator.flow_from_directory('/content/drive/My Drive/new_classes_6/train_classes/train/',
                                                   class_mode="categorical",
                                                   target_size=(image_size, image_size),
                                                   color_mode="rgb",
                                                   shuffle=True,
                                                   batch_size=32)

#val data load
validGenerator =  test_Data_Generator.flow_from_directory('/content/drive/My Drive/new_classes_6/train_classes/val/',
                                                          target_size=(image_size,image_size),
                                                          class_mode='categorical',
                                                          shuffle=False,
                                                          batch_size=32)

#test data load
test_gen = test_Data_Generator.flow_from_directory('/content/drive/My Drive/new_classes_6/test_class/',

                                                   target_size=(image_size,image_size),
                                                   class_mode='categorical',
                                                   shuffle=False,
                                                   batch_size=32)
```

Figure 29: Test, train and validation data fetching code

4.3.1 InceptionV3

InceptionV3 [2] has been trained on ”ImageNet” weights and same has been imported as shown in Figure 30. The input image size was given as 175 X 175. A dense layer is added in fully connected block with softmax as activation function for image clasification.

[2]https://www.tensorflow.org/api docs/python/tf/keras/

```
# importing inceptionV3 model
from tensorflow.keras.applications.inception_v3 import InceptionV3

#import ImageNet weights for inceotion model and defined input size

pre_trained_model = InceptionV3(input_shape = (175, 175, 3), # Shape of our images
                                include_top = False, # Leave out the last fully connected layer
                                weights = 'imagenet')

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/i
87916544/87910968 [==============================] - 1s 0us/step

# trainig data on pretrained weights

for layer in pre_trained_model.layers:
    layer.trainable = False

#adding flatten layer to model architecture
x = layers.Flatten()(pre_trained_model.output)

# Add a fully connected layer with 1,024 hidden units and ReLU activation
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dense(6, activation='softmax')(x)   #softmax

#final modelling
model = Model( pre_trained_model.input, x)
```

Figure 30: InceptionV3 Modelling

The model compilation included hyper-parameter shown in Figure 31 and time was calculated using time function. Model has been supplied with pre-defined steps per epoch value.

```
# hyper paraemeter tunning
adam = Adam(lr=0.0001)
model.compile(loss='categorical_crossentropy',optimizer=adam, metrics=['accuracy'])

# training model

train_step = trainGenerator.n//trainGenerator.batch_size # step per epoch for train data
val_step = validGenerator.n//validGenerator.batch_size # validation per epoch for val data

time_callback = TimeHistory()

history = model.fit_generator(trainGenerator,
                              steps_per_epoch=train_step,
                              validation_data=validGenerator,
                              validation_steps=val_step,
                              epochs=100, callbacks=[es,mc,log,time_callback])

Epoch 1/100
180/180 [==============================] - ETA: 0s - loss: 1.4983 - accuracy: 0.3951
Epoch 00001: val_accuracy improved from -inf to 0.49028, saving model to ./drive/My Drive/large_inceptionV3_1_mode
180/180 [==============================] - 94s 521ms/step - loss: 1.4983 - accuracy: 0.3951 - val_loss: 1.1435 - v
y: 0.4903
```

Figure 31: InceptionV3 compilation and epoch run

4.3.2 DenseNet

The DenseNet [3] model has higher parameter count than Inception which also get reflected in time consumption by both the models Figure 32. Also, model compilation and run is shown in Figure 33.

[3]https://www.tensorflow.org/api docs/python/tf/keras/

```
#DenseNet model import

base_model = densenet.DenseNet169(input_shape=(175, 175, 3),
                                  #weights='imagenet',
                                  weights = "imagenet",
                                  include_top=False,
                                  pooling='avg')
```

```
for layer in base_model.layers:
    layer.trainable = True

x = base_model.output

#Adding softmax layer for image classification
predictions = Dense(6, activation='softmax')(x)
```

```
# importing base model
model = Sequential()
model = Model(base_model.input, predictions)
```

```
#model summary
model.summary()
```

Figure 32: DenseNet modelling

```
#Hyper-parameter tunning

optimizer = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc', 'mse'])
```

```
train_step =  trainGenerator.n//trainGenerator.batch_size # step per epoch for train data
val_step = validGenerator.n//validGenerator.batch_size # step per epoch for validation data

time_callback = TimeHistory()
model_history = model.fit_generator(
    train,
    epochs=100,
    steps_per_epoch=train_step,
    validation_data=valid,
    validation_steps=val_step,
    callbacks=[time_callback])
```

```
Epoch 1/100
180/180 [==============================] - 100s 555ms/step - loss: 1.4452 - acc: 0.4091 - mse
Epoch 2/100
180/180 [==============================] - 58s 321ms/step - loss: 1.2317 - acc: 0.4489 - mse:
Epoch 3/100
180/180 [==============================] - 60s 332ms/step - loss: 1.2335 - acc: 0.4649 - mse:
```

Figure 33: DenseNet compilation and run

5 Model Evaluation Comparison

Model evaluation is initially checked using model accuracy and validation accuracy. For further analysis, benchmarks like F1-Score, Recall and Precision were used. Model comparison is shown in Figure 35.

5.1 Evaluation

Model evaluation is performed using classification report from sklearn library. The report consist of weight average result of recall, F1-score and precision Figure 34.

```
from sklearn.metrics import classification_report

target_names = ['1', '2', '3', '4','5','6'] #classes
print(classification_report(test_gen.classes, pred, tar

              precision    recall  f1-score   support

           1       0.88      0.65      0.75        23
           2       0.83      0.92      0.87        48
           3       0.93      0.90      0.92        30
           4       0.80      0.80      0.80        15
           5       0.78      0.97      0.86        29
           6       0.93      0.68      0.79        19

    accuracy                           0.85       164
   macro avg       0.86      0.82      0.83       164
weighted avg       0.86      0.85      0.84       164
```

Figure 34: Preision report of Inception

5.2 Model Comparison

Different model are compared in below Figure 35. From table it can be inferred that the ConvCaps and inception model were performed better than state-of-the-art model.

Method	Accuracy	Precision	Recall	F1-Score
CNN (State-of-the-art)	79%	89%	50%	64%
ConvCaps	81%	83%	80%	80%
InceptionV3	85%	86%	85%	84%
DenseNet	60%	18%	17%	17%

Figure 35: Model comparison table

References

Marcus, D. S., Wang, T. H., Parker, J., Csernansky, J. G., Morris, J. C. and Buckner, R. L. (2007). Open access series of imaging studies (oasis): Cross-sectional mri data in young, middle aged, nondemented, and demented older adults, *Journal of Cognitive Neuroscience* **19**(9): 1498–1507.

Mikolajczyk, A. and Grochowski, M. (2018). Data augmentation for improving deep learning in image classification problem, *2018 International Interdisciplinary PhD Workshop (IIPhDW)* .