Configuration Manual

MSc Research Project
Data Analytics

Gaurav Singh
Student ID: 21136921

School of Computing
National College of Ireland

Supervisor: Prof. Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Gaurav Singh
Student ID:	21136921
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Prof. Jorge Basilio
Submission Due Date:	15/12/2022
Project Title:	Human Face Analysis using Transfer Learning Approach
Word Count:	762
Page Count:	16

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:	31st January 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

Gaurav Singh
21136921

1 Introduction

This configuration manual may be used to accomplish the same goals as the original work by producing identical outcomes. It encompasses the system setup that the project was executed on, the procedures involved in exploratory data analysis, the model implementation, and the model assessments. The code samples have been appended at the conclusion of this section.

2 System and Hardware Pre-requisites Requirements

In this part, I will detail all of the tools, system prerequisites, and hardware configurations that are necessary to reproduce my work are shown in Table 1.

Table 1: System & Hardware Requirements

Environment	Kaggle
Operating System	Linux x86_64 GNU/Linux
RAM(Random Access Memory)	16390868 kB(16GB)
CPU(Processor)	Intel(R) Xeon(R) CPU @ 2.00GHz
Graphics processing unit(GPU)	GPU P100
Harddisk(Storage)	107.37 GB

2.1 Initial Requirements

In this research, I have below tools and libraries which are below

1. Microsoft office 360
2. Python 3.7.12
3. Jupyter Notebook
4. Anaconda custom (64-bit)

Tools from Microsoft Office, such as Microsoft Excel and Microsoft Word, have been used. Python was chosen as the research project's language of choice, and the whole project, including data collection, data cleaning, transformation, and analysis, was carried out in Python. Python 3.7.12 may be downloaded from the Python website at the following address: 'https://www.python.org/'. Kaggle, which includes a special embedded version of Anaconda, provides the platform for the coding competition (64-bit) Jupyter Notebook.

3 Datasets Used

In this research I have used three datasets which are as follows:

1. Dataset containing smoking and not-smoking images (smoker vs non-smoker) Link: *https://data.mendeley.com/datasets/7b52hhzs3r/1*

Figure 1: Smoker Dataset

2. VIP Attribute Dataset , Link: *http://antitza.com/VIP_attribute-dataset.html*

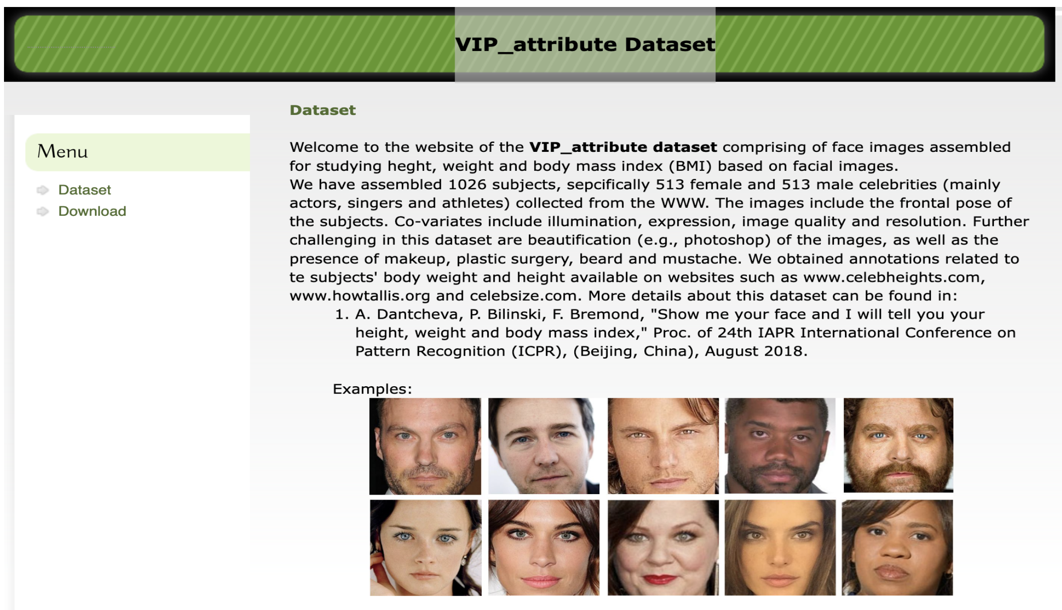

Figure 2: VIP Attribute Dataset

3. UTKFace Large Scale Face Dataset ,Link: *https://susanqq.github.io/UTKFace/*

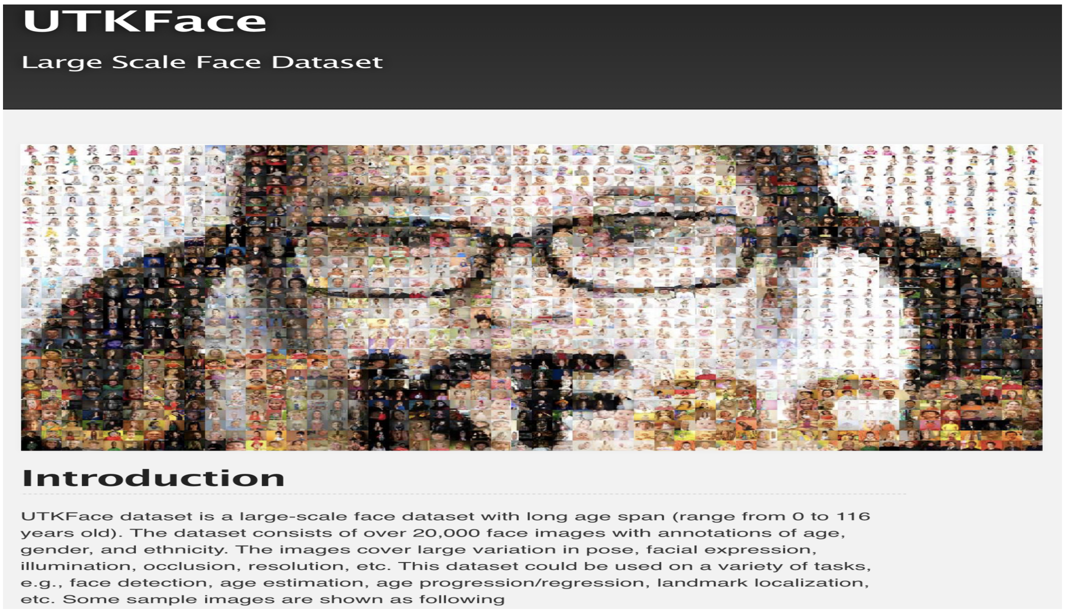

Figure 3: UTKFace Dataset

4 Research Workflow and Design

The overall workflow and the methodology followed are shown in Figure 4.

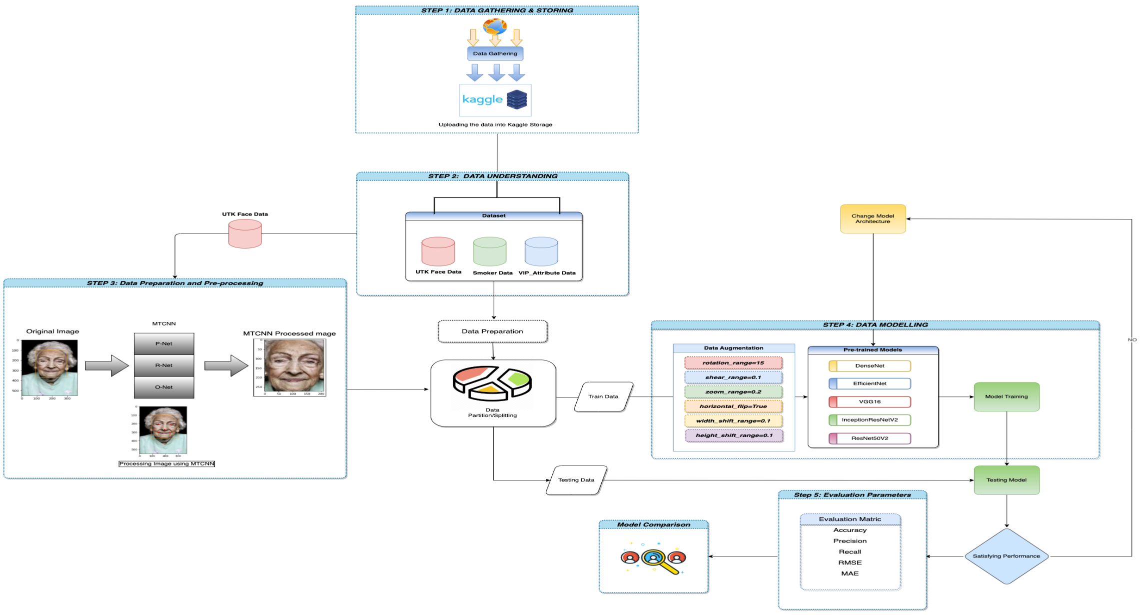

Figure 4: Workflow Diagram

5 Python packages and Libraries used

In this part, I will provide a rundown of all of the packages, Python packages, and third-party libraries (if any) utilized in the study. The fact that these packages are freely accessible, will make it easier to reuse the same work and recreate it. Table 2 shows the list of all the libraries used.

Table 2: List of Python packages and libraries used in the research

Python Library Name	Description
os	This is used in order to create folders and manage files and directories
warnings	It displays a message but runs. Warning messages are shown using the "warning" module's warn() method. Python's built-in class Exception is the warning module's superclass.
pandas	Python data analysis programming language. Its data structures and actions alter numerical tables and data series.
numpy	NumPy, a Python package, supports massive, multi-dimensional arrays and matrices and a large number of high-level mathematical functions.
cv2	This is open-source computer vision library.
tqdm	The Python module tqdm creates progress metres and bars.
matplotlib.pyplottqdm	For plotting graphs
tensorflow.keras.layers	Input, Conv2D , BatchNormalization,Activation,MaxPool2D, UpSampling2D,Concatenate,MaxPooling2D, Dropout,Flatten,Dense,GlobalAveragePooling2D
sklearn.model_selection	train_test_split
skimage.transform	used for image transformation
sklearn.metrics	classification_report, confusion_matrix
mlxtend.plotting	plot_confusion_matrix
tensorflow.keras.applications	ResNet50V2, VGG16, DenseNet201, EfficientNetB7, InceptionResNet50V2
tensorflow.keras.preprocessing.image	ImageDataGenerator
tensorflow.keras.callbacks	ModelCheckpoint,ReduceLROnPlateau
tensorflow.keras.models	Model
skimage.transform	resize
MTCNN	mtcnn

6 Data Pre-processing Code and Image Data Generators Code

Below are images of all the data processing steps. Here. Figure 5 and 6 shows the code for first converting the UTKFace dataset images into pre-processed one using MTCNN(a library that is useful in grasping the face alignment and face extraction).

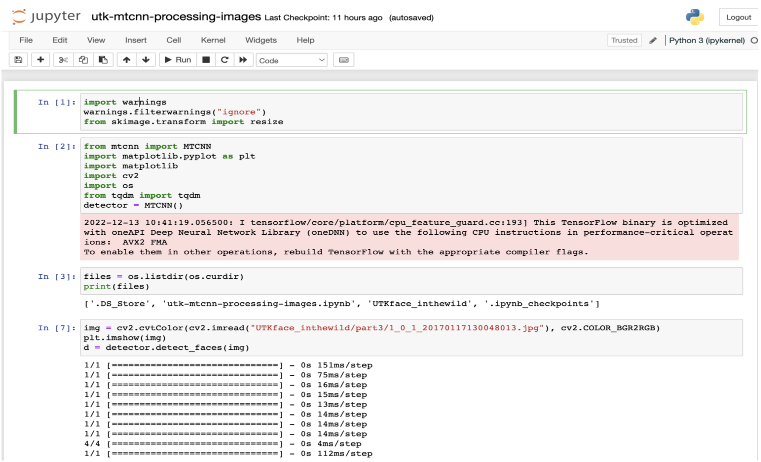

Figure 5: MTCNN UTK Face Alignment

Figure 7 shows the code which I have developed for generating the images based on the CSV data available. When performing model building image data generators are useful for such operation.

The overall exploratory analysis performed is shown in Figure 8.

Part-2

```
In [36]: path = r"UTKface_inthewild/part2"
         files = os.listdir(path)
```

```
In [37]: for f in tqdm(files):
             try:

                 img = cv2.cvtColor(cv2.imread(path+"/"+f), cv2.COLOR_BGR2RGB)
                 d = detector.detect_faces(img)
                 if(len(d)>0):
                     x, y, w, h = d[0]["box"]
                     x1 = resize(img[y:y+h, x:x+w,:], (224, 224))
                     matplotlib.image.imsave("Preprocessed_images/Part-2/"+f, x1)
             except:
                 pass
```

```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
2/2 [==============================] - 0s 4ms/step
1/1 [==============================] - 0s 20ms/step

 75%|█████████████████████        | 8015/10719 [1:42:09<35:45,  1.26it/s]

1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
2/2 [==============================] - 0s 5ms/step
1/1 [==============================] - 0s 19ms/step
```

Figure 6: MTCNN UTK Face Alignment

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# data augmentation like rotation, shearing , horizontal filp
# normalization step also included in this data generator
traindatagen= ImageDataGenerator(
    rotation_range=15,

    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1

)

train_generator = traindatagen.flow_from_dataframe(


    train,
    x_col='FileName',
    y_col='Label',
    target_size=(224,224),
    color_mode='rgb',
    class_mode='binary',
    classes=['NO SMOKE', 'SMOKE'],
    batch_size=8
)

testdatagen = ImageDataGenerator()

test_generator = testdatagen.flow_from_dataframe(
    validate,
    x_col='FileName',
    y_col='Label',
    target_size=(224,224),
    class_mode='binary',
    color_mode='rgb',
    batch_size=8,classes=['NO SMOKE', 'SMOKE'],
)
```

Figure 7: Image Data Generators Code

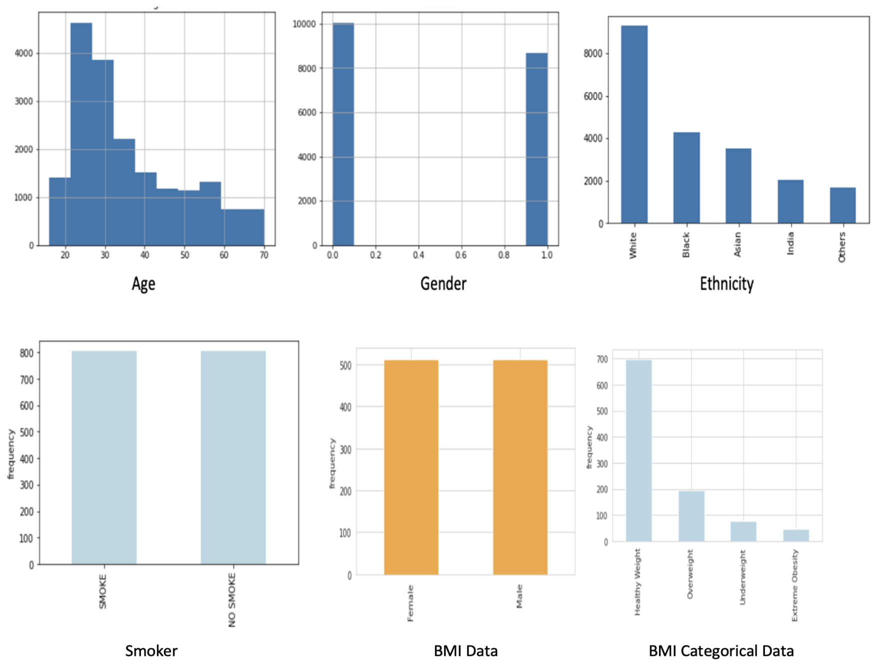

Figure 8: Exploratory Data Analysis

7 Model Implementation Code

7.1 Smoker or Non-Smoker Classification

In this section, some code snippets are attached for reference as shown in Figure 9 and 10 which shows the model building for Smoke Classification Problem using EfficientNetB7.

Model Building

```
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, MaxPool2D,
UpSampling2D, Concatenate,MaxPooling2D,Dropout,Flatten,Dense,GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
import tensorflow as tf
```

```
2022-12-07 18:38:50.433096: I tensorflow/stream_executor/platform/default/dso_loader.cc:49]
Successfully opened dynamic library libcudart.so.11.0
```

```
from tensorflow.keras.applications import EfficientNetB7

efficientNetB7 = EfficientNetB7(weights='imagenet', include_top=False)
```

Figure 9: Smoker Model Building

Figure 11 shows the Model summary for Smoker Classification problem using EfficientNetB7.

```
for x in efficientNetB7.layers:
    x.trainable = False
```

```
input_layer = Input(shape=(224, 224, 3)) # input layer with 224,224,3
model_layer = efficientNetB7(input_layer) # passing input layer to dense layer
model_layer = GlobalAveragePooling2D()(model_layer) # global average pooling
model_layer = Dense(256, activation='relu')(model_layer)# dense layer with 256 neurons and relu
activation
model_layer = Dropout(0.25)(model_layer)# drop out layer with drop out rate of 0.25 to avoid ov
erfitting
model_layer = BatchNormalization()(model_layer) # batch norm alization to speed up the training
process
model_layer = Dense(128, activation='relu')(model_layer) #dense layer with 128 neurons and relu
activation
```

```
output = Dense(1,activation = 'sigmoid')(model_layer)#dense layer with 1 neurons and sigmoid ac
tivation
```

```
model = Model(input_layer,output)
```

Figure 10: Smoker Model Building

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
efficientnetb7 (Functional)  (None, None, None, 2560)  64097687
_________________________________________________________________
global_average_pooling2d (Gl (None, 2560)              0
_________________________________________________________________
dense (Dense)                (None, 256)               655616
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 64,787,352
Trainable params: 689,153
Non-trainable params: 64,098,199
_________________________________________________________________
```

Figure 11: Smoker Model summary

7.2 BMI Identification

For the second experiment, i.e BMI Identification the model-building steps are shown in Figure 12 and 13. As seen, first various libraries were imported namely from the TensorFlow layers package, then we added input layers and various model layers.

Model Building

```
In [12]:
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, MaxPool2D,
UpSampling2D, Concatenate,MaxPooling2D,Dropout,Flatten,Dense,GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
import tensorflow as tf #importing requried tensorflow packages and modules
```

```
In [13]:
from tensorflow.keras.applications import EfficientNetB7

efficientNetB7 = EfficientNetB7(weights='imagenet', include_top=False) # densenet201 pretrained
model loading by discarding top layer that is nothing but a softmax layer
```

```
In [15]:
input_layer = Input(shape=(224, 224, 3)) # input layer with 224,224,3
model_layer = efficientNetB7(input_layer) # passing input layer to dense layer
model_layer = GlobalAveragePooling2D()(model_layer) # global average pooling
model_layer = Dense(256, activation='relu')(model_layer)# dense layer with 256 neurons and relu
activation
model_layer = Dropout(0.25)(model_layer)# drop out layer with drop out rate of 0.25 to avoid ov
erfitting
model_layer = BatchNormalization()(model_layer) # batch norm alization to speed up the training
process
model_layer = Dense(128, activation='relu')(model_layer) #dense layer with 128 neurons and relu
activation
```

```
In [16]:
output = Dense(1,activation = 'linear')(model_layer)#dense layer with 1 neurons and linear acti
vation
```

```
In [17]:
model = Model(input_layer,output)
```

Figure 12: BMI EfficientNetB7 Model Building Steps

```
model.compile(loss='mse', optimizer="adam", metrics=[tf.keras.metrics.RootMeanSquaredError(),t
f.keras.metrics.MeanAbsoluteError()])
model.summary()

Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
efficientnetb7 (Functional)  (None, None, None, 2560)  64097687
_________________________________________________________________
global_average_pooling2d (Gl (None, 2560)              0
_________________________________________________________________
dense (Dense)                (None, 256)               655616
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 64,787,352
Trainable params: 689,153
Non-trainable params: 64,098,199
_________________________________________________________________
```

Figure 13: BMI EfficientNetB7 Model summary

7.3 Gender Classification (male or female)

For the third experiment, i.e Gender Classification (male or female) the model-building steps are shown in Figure 14. As seen, first various libraries were imported namely from the TensorFlow layers package, then we added input layers and various model layers. Then, in Figure 15 shows the model summary after adding input dense layers, and also, the data frame is used with ImageDataGenerator() to produce the images.

Model Building

In [17]:
```
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, MaxPool2D,
UpSampling2D, Concatenate,MaxPooling2D,Dropout,Flatten,Dense,GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
import tensorflow as tf
```

In [18]:
```
from tensorflow.keras.applications import EfficientNetB7

pre_model = EfficientNetB7(weights='imagenet', include_top=False)
```

```
2022-12-10 11:59:40.282640: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] suc
cessful NUMA node read from SysFS had negative value (-1), but there must be at least one N
UMA node, so returning NUMA node zero
2022-12-10 11:59:40.375921: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] suc
cessful NUMA node read from SysFS had negative value (-1), but there must be at least one N
UMA node, so returning NUMA node zero
2022-12-10 11:59:40.376697: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] suc
cessful NUMA node read from SysFS had negative value (-1), but there must be at least one N
```

In [19]:
```
for x in pre_model.layers:
    x.trainable = False
```

In [20]:
```
input_layer = Input(shape=(224, 224, 3)) # input layer with 224,224,3
model_layer = pre_model(input_layer) # passing input layer to dense layer
model_layer = GlobalAveragePooling2D()(model_layer) # global average pooling
model_layer = Dense(256, activation='relu')(model_layer)# dense layer with 256 neurons and relu
activation
model_layer = Dropout(0.25)(model_layer)# drop out layer with drop out rate of 0.25 to avoid ov
erfitting
model_layer = BatchNormalization()(model_layer) # batch norm alization to speed up the training
process
model_layer = Dense(128, activation='relu')(model_layer) #dense layer with 128 neurons and relu
activation
```

In [21]:
```
output = Dense(1,activation = 'sigmoid')(model_layer)#dense layer with 1 neurons and sigmoid ac
tivation
```

In [22]:
```
model = Model(input_layer,output)
```

Figure 14: Gender Classification (Male or Female) EfficientNetB7 Model Building Steps

Model Summary

In [23]:

```
# loss function is binary cross entropy and optimizer is adam , metrics are accuracy , precision
, recall
model.compile(loss='binary_crossentropy', optimizer="adam", metrics=['accuracy',tf.keras.metri
cs.Recall(),tf.keras.metrics.Precision()])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
efficientnetb7 (Functional)  (None, None, None, 2560)  64097687
_________________________________________________________________
global_average_pooling2d (Gl (None, 2560)              0
_________________________________________________________________
dense (Dense)                (None, 256)               655616
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 64,787,352
Trainable params: 689,153
Non-trainable params: 64,098,199
_________________________________________________________________
```

```
Epoch 00099: val_loss did not improve from 0.26000
Epoch 100/100
261/261 [==============================] - 275s 1s/step - loss: 0.2231 - accuracy: 0.9057 -
recall: 0.9082 - precision: 0.9013 - val_loss: 0.2907 - val_accuracy: 0.8895 - val_recall:
0.8943 - val_precision: 0.8876

Epoch 00100: val_loss did not improve from 0.26000
```

In [35]:

```
model.save_weights('gender_efficientnetb7(w).h5')
```

In [28]:

```
train
```

Out[28]:

	FilePath	FileName	Label
3267	../input/urkfacepreprocessed/Part-1/Part-1/18_...	18_1_0_20170109212818755.jpg	Female
1216	../input/urkfacepreprocessed/Part-1/Part-1/3_1...	3_1_3_20161219230259272.jpg	Female
4597	../input/urkfacepreprocessed/Part-1/Part-1/65_...	65_1_0_20170110160643923.jpg	Female
14670	../input/urkfacepreprocessed/Part-2/Part-2/31_...	31_0_0_20170117175719891.jpg	Male
16973	../input/urkfacepreprocessed/Part-2/Part-2/43_...	43_0_3_20170112220309502.jpg	Male
...	...	...	...
18727	../input/urkfacepreprocessed/Part-2/Part-2/22_...	22_0_3_20170117154523094.jpg	Male
13589	../input/urkfacepreprocessed/Part-2/Part-2/24_...	24_1_2_20170116163702026.jpg	Female
9258	../input/urkfacepreprocessed/Part-1/Part-1/14_...	14_1_0_20170103200819591.jpg	Female
5010	../input/urkfacepreprocessed/Part-1/Part-1/5_0...	5_0_3_20161220222940507.jpg	Male
16449	../input/urkfacepreprocessed/Part-2/Part-2/35_...	35_1_0_20170116222143959.jpg	Female

16645 rows × 3 columns

Figure 15: Gender Classification (Male or Female) EfficientNetB7 Model summary

7.4 Ethnicity Multi-Classification ("White", "Black", "Asian", "India", "Others")

For the fourth experiment, i.e Ethnicity Multi-Classification the model-building steps are shown in Figure 16. As seen, first various libraries were imported namely from the TensorFlow layers package, then we added input layers and various model layers. Then, Figure 17 shows the model summary.

```
In [15]:
from sklearn.model_selection import train_test_split

df = df.sample(frac = 1)

train, test = train_test_split(df, test_size=0.20, random_state=42,shuffle=True)
train = train.sample(frac = 1)
test = test.sample(frac = 1)
```

Model Building

```
In [16]:
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, MaxPool2D,
UpSampling2D, Concatenate,MaxPooling2D,Dropout,Flatten,Dense,GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
import tensorflow as tf
```

```
In [17]:
from tensorflow.keras.applications import DenseNet201

densenet = DenseNet201(weights='imagenet', include_top=False)
```

```
In [18]:
for x in densenet.layers:
    x.trainable = False
```

```
In [19]:
input_layer = Input(shape=(224, 224, 3)) # input layer with 224,224,3
model_layer = densenet(input_layer) # passing input layer to dense layer
model_layer = GlobalAveragePooling2D()(model_layer) # global average pooling
model_layer = Dense(256, activation='relu')(model_layer)# dense layer with 256 neurons and relu
activation
model_layer = Dropout(0.25)(model_layer)# drop out layer with drop out rate of 0.25 to avoid ov
erfitting
model_layer = BatchNormalization()(model_layer) # batch norm alization to speed up the training
process
model_layer = Dense(128, activation='relu')(model_layer) #dense layer with 128 neurons and relu
activation
```

```
In [20]:
output = Dense(5,activation = 'softmax')(model_layer)#dense layer with 1 neurons and sigmoid ac
tivation
```

```
In [21]:
model = Model(input_layer,output)
```

```
In [32]:
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
# defining checkpoints for best epoch model saving and early stopping if there is no improvement
in learning
red = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=5, verbose=1, min_lr=1e-
3)
checkpoint = ModelCheckpoint('model.h5', verbose=1, save_best_only=True)
```

```
In [33]:
modelhistory = model.fit_generator(
    train_generator,
    epochs=100,
    validation_data=test_generator,
    callbacks=[red, checkpoint]
    )
```

Figure 16: Ethnicity Multi-Classification DenseNet201 Model Building Steps

Model Summary

In [22]:
```
# loss function is binary cross entropy and optimizer is adam , metrics are accuracy , precision
, recall
model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy',tf.keras.
metrics.Recall(),tf.keras.metrics.Precision()])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
densenet201 (Functional)     (None, None, None, 1920)  18321984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 256)               491776
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645
=================================================================
Total params: 18,848,325
Trainable params: 525,829
Non-trainable params: 18,322,496
_________________________________________________________________
```

Figure 17: Ethnicity Multi-Classification DenseNet201 Model summary

7.5 Finding the age of a person

For the last experiment, i.e finding the age of a person the model-building steps are shown in Figure 18. As seen, first various libraries were imported namely from the TensorFlow layers package, then we added input layers and various model layers. Then, Figure 19 shows the model summary.

Model Building

In [15]:
```
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, MaxPool2D,
UpSampling2D, Concatenate,MaxPooling2D,Dropout,Flatten,Dense,GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
import tensorflow as tf
```

In [16]:
```
from tensorflow.keras.applications import DenseNet201

densenet = DenseNet201(weights='imagenet', include_top=False)
```

In [17]:
```
for x in densenet.layers:
    x.trainable = False
```

In [18]:
```
input_layer = Input(shape=(224, 224, 3)) # input layer with 224,224,3
model_layer = densenet(input_layer) # passing input layer to dense layer
model_layer = GlobalAveragePooling2D()(model_layer) # global average pooling
model_layer = Dense(256, activation='relu')(model_layer)# dense layer with 256 neurons and relu
activation
model_layer = Dropout(0.25)(model_layer)# drop out layer with drop out rate of 0.25 to avoid ov
erfitting
model_layer = BatchNormalization()(model_layer) # batch norm alization to speed up the training
process
model_layer = Dense(128, activation='relu')(model_layer) #dense layer with 128 neurons and relu
activation
```

In [19]:
```
output = Dense(1,activation = 'linear')(model_layer)#dense layer with 1 neurons and sigmoid act
ivation
```

In [20]:
```
model = Model(input_layer,output)
```

n [30]:
```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
# defining checkpoints for best epoch model saving and early stopping if there is no improvement
in learning
red = ReduceLROnPlateau(monitor='val_root_mean_squared_error', factor=0.5, patience=5, verbose
=1, min_lr=1e-3)
checkpoint = ModelCheckpoint('model.h5', verbose=1, save_best_only=True)
```

n [31]:
```
modelhistory = model.fit_generator(
    train_generator,
    epochs=100,
    validation_data=test_generator,
    callbacks=[red, checkpoint]
    )
```

Figure 18: Age DenseNet201 Model Building Steps

Model Summary

In [21]:

```
# loss function is mean sqaured error and optimizer is adm , metrics are rmse , mae
model.compile(loss='mse', optimizer="adam", metrics=[tf.keras.metrics.RootMeanSquaredError(),t
f.keras.metrics.MeanAbsoluteError()])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
densenet201 (Functional)     (None, None, None, 1920)  18321984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 256)               491776
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 18,847,809
Trainable params: 525,313
Non-trainable params: 18,322,496
_________________________________________________________________
```

Figure 19: Age DenseNet201 Model summary

8 Model Implementation

Figure 20 shows the overall model implementation results, I have run the epochs till 100.

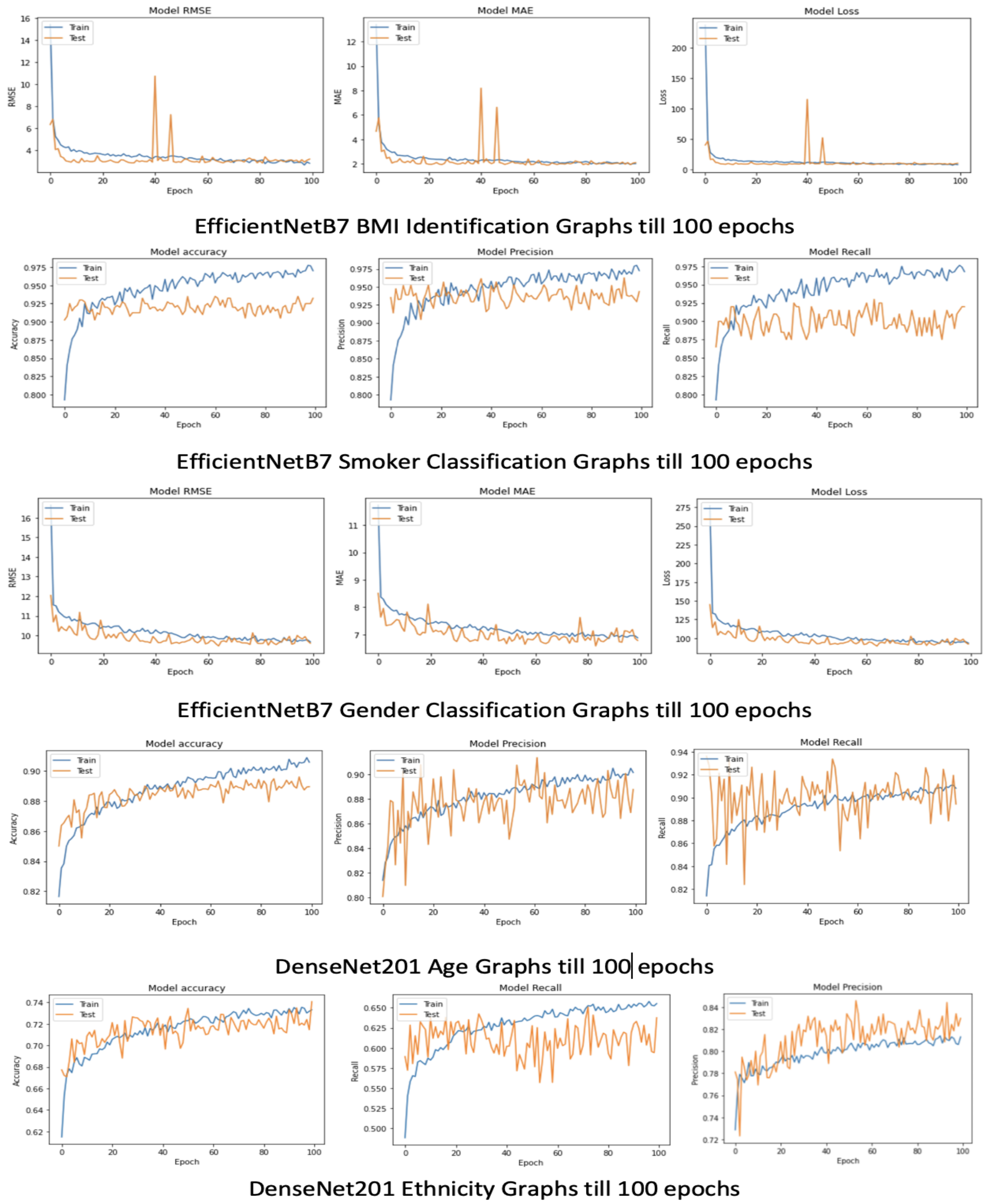

Figure 20: Model Implementation

9 Model Evaluation

Figure 21 shows the overall model evaluation results, as seen for BMI, Smoker and Gender the EfficientNetB7 model performed the best, and for age and DenseNet201 performed the best.

Table 1: BMI results

Model	RMSE		MAE	
	Test	Train	Test	Train
EfficientNetB7	3.1871	2.8063	2.0966	2.0118
DensetNet201	3.3346	3.2611	2.1761	2.2317
VGG16	3.795	3.6466	2.3436	2.3421
InceptionResNetV2	3.1795	3.6867	2.1927	2.3738
ResNet50V2	4.6423	2.9988	3.0637	2.0788

Table 2: Evaluation of Smoking Based Binary Classification Results

Model	Accuracy		Precision		Recall	
	Test	Train	Test	Train	Test	Train
EfficientNetB7	92.48%	99.75%	92.52%	99.75%	92.48%	99.75%
DensetNet201	87.72	95.02%	88.15	95.02%	87.71	95.02%
VGG16	83.46%	86.82%	83.45%	87.22%	83.49%	86.82%
InceptionResNetV2	57.89%	56.47%	62.93%	61.44%	57.97%	56.47%
ResNet50V2	89.72%	98.01%	90.10%	98.01%	89.71%	98.01%

Table 3: Evaluation of Age Results

Model	RMSE		MAE	
	Test	Train	Test	Train
EfficientNetB7	20.0096	20.351	15.4478	15.5997
DensetNet201	9.6027	9.6719	6.7815	6.8777
VGG16	12.0148	12.4385	9.0947	9.14
InceptionResNetV2	19.8154	19.1108	16.1248	14.8278
ResNet50V2	19.9665	19.1297	15.5431	14.7911

Table 4: Gender Based Binary Classification Results

Model	Accuracy		Precision		Recall	
	Test	Train	Test	Train	Test	Train
EfficientNetB7	89.02%	93.99%	89.02%	93.99%	89.02%	93.99%
DensetNet201	87.19%	89.44%	87.25%	89.49%	87.18%	89.47%
VGG16	77.15%	78.04.%	78.09%	79.22%	77.09%	78.17%
InceptionResNetV2	54.76%	55.07%	56.33%	57.49%	54.75%	55.39%
ResNet50V2	71.34%	72.96%	76.08%	77.15%	71.86%	73.08%

Table 5: Multi class Ethnicity classification Results

Model	Accuracy		Precision		Recall	
	Test	Train	Test	Train	Test	Train
EfficientNetB7	43.22%	45.01%	47.01%	48.71%	48.96%	42.97%
DensetNet201	74.05%	77.21%	65.09%	73.71%	58.96%	62.97%
VGG16	49.21%	64.59%	22.37%	75.41	83.57%	52.64%
InceptionResNetV2	47.62%	48.31%	58.20%	58.56%	49.60%	48.90%
ResNet50V2	51.66%	56.78%	61.57%	67.69%	32.34%	40.70%

Figure 21: Model Evaluation Results