Configuration Manual

MSc Research Project
Data Analytics

Soumya Nayak
Student ID: x21121427

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Soumya Nayak……………………………………………………

Student ID: X21121427……………………………………………………………………………..……

Programme: Data Analytics……………………………… **Year:** 2022……………..

Module: MSc Research Project………………………………………………………….………

Lecturer: Vladimir Milosavljevic…………………………………………………………….………

Submission Due Date: 01/02/2023…………………………………………………………………………….………

Project Title: Leveraging Transfer Learning Techniques for Homophobia and Transphobia Detection

Word Count: 823…………………………… **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: Soumya Nayak…………………….………………………………………….

Date: 01/02/2023……………………………………………………………………….

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

Soumya Nayak
X21121427

1 Introduction

This document provides complete information about the software and hardware configuration and components required for the implementation of research project for the Classification of Melanoma Skin Cancer from Melanocyte cell images using Transfer Learning Techniques. The steps mentioned in this configuration manual can be considered to run the code and obtain the desired results.

2 Hardware and Software Configuration

The Technical specifications of device used and windows operating system on which implementation of this research work has been carried out is shown in Figure 1 and 2

Device specifications

Device name	DESKTOP-FEJ5I7B
Processor	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz
Installed RAM	8.00 GB (7.88 GB usable)
Device ID	C62A3449-8F66-4791-A5DD-604A8B6E2494
Product ID	00330-80000-00000-AA915
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Device Specifications

Windows specifications

Edition	Windows 10 Pro
Version	21H2
Installed on	15-01-2022
OS build	19044.1526
Experience	Windows Feature Experience Pack 120.2212.4170.0

Figure 2: Windows Specifications

Python has been opted as the programming language and Python 3.7 is used for the research implementation. The setup configuration is shown in Figure 3.

IDE	Google Colab Pro,Jupyter Notebook, Kaggle Notebook
Programming language	Python
Framework	Tensorflow
Modules	Matplotlib, Pandas, Numpy Seaborn, Scikit-learn, cv2, imblearn
Computation	GPU
Number of GPU	1
Type	Tesla P100-PCIE-16GB

Figure 3: Setup Configuration

3 Dataset

The dataset selected to carry out this thesis is downloaded from kaggle from ISIC (International Skin Imaging Collaboration) 2019 Skin Lesion images for classification containing 8 classes which is publicly available dataset. The link for the dataset is https://www.kaggle.com/datasets/salviohexia/isic-2019-skin-lesion-images-for-classification. The data is uploaded to Google drive and Kaggle account for implementing the models. Moreover, it was downloaded on local system to carry out Exploratory Data Analysis on raw data.

4 Implementation on Models

4.1 MobileNetV2 Model

All the modules and libraries required for the successful execution of MobileNetV2 model are imported as shown in Figure 4.

```
#importing all the required packages
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import pathlib
```

Figure 4: Importing required modules

Reading root path directly from Kaggle as shown in Figure 5 to fetch the dataset since size of dataset is 10GB which can be inconvenient to download in local system due to requirement of more storage.

```
#Reading root path directly from Kaggle to fetch the dataset
dat_dir = "/kaggle/input/isic-2019-skin-lesion-images-for-classification"#Dataset path
dat_dir = pathlib.Path(dat_dir) #converting data directory into pathlib
dat_dir
```

Figure 5: Reading Dataset directly from kaggle

Figure 6 and 7 shows the code to create dictionary for each class and labelling all the eight classes of dataset.

```
#Creating a dictionary for each class with key-value pair
#key represents path, value represents path
image_dict = {'AK' : list(dat_dir.glob('AK/*.jpg')),
'BCC' : list(dat_dir.glob('BCC/*.jpg')),
'BKL' : list(dat_dir.glob('BKL/*.jpg')),
'DF' : list(dat_dir.glob('DF/*.jpg')),
'MEL' : list(dat_dir.glob('MEL/*.jpg')),
'NV' : list(dat_dir.glob('NV/*.jpg')),
'SCC' : list(dat_dir.glob('SCC/*.jpg')),
'VASC' : list(dat_dir.glob('VASC/*.jpg'))}
```

Figure 6: Create dictionary for each class

```
#Labeling each class
class_dict = {'AK' : 0,
'BCC' : 1,
'BKL' :2,
'DF' : 3,
'MEL' :4,
'NV' : 5,
'SCC' :6,
'VASC' : 7}
```

Figure 7: Labeling all the classes

The data pre-processing step is shown in Figure 8 where majority classes are downsampled to 2500.

```
#Downsampling all the majority classes to 2500
df_ak = df[df['class'] == 0]

df_bcc = df[df['class'] == 1]
df_bcc_shuffled = df_bcc.sample(frac=1,random_state = 42)
df_bcc_shuffled_subset=df_bcc_shuffled.iloc[:2500]

df_bkl = df[df['class'] == 2]
df_bkl_shuffled = df_bkl.sample(frac=1,random_state = 42)
df_bkl_shuffled_subset=df_bkl_shuffled.iloc[:2500]

df_df = df[df['class'] == 3]

df_mel = df[df['class'] == 4]
df_mel_shuffled = df_mel.sample(frac=1,random_state = 42)
df_mel_shuffled_subset=df_mel_shuffled.iloc[:2500]

df_nv = df[df['class'] == 5]
df_nv_shuffled = df_nv.sample(frac=1,random_state = 42)
df_nv_shuffled_subset=df_nv_shuffled.iloc[:2500]

df_scc = df[df['class'] == 6]

df_vasc = df[df['class'] == 7]
```

Figure 8: Downsampling majority class

Figure 9 and 10 shows combining the dataframes and shuffling it to make the dataset random

```
#Combining all the dataframes in a single dataframe
frames = [df_ak,df_bcc_shuffled_subset, df_bkl_shuffled_subset,df_df,
          df_mel_shuffled_subset,df_nv_shuffled_subset,df_scc,df_vasc]
df_aug = pd.concat(frames)
```

Figure 9: Combining dataframes

```
#sample function to shuffle the dataframe after downsampling
df_aug = df_aug.sample(frac=1,random_state = 42).reset_index(drop=True)
```

Figure 9: Shuffling the dataframe

Figure 10 shows the splitting of dataset into test and training datasets.

```
#Splitting the data into test and train sets
from sklearn.model_selection import train_test_split
train, test=train_test_split(df_aug, test_size=0.2, random_state=42)
```

Figure 10: Splitting the data into test and train sets

RandomOverSampler is used to balance the imbalanced data. Figure 11 shows the upsampling of minority classes in order to achieve balance dataset for training purpose.

```
#RandomOverSampler to handle imbalanced data
#Upsampling minority classes to balance the data
from sklearn.datasets import make_classification
from imblearn.over_sampling import RandomOverSampler
os =  RandomOverSampler(sampling_strategy= 'not majority', random_state=42)
```

Figure 11: Upsampling minority classes

Data Transformation is done to convert images into MobileNetV2 compatible input format as shown in Figure 12

```
#Creating the image dataset using Tensorflow
#Converting images into mobilenet compatible input format using ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.resnet import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator

trainGen = ImageDataGenerator(preprocessing_function=preprocess_input,
                              validation_split=0.3, rescale = 0.0039)#,horizontal_flip=
testGen =ImageDataGenerator(preprocessing_function= preprocess_input, rescale = 0.0039)
X_train_img = trainGen.flow_from_dataframe(dataframe=df_sampled_train,
                                           x_col='image', y_col='class',
                                           class_mode='sparse', subset='training',
                                           color_mode='rgb', batch_size=32)
X_val_img = trainGen.flow_from_dataframe(dataframe=df_sampled_train, x_col='image',
                                         y_col='class',class_mode='sparse',
                                         subset='validation', color_mode='rgb',
                                         batch_size=32)
X_test_img =testGen.flow_from_dataframe(dataframe=test, x_col='image', y_col='class',
                                        class_mode='sparse', color_mode='rgb',
                                        batch_size=32, shuffle=False)
```

Figure 12: Data Transformation

Figure 13 shows the model building steps of MobileNetV2. To fine tune the model fews extra hidden layers have been added in pre-trained MobileNetV2 transfer learning model with relu and softmax as activation functions. Moreover, 50% of active neurons are considered in dropout layer and model was trained for 30 epochs using Adam's optimizer.

```
#Model building
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Activation, Dropout, Flatten,
Dense, Conv2D, MaxPooling2D
pre_trained= MobileNetV2(include_top=False, pooling='avg', input_shape=image_shape)

#for layers in pre_trained.layers:
#    layers.trainable=False
pre_trained.trainable=False

inp_model = pre_trained.input
#Extra layers added to fin tune MobileNetV2 model
x=Dense(128, activation='relu')(pre_trained.output)
x=Dropout(0.5)(x)
x=Dense(128, activation='relu')(x)
output=Dense(8, activation='softmax')(x)
model = Model(inputs=inp_model, outputs=output)

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              optimizer='adam',
              metrics=['accuracy'])


results = model.fit(X_train_img,epochs=30,
                              validation_data=X_val_img)
```

Figure 12: Model Building

Figure 13 and 14 shows the plotting of learning curves after successful execution of training model.

```
#Accuracy graph for train vs validation data
result=pd.DataFrame(results.history)
plt.plot(result['accuracy'], label='train data')
plt.plot(result['val_accuracy'], label='validation data')
plt.xlabel('Epoch')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```

Figure 13: Accuracy graph for training and validation data

```
#Loss graph for train vs validation data
result=pd.DataFrame(results.history)
plt.plot(result['loss'], label='train data')
plt.plot(result['val_loss'], label='validation data')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Figure 13: Loss graph for training and validation data

Figure 14 shows the code to print confusion matrix and test accuracy of final model.

```
#Confusion matrix
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
print(f"Accuracy Score: {accuracy_score(pred_df['class'],pred_df['pred'])}")
plt.figure(figsize = (15,8))
sns.heatmap(confusion_matrix(pred_df['class'],pred_df['pred']), annot=True, fmt='2d')
```

Figure 14: Confusion matrix and overall test accuracy

Figure 15 shows the classification report which is used to identify how well the model is able to classify melanoma skin cancer and which class produces significant results.

```
#Printing classification report
from sklearn.metrics import classification_report
report = classification_report(pred_df['class'],pred_df['pred'])
print(report)
```

Figure 14: Classification report

4.2 DenseNet201 and InceptionV3 Model

The data loading, data pre-processing and data transformation steps for DenseNet201model and InceptionV3 model is almost similar. Only the model building steps in both the models are different. These steps of the whole process is mentioned below.

The data is first loaded in Google Drive and its mounting is done with Google Colab pro for the smooth execution. Figure 15 shows the process of mounting.

```
#Mounting google drive to load the dataset
from google.colab import drive
drive.mount('/content/drive')
```

Figure 15. Mounting google drive

All the modules and libraries required for the successful execution of DenseNet201 and InceptionV3 models are imported as shown in Figure 16

```python
#importing all the required packages
import pandas as pd
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import time
import matplotlib.pyplot as plt
import cv2
import seaborn as sns
sns.set_style('darkgrid')
import shutil
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Activation,Dropout,Conv2D, MaxPooling2D,BatchNormalization
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
import time
from tqdm import tqdm
from sklearn.metrics import f1_score
import sys
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
```

Figure 16: Importing required packages

Figure 17 and 18 shows the code for reading image paths and labels and creating test, train and validation dataframes.

```python
#Read the paths and labels of images in the dataset
def make_dataframes(sdir):
    filepaths=[]
    labels=[]
    classlist=sorted(os.listdir(sdir) )
    for klass in classlist:
        classpath=os.path.join(sdir, klass)
        if os.path.isdir(classpath):
            flist=sorted(os.listdir(classpath))
            desc=f'{klass:25s}'
            for f in tqdm(flist, ncols=130,desc=desc, unit='files', colour='blue'):
                fpath=os.path.join(classpath,f)
                filepaths.append(fpath)
                labels.append(klass)
    Fseries=pd.Series(filepaths, name='filepaths')
    Lseries=pd.Series(labels, name='labels')
    df=pd.concat([Fseries, Lseries], axis=1)
    #spilt the dataset into train, test and validation
    train_df, dummy_df=train_test_split(df, train_size=.95, shuffle=True, random_state=123, stratify=df['labels'])
    valid_df, test_df=train_test_split(dummy_df, train_size=.5, shuffle=True, random_state=123, stratify=dummy_df['labels'])
    classes=sorted(train_df['labels'].unique())
    class_count=len(classes)
    sample_df=train_df.sample(n=50, replace=False)
```

Figure 17: Reading the path splitting the dataset

```
    # calculate the average image height and width
    ht=0
    wt=0
    count=0
    for i in range(len(sample_df)):
        fpath=sample_df['filepaths'].iloc[i]
        try:
            img=cv2.imread(fpath)
            h=img.shape[0]
            w=img.shape[1]
            wt +=w
            ht +=h
            count +=1
        except:
            pass
    have=int(ht/count)
    wave=int(wt/count)
    aspect_ratio=have/wave
    print('number of classes in processed dataset= ', class_count)
    counts=list(train_df['labels'].value_counts())
    print('the maximum files in any class in train_df is ', max(counts), '  the minimum files in any class in train_df is ', min(counts))
    print('train_df length: ', len(train_df), '  test_df length: ', len(test_df), '  valid_df length: ', len(valid_df))
    print('average image height= ', have, '  average image width= ', wave, ' aspect ratio h/w= ', aspect_ratio)
    return train_df, test_df, valid_df, classes, class_count

sdir=r'/content/drive/MyDrive/data'
train_df, test_df, valid_df, classes, class_count=make_dataframes(sdir)
```

Figure 18: Calculating average height and width of image

Figure 19 shows the code to trim the dataframe to maximum class size that is 800 which is considered here.

```
# Set maximum sample size in any class to 800 and define a function to trim classes which contains
#more than 800 samples without modifying other classes
def trim(df, max_samples, min_samples, column):
    df=df.copy()
    classes=df[column].unique()
    class_count=len(classes)
    length=len(df)
    print ('dataframe initially is of length ',length, ' with ', class_count, ' classes')
    groups=df.groupby(column)
    trimmed_df = pd.DataFrame(columns = df.columns)
    groups=df.groupby(column)
    for label in df[column].unique():
        group=groups.get_group(label)
        count=len(group)
        if count > max_samples:
            sampled_group=group.sample(n=max_samples, random_state=123,axis=0)
            trimmed_df=pd.concat([trimmed_df, sampled_group], axis=0)
        else:
            if count>=min_samples:
                sampled_group=group
                trimmed_df=pd.concat([trimmed_df, sampled_group], axis=0)
    print('After trimming, the maximum samples in any class is now ',max_samples,
          ' and the minimum samples in any class is ', min_samples)
    classes=trimmed_df[column].unique()# return this in case some classes have less than min_samples
    class_count=len(classes) # return this in case some classes have less than min_samples
    length=len(trimmed_df)
    print ('The trimmed dataframe now is of length ',length, ' with ', class_count, ' classes')
    return trimmed_df, classes, class_count
```

Figure 19: Trimming dataframe to maximum size of 800

Figure 20 and 21 shows the code for data transformation by using different data augmentation techniques.

```
# Perform data augmentation to mitigate class imbalance so that each class will have 800 images
def balance(df, n, working_dir, img_size):
    df=df.copy()
    print('Initial length of dataframe is ', len(df))
    aug_dir=os.path.join(working_dir, 'aug')# directory to store augmented images
    if os.path.isdir(aug_dir):# start with an empty directory
        shutil.rmtree(aug_dir)
    os.mkdir(aug_dir)
    for label in df['labels'].unique():
        dir_path=os.path.join(aug_dir,label)
        os.mkdir(dir_path) # make class directories within aug directory
```

Figure 20

```
# create and store the augmented images
total=0
gen=ImageDataGenerator(horizontal_flip=True,  rotation_range=20, width_shift_range=.2,
                              height_shift_range=.2, zoom_range=.2)
groups=df.groupby('labels') # group by class
for label in df['labels'].unique():  # for every class
    group=groups.get_group(label)  # a dataframe holding only rows with the specified label
    sample_count=len(group)   # determine how many samples there are in this class
    if sample_count< n: # if the class has less than target number of images
        aug_img_count=0
        delta=n - sample_count  # number of augmented images to create
        target_dir=os.path.join(aug_dir, label)  # define where to write the images
        msg='{0:40s} for class {1:^30s} creating {2:^5s} augmented images'.format(' ', label, str(delta))
        print(msg, '\r', end='') # prints over on the same line
        aug_gen=gen.flow_from_dataframe( group,  x_col='filepaths', y_col=None, target_size=img_size,
                                        class_mode=None, batch_size=1, shuffle=False,
                                        save_to_dir=target_dir, save_prefix='aug-', color_mode='rgb',
                                        save_format='jpg')
        while aug_img_count<delta:
            images=next(aug_gen)
            aug_img_count += len(images)
        total +=aug_img_count
print('Total Augmented images created= ', total)
```

Figure 21: ImageDataGenerator to perform data augmentation and storing the augmented images

Figure 22 shows the code to create augmented dataframe and merging it with training dataframe a create a single dataframe including all the original and augmented images to train the final model.

```python
    # create aug_df and merge with train_df to create composite training set ndf
    aug_fpaths=[]
    aug_labels=[]
    classlist=os.listdir(aug_dir)
    for klass in classlist:
        classpath=os.path.join(aug_dir, klass)
        flist=os.listdir(classpath)
        for f in flist:
            fpath=os.path.join(classpath,f)
            aug_fpaths.append(fpath)
            aug_labels.append(klass)
    Fseries=pd.Series(aug_fpaths, name='filepaths')
    Lseries=pd.Series(aug_labels, name='labels')
    aug_df=pd.concat([Fseries, Lseries], axis=1)
    df=pd.concat([df,aug_df], axis=0).reset_index(drop=True)
    print('Length of augmented dataframe is now ', len(df))
    return df
img_size=(250,300)
working_dir=r'./'
n=800
train_df=balance(train_df, n, working_dir, img_size)
```

Figure 22: Creating a composite dataframe

Figure 23 shows the code to convert training, validation and test data into DenseNet201 and InceptionV3 compatible input format.

```python
def make_gens(batch_size, train_df, test_df, valid_df, img_size):
    trgen=ImageDataGenerator()
    t_and_v_gen=ImageDataGenerator()
    msg='{0:70s} for train generator'.format(' ')
    print(msg, '\r', end='') # prints over on the same line
    train_gen=trgen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                        class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=batch_size)
    msg='{0:70s} for valid generator'.format(' ')
    print(msg, '\r', end='') # prints over on the same line
    valid_gen=t_and_v_gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                        class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=batch_size)
    # for the test_gen we want to calculate the batch size and test steps such that batch_size X test_steps= number of samples in test set
    # this insures that we go through all the sample in the test set exactly once.
    length=len(test_df)
    test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
    test_steps=int(length/test_batch_size)
    msg='{0:70s} for test generator'.format(' ')
    print(msg, '\r', end='') # prints over on the same line
    test_gen=t_and_v_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                        class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=test_batch_size)
    # from the generator we can get information we will need later
    classes=list(train_gen.class_indices.keys())
    class_indices=list(train_gen.class_indices.values())
    class_count=len(classes)
    labels=test_gen.labels
    print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps, ' number of classes : ', class_count)
    return train_gen, test_gen, valid_gen, test_batch_size, test_steps, classes


batch_size=20
train_gen, test_gen, valid_gen, test_batch_size, test_steps, classes=make_gens(batch_size, train_df, test_df, valid_df, img_size)
```

Figure 23: Data Transformation into DenseNet201 and InceptionV3 compatible input format

Figure 24 shows the code to display sample images from training data

```
#Display sample images from train data
def show_image_samples(gen ):
    t_dict=gen.class_indices
    classes=list(t_dict.keys())
    images,labels=next(gen) # get a sample batch from the generator
    plt.figure(figsize=(25, 25))
    length=len(labels)
    if length<25:   #show maximum of 25 images
        r=length
    else:
        r=25
    for i in range(r):
        plt.subplot(5, 5, i + 1)
        image=images[i] /255
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color='blue', fontsize=18)
        plt.axis('off')
    plt.show()

show_image_samples(train_gen )
```

Figure 24: Displaying sample images

Figure 25 and 26 shows the code for building DenseNet201 and InceptionV3 model respectively.

```
# Initialize the Densenet model and define hyperparameters
def make_model(img_size, lr, mod_num=0):
    img_shape=(img_size[0], img_size[1], 3)
    base_model=tf.keras.applications.densenet.DenseNet201(include_top=False, weights="imagenet",input_shape=img_shape, pooling='max')
    msg='Created DenseNet201 model'
    base_model.trainable=True
    x=base_model.output
    x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
    x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                    bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
    x=Dropout(rate=.4, seed=123)(x)
    output=Dense(class_count, activation='softmax')(x)
    model=Model(inputs=base_model.input, outputs=output)
    model.compile(Adamax(learning_rate=lr), loss='categorical_crossentropy', metrics=['accuracy'])
    msg=msg + f' with initial learning rate set to {lr}'
    print(msg)
    return model

lr=.001
model=make_model(img_size, lr, 0) # using B4 model
```

Figure 25: Model Training of DenseNet201

```python
# Initialize the Densenet model and define hyperparameters
def make_model(img_size, lr, mod_num=0):
    img_shape=(img_size[0], img_size[1], 3)
    base_model=tf.keras.applications.densenet.DenseNet201(include_top=False, weights="imagenet",input_shape=img_shape, pooling='max')
    msg='Created DenseNet201 model'
    base_model.trainable=True
    x=base_model.output
    x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
    x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                    bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
    x=Dropout(rate=.4, seed=123)(x)
    output=Dense(class_count, activation='softmax')(x)
    model=Model(inputs=base_model.input, outputs=output)
    model.compile(Adamax(learning_rate=lr), loss='categorical_crossentropy', metrics=['accuracy'])
    msg=msg + f' with initial learning rate set to {lr}'
    print(msg)
    return model

lr=.001
model=make_model(img_size, lr, 0) # using B4 model
```
Figure 26: Model Training of InceptionV3

A custom Keras callback mechanism is created using the code shown in Figure 27. This function allows us to continue or stop the training when specific number of epochs are reached.

```python
class LR_ASK(keras.callbacks.Callback):
    def __init__ (self, model, epochs,  ask_epoch, dwell=True, factor=.4): # initialization of the callback
        super(LR_ASK, self).__init__()
        self.model=model
        self.ask_epoch=ask_epoch
        self.epochs=epochs
        self.ask=True # if True query the user on a specified epoch
        self.lowest_vloss=np.inf
        self.lowest_aloss=np.inf
        self.best_weights=self.model.get_weights() # set best weights to model's initial weights
        self.best_epoch=1
        self.plist=[]
        self.alist=[]
        self.dwell= dwell
        self.factor=factor
```
Figure 27: Keras custom callback mechanism

Figure 28 and 29 shows the code for final model training and at the same time initializing callback mechanism

```python
#Initialize the callback
epochs=20
ask_epoch=10
ask=LR_ASK(model, epochs,  ask_epoch)
callbacks=[ask]
```
Figure 28: Initializing callback

```
#Train the model on train dataset. Validation is done using validation data
history=model.fit(x=train_gen,  epochs=epochs, verbose=1, callbacks=callbacks,  validation_data=valid_gen,
               validation_steps=None,  shuffle=False,  initial_epoch=0)
```

Figure 29: Model Training

Figure 30 and 31 shows the code to plot learning curves that is loss and accuracy curves for training and validation data.

```
#Display the Loss and Accuracy plot for train vs validation data for each epoch
def tr_plot(tr_data, start_epoch):
    #Plot the training and validation data
    tacc=tr_data.history['accuracy']
    tloss=tr_data.history['loss']
    vacc=tr_data.history['val_accuracy']
    vloss=tr_data.history['val_loss']
    Epoch_count=len(tacc)+ start_epoch
    Epochs=[]
    for i in range (start_epoch ,Epoch_count):
        Epochs.append(i+1)
    index_loss=np.argmin(vloss)#  this is the epoch with the lowest validation loss
    val_lowest=vloss[index_loss]
    index_acc=np.argmax(vacc)
    acc_highest=vacc[index_acc]
    plt.style.use('fivethirtyeight')
    sc_label='best epoch= '+ str(index_loss+1 +start_epoch)
    vc_label='best epoch= '+ str(index_acc + 1+ start_epoch)
    fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(13,5))
    axes[0].plot(Epochs,tloss, 'r', label='Training loss')
    axes[0].plot(Epochs,vloss,'g',label='Validation loss' )
    axes[0].scatter(index_loss+1 +start_epoch,val_lowest, s=150, c= 'blue', label=sc_label)
    axes[0].scatter(Epochs, tloss, s=100, c='red')
    axes[0].set_title('Training and Validation Loss')
    axes[0].set_xlabel('Epochs', fontsize=18)
    axes[0].set_ylabel('Loss', fontsize=18)
    axes[0].legend()
```

Figure 30: Plotting learning curves

```
    axes[1].plot (Epochs,tacc,'r',label= 'Training Accuracy')
    axes[1].scatter(Epochs, tacc, s=100, c='red')
    axes[1].plot (Epochs,vacc,'g',label= 'Validation Accuracy')
    axes[1].scatter(index_acc+1 +start_epoch,acc_highest, s=150, c= 'blue', label=vc_label)
    axes[1].set_title('Training and Validation Accuracy')
    axes[1].set_xlabel('Epochs', fontsize=18)
    axes[1].set_ylabel('Accuracy', fontsize=18)
    axes[1].legend()
    plt.tight_layout
    plt.show()
    return index_loss

loss_index=tr_plot(history,0)
```

Figure 31

Figure 32 and 33 shows the code for model evaluation and predictions by displaying confusion matrix and classification report.

```
#Generate predictions on test data.
#Display the confusion matrix and classification report for the predictions on test data.
def predictor(test_gen):
    y_pred= []
    error_list=[]
    error_pred_list = []
    y_true=test_gen.labels
    classes=list(test_gen.class_indices.keys())
    class_count=len(classes)
    errors=0
    preds=model.predict(test_gen, verbose=1)
    tests=len(preds)
    for i, p in enumerate(preds):
        file=test_gen.filenames[i]
        pred_index=np.argmax(p)
        true_index=test_gen.labels[i]  # labels are integer values
        if pred_index != true_index: # a misclassification has occurred
            errors=errors + 1
            file=test_gen.filenames[i]
            error_class=classes[pred_index]
            t=(file, error_class)
            error_list.append(t)
        y_pred.append(pred_index)
```

Figure 32: Generating predictions on test data

```
    acc=( 1-errors/tests) * 100
    msg=f'there were {errors} errors in {tests} tests for an accuracy of {acc:6.2f}'
    print(msg)
    ypred=np.array(y_pred)
    ytrue=np.array(y_true)
    f1score=f1_score(ytrue, ypred, average='weighted')* 100
    if class_count <=30:
        cm = confusion_matrix(ytrue, ypred )
        # plot the confusion matrix
        plt.figure(figsize=(12, 8))
        sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False)
        plt.xticks(np.arange(class_count)+.5, classes, rotation=90)
        plt.yticks(np.arange(class_count)+.5, classes, rotation=0)
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title("Confusion Matrix")
        plt.show()
    clr = classification_report(y_true, y_pred, target_names=classes, digits= 4) # create classification report
    print("Classification Report:\n----------------------\n", clr)
    return errors, tests, error_list, f1score

errors, tests, error_list, f1score =predictor(test_gen)
```

Figure 33: Calculating test accuracy

5 Results

5.1 MobileNetV2

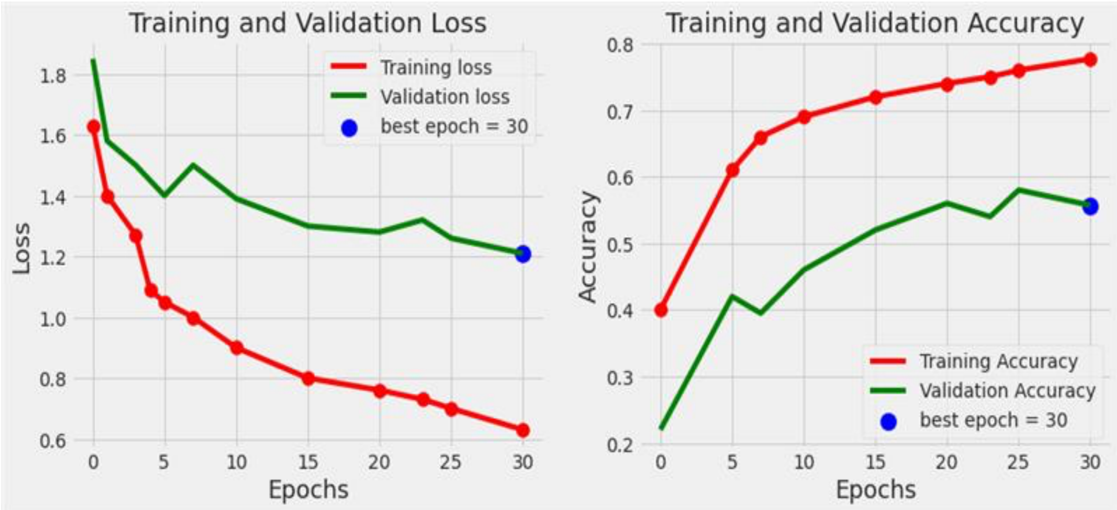

Figure 34: Learning Curves

	precision	recall	f1-score	support
0	0.29	0.57	0.38	171
1	0.58	0.50	0.54	497
2	0.48	0.47	0.47	530
3	0.22	0.33	0.26	43
4	0.56	0.43	0.49	472
5	0.68	0.59	0.63	530
6	0.28	0.44	0.34	111
7	0.45	0.66	0.54	44
accuracy			0.50	2398
macro avg	0.44	0.50	0.46	2398
weighted avg	0.53	0.50	0.51	2398

Figure 35: Classification Report

5.2 DenseNet201

Figure 36: Learning Curves

Classification Report:

	precision	recall	f1-score	support
AK	0.3696	0.7727	0.5000	22
BCC	0.7234	0.8193	0.7684	83
BKL	0.5636	0.4697	0.5124	66
DF	0.3333	0.5000	0.4000	6
MEL	0.4690	0.6018	0.5271	113
NV	0.8893	0.7236	0.7979	322
SCC	0.5000	0.4375	0.4667	16
VASC	0.6667	1.0000	0.8000	6
accuracy			0.6830	634
macro avg	0.5644	0.6656	0.5966	634
weighted avg	0.7235	0.6830	0.6936	634

Figure 37: Classification Report

5.3 InceptionV3

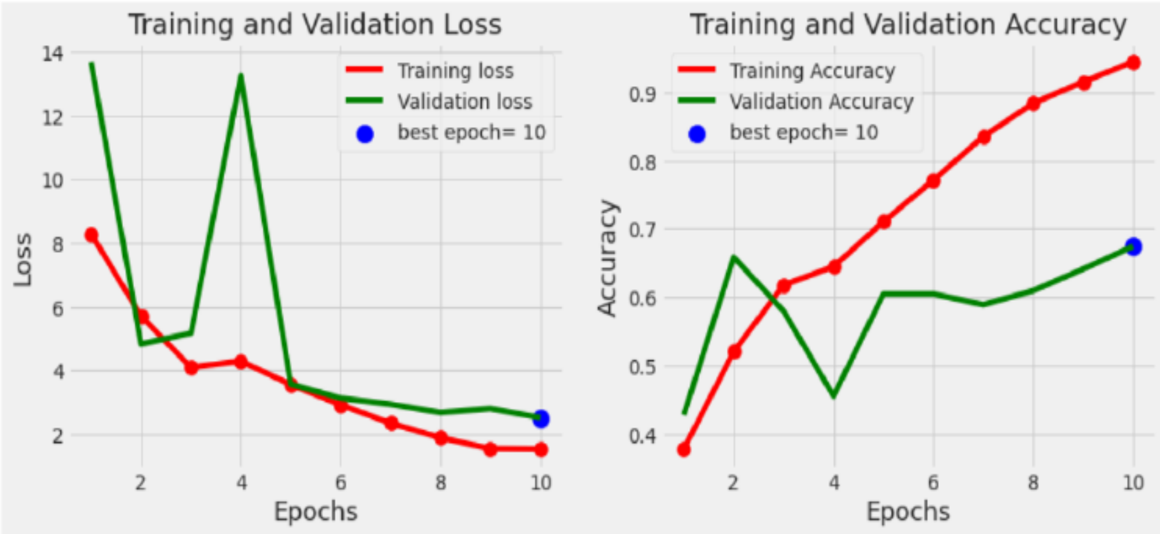

Figure 38: Learning Curves

Classification Report:

----

	precision	recall	f1-score	support
AK	0.2759	0.7273	0.4000	22
BCC	0.7206	0.5904	0.6490	83
BKL	0.3838	0.5758	0.4606	66
DF	0.0000	0.0000	0.0000	6
MEL	0.5700	0.5044	0.5352	113
NV	0.8529	0.7205	0.7811	322
SCC	0.4211	0.5000	0.4571	16
VASC	0.4000	1.0000	0.5714	6
accuracy			0.6404	634
macro avg	0.4530	0.5773	0.4818	634
weighted avg	0.6931	0.6404	0.6559	634

Figure 39: Classification Report