National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Suvrojitmoy Ghosh
Student ID: x19179090

School of Computing
National College of Ireland

Supervisor: DR. Rashmi Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Suvrojitmoy Ghosh		
Student ID:	X19179090		
Programme:	MSC Data Analytics	**Year:**	2019-2020
Module:	MSC Research Project		
Lecturer:	DR. Rashmi Gupta		
Submission Due Date:	28/9/2020		
Project Title:	Configuration Manual		
Word Count:	2855	**Page Count:**	22

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: 17/08/2020

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

Suvrojitmoy Ghosh
Student ID: x19179090

1 Pre-Requisites

To successfully implement this project you need to have a valid Google id in order to access Google colab either in windows or in IOS. It is a Jupyter notebook environment that requires no setup and runs entirely on cloud. It uses two versions of python (2.7 and 3.6.7). Here we will be using python 3.6.7[1]

2 System Specification

The data was processed using GPU (Graphic process unit) which is provided in Google colab, and the GPU specification in colab is as follows:

- n1-highmem-2-instance
- 2vCPU @ 2.2GHz
- 13GB RAM
- 100 GB free space
- Idle cut-off 90 minutes
- Maximum 12 hours

3 Access Google Colab

Firstly we need to type in the below given code then go to the URL to get the code to access colab and then finally paste the authentication code to get access to our Google drive where the data is stored.

```
from google.colab import drive
drive.mount('/content/drive/')
Go to this URL in a browser:
https://accounts.google.com/o/oauth2/auth?client_id=947318989803-
6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive/
```

[1] https://colab.research.google.com/notebooks/intro.ipynb

4 Import Libraries[2]

```
import pandas as pd
import os
from glob import glob
import numpy as np
import cv2
import os
import sys
import cv2
import shutil
import random
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import multiprocessing as mp
import matplotlib.pyplot as plt

from keras.activations import elu

from sklearn.utils import class_weight
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, cohen_kappa_score
from keras import backend as K
from keras.models import Model
from keras.utils import to_categorical
from keras import optimizers, applications
from keras.layers import Dense, Dropout, GlobalAveragePooling2D, Input
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, Callback, LearningRat
eScheduler


from sklearn.metrics import classification_report
from imgaug import augmenters as iaa
from keras.applications.vgg16 import VGG16
from keras_preprocessing.image import ImageDataGenerator
from skimage import img_as_ubyte
from skimage import exposure
from skimage.exposure import histogram
from skimage.color import rgb2gray
from PIL import ImageFile
```

[2] https://keras.io/

```
ImageFile.LOAD_TRUNCATED_IMAGES = True
from keras import models
from keras import layers
from keras import optimizers
from keras import losses
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.applications import DenseNet169
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, Flatten, BatchNormalization
from keras.optimizers import Adam
```

5 Uninstall Keras

```
!pip uninstall keras-preprocessing
```

```
Uninstalling Keras-Preprocessing-1.1.2:
  Would remove:
    /usr/local/lib/python3.6/dist-packages/Keras_Preprocessing-
1.1.2.dist-info/*
    /usr/local/lib/python3.6/dist-packages/keras_preprocessing/*
Proceed (y/n)? y
  Successfully uninstalled Keras-Preprocessing-1.1.2
```

6 Install the Git version of Keras

```
!pip install git+https://github.com/keras-team/keras-preprocessing.git
```

7 Import Dataset and CSV Files

```
train_data = '/content/drive/My Drive/DR_Levels/Training'
valid_data = '/content/drive/My Drive/DR_Levels/Validation'
test_data = '/content/drive/My Drive/DR_Levels/Evaluation'
```

```
train_df = pd.read_csv('/content/drive/My Drive/DR_Levels/regular-fundus-training.csv')
valid_df = pd.read_csv('/content/drive/My Drive/DR_Levels/regular-fundus-validation.csv')
test_df = pd.read_csv('/content/drive/My Drive/DR_Levels/Challenge1_upload.csv')
```

8 Map the CSV with the image files

```
train_df['image_id'] = train_df['image_id'] + ".jpg"# Two meathods add jpg
valid_df['image_id'] = valid_df['image_id'] + ".jpg"# Two meathods add jpg
test_df["image_id"] = test_df["image_id"].apply(lambda x: x + ".jpg")
```

9 Convert the target variable to String

```
train_df['patient_DR_Level'] = train_df['patient_DR_Level'].apply(str)
valid_df['patient_DR_Level'] = valid_df['patient_DR_Level'].apply(str)
```

10 Creating Functions for Pre-processing

10.1 Creating image tolerance level

```
def crop_image1(img,tol=7):
    # img is image data
    # tol is tolerance

    mask = img>tol
    return img[np.ix_(mask.any(1),mask.any(0))]
```

10.2 Converting the image to Grayscale and perform circular crop

```
 def crop_image_from_gray(img, tol=7):
  if img.ndim == 2:
       mask = img > tol
       return img[np.ix_(mask.any(1),mask.any(0))]
   # If we have a normal RGB images
  elif img.ndim == 3:
       gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
       mask = gray_img > tol

       check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
       if (check_shape == 0): # image is too dark so that we crop out everything,
```

```
            return img # return original image
        else:
            img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
            img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
            img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
            img = np.stack([img1,img2,img3],axis=-1)
        return img
```

10.3 Function to add Gaussian Blur to Image

```
def ben_color(image, sigmaX=20):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = crop_image_from_gray(image)
    image = cv2.resize(image, (299, 299))

    height, width, depth = image.shape

    x = int(width/2)
    y = int(height/2)
    r = np.amin((x,y))

    circle_img = np.zeros((height, width), np.uint8)
    cv2.circle(circle_img, (x,y), int(r), 1, thickness=-1)
    image = cv2.bitwise_and(image, image, mask=circle_img)

    image = cv2.addWeighted ( image,4, cv2.GaussianBlur( image , (0,0) , sigmaX=20)
 ,-4 ,128)

    return image
```

11 Image Transformation, Augmentation & Pre-Processing

```
 datagen = ImageDataGenerator(rescale=1. / 255,
                                    rotation_range=15,
                                    width_shift_range=0.1,
                                    #zca_whitening = True,
                                    height_shift_range=0.1,
                                    shear_range=0.01,
                                    zoom_range=[0.9, 1.25],
                                    horizontal_flip=True,
                                    vertical_flip=True,
                                    fill_mode='nearest',
                                    brightness_range=[0.5, 1.5],
                                   preprocessing_function=ben_color )
```

12 Creating Train_Generator

```
train_generator=datagen.flow_from_dataframe(dataframe = train_df,
                                            directory = train_data,
                                            x_col="image_id",
                                            y_col="patient_DR_Level",
                                            shuffle = True,
                                            batch_size=batch_size,
                                            target_size=(299, 299),
                                            class_mode = "categorical")
```

13 Creating Valid_Generator

```
valid_generator=datagen.flow_from_dataframe(dataframe = valid_df,
                                            directory = valid_data,
                                            x_col="image_id",
                                            y_col="patient_DR_Level",
                                            batch_size=batch_size,
                                            target_size=(299, 299),
                                            shuffle = True,
                                            class_mode = "categorical")
```

14 Visualizing Image after Pre-Processing

```
from keras.preprocessing import image
x,y=valid_generator.next()
for i in range(0,2):
    image=x[i]
    label=y[i]
    print(label)
    plt.imshow(image)
    plt.show()
```

Figure 15 – Original Image

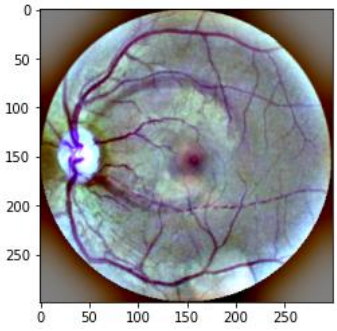

Figure 15 – Pre-processed Image

15 Model Creation

Six different types of pre-trained models were implemented out of which ResNet-50, Inception-v3, InceptionResNet-v2 were fine-tuned, whereas DenseNet-169, DenseNet-201 and Inception-v4 was used as conventional models.

15.1 Building the Model DenseNet-169

```
densenet = DenseNet169(
    weights='/content/drive/My Drive/DenseNet-BC-169-32-no-top.h5',
    include_top=False,
    input_shape=(299,299,3)
)

def build_model():
    model = Sequential()
    model.add(densenet)
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(5, activation='softmax'))

    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=Adam(lr=0.000005),
        metrics=['accuracy']
    )
```

```
    return model

model = build_model()
model.summary()
```

15.1.1 Steps per training/Steps per Validation

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)
```

15.1.2 Creating callback list

```
es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE, restore_be
st_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE,
factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accu
racy'])
model.summary()
```

15.1.3 Model Implementation

```
 history_finetunning = model.fit_generator(generator=train_generator,
                                           steps_per_epoch=STEP_SIZE_TRAIN,
                                           validation_data=valid_generator,
                                           validation_steps=STEP_SIZE_VALID,
                                           epochs=50,
                                           callbacks=callback_list)
```

15.2 Building the Model DenseNet-201

```
densenet = DenseNet201(
    weights='/content/drive/My Drive/densenet201_weights_tf_dim_ordering_tf_kernels
_notop.h5',
    include_top=False,
    input_shape=(299,299,3)
)

 def build_model():
```

```
    model = Sequential()
    model.add(densenet)
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(5, activation='softmax'))

    model.compile(
        loss='categorical_crossentropy',
        optimizer=Adam(lr=0.000005),
        metrics=['accuracy']
    )

    return model

model = build_model()
model.summary()
```

15.2.1 Steps per training/Steps per Validation

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)
```

15.2.2 Creating Callback List

```
es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE, restore_be
st_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE,
factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="binary_crossentropy",  metrics=['accuracy'
])
model.summary()
```

15.2.3 Model Implementation

```
history_finetunning = model.fit_generator(generator=train_generator,
                                          steps_per_epoch=STEP_SIZE_TRAIN,
                                          validation_data=valid_generator,
                                          validation_steps=STEP_SIZE_VALID,
                                          epochs=50,
                                          callbacks=callback_list,
                                          verbose=1)
```

15.3 Inception-V4

15.3.1 Adding weights for Inception V4

```
WEIGHTS_PATH = 'https://github.com/kentsommer/keras-
inceptionV4/releases/download/2.1/inception-
v4_weights_tf_dim_ordering_tf_kernels.h5'
WEIGHTS_PATH_NO_TOP = 'https://github.com/kentsommer/keras-
inceptionV4/releases/download/2.1/inception-
v4_weights_tf_dim_ordering_tf_kernels_notop.h5'

def preprocess_input(x):
    x = np.divide(x, 255.0)
    x = np.subtract(x, 0.5)
    x = np.multiply(x, 2.0)
    return x
```

15.3.2 Creating Convolutional Block

```
def conv2d_bn(x, nb_filter, num_row, num_col,
              padding='same', strides=(1, 1), use_bias=False):
    """
    Utility function to apply conv + BN.
    (Slightly modified from https://github.com/fchollet/keras/blob/master/keras/app
lications/inception_v3.py)
    """
    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = -1
    x = Convolution2D(nb_filter, (num_row, num_col),
                      strides=strides,
                      padding=padding,
                      use_bias=use_bias,
                      kernel_regularizer=regularizers.l2(0.00004),
                      kernel_initializer=initializers.VarianceScaling(scale=2.0, mo
de='fan_in', distribution='normal', seed=None))(x)
    x = BatchNormalization(axis=channel_axis, momentum=0.9997, scale=False)(x)
    x = Activation('relu')(x)
    return x
```

15.3.3 Adding an Inception Block A

```
def block_inception_a(input):
    if K.image_data_format() == 'channels_first':
```

```
        channel_axis = 1
    else:
        channel_axis = -1

    branch_0 = conv2d_bn(input, 96, 1, 1)

    branch_1 = conv2d_bn(input, 64, 1, 1)
    branch_1 = conv2d_bn(branch_1, 96, 3, 3)

    branch_2 = conv2d_bn(input, 64, 1, 1)
    branch_2 = conv2d_bn(branch_2, 96, 3, 3)
    branch_2 = conv2d_bn(branch_2, 96, 3, 3)

    branch_3 = AveragePooling2D((3,3), strides=(1,1), padding='same')(input)
    branch_3 = conv2d_bn(branch_3, 96, 1, 1)

    x = concatenate([branch_0, branch_1, branch_2, branch_3], axis=channel_axis)
    return x
```

15.3.4 Adding a Reduction Block A

```
def block_reduction_a(input):
    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = -1

    branch_0 = conv2d_bn(input, 384, 3, 3, strides=(2,2), padding='valid')

    branch_1 = conv2d_bn(input, 192, 1, 1)
    branch_1 = conv2d_bn(branch_1, 224, 3, 3)
    branch_1 = conv2d_bn(branch_1, 256, 3, 3, strides=(2,2), padding='valid')

    branch_2 = MaxPooling2D((3,3), strides=(2,2), padding='valid')(input)

    x = concatenate([branch_0, branch_1, branch_2], axis=channel_axis)
    return x
```

15.3.5 Adding Inception Block B

```
def block_inception_b(input):
    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = -1

    branch_0 = conv2d_bn(input, 384, 1, 1)
```

```
    branch_1 = conv2d_bn(input, 192, 1, 1)
    branch_1 = conv2d_bn(branch_1, 224, 1, 7)
    branch_1 = conv2d_bn(branch_1, 256, 7, 1)

    branch_2 = conv2d_bn(input, 192, 1, 1)
    branch_2 = conv2d_bn(branch_2, 192, 7, 1)
    branch_2 = conv2d_bn(branch_2, 224, 1, 7)
    branch_2 = conv2d_bn(branch_2, 224, 7, 1)
    branch_2 = conv2d_bn(branch_2, 256, 1, 7)

    branch_3 = AveragePooling2D((3,3), strides=(1,1), padding='same')(input)
    branch_3 = conv2d_bn(branch_3, 128, 1, 1)

    x = concatenate([branch_0, branch_1, branch_2, branch_3], axis=channel_axis)
    return x
```

15.3.6 Adding Reduction Block B

```
def block_reduction_b(input):
    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = -1

    branch_0 = conv2d_bn(input, 192, 1, 1)
    branch_0 = conv2d_bn(branch_0, 192, 3, 3, strides=(2, 2), padding='valid')

    branch_1 = conv2d_bn(input, 256, 1, 1)
    branch_1 = conv2d_bn(branch_1, 256, 1, 7)
    branch_1 = conv2d_bn(branch_1, 320, 7, 1)
    branch_1 = conv2d_bn(branch_1, 320, 3, 3, strides=(2,2), padding='valid')

    branch_2 = MaxPooling2D((3, 3), strides=(2, 2), padding='valid')(input)

    x = concatenate([branch_0, branch_1, branch_2], axis=channel_axis)
    return x
```

15.3.7 Adding Inception Block C

```
def block_inception_c(input):
    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = -1

    branch_0 = conv2d_bn(input, 256, 1, 1)
```

```
    branch_1 = conv2d_bn(input, 384, 1, 1)
    branch_10 = conv2d_bn(branch_1, 256, 1, 3)
    branch_11 = conv2d_bn(branch_1, 256, 3, 1)
    branch_1 = concatenate([branch_10, branch_11], axis=channel_axis)


    branch_2 = conv2d_bn(input, 384, 1, 1)
    branch_2 = conv2d_bn(branch_2, 448, 3, 1)
    branch_2 = conv2d_bn(branch_2, 512, 1, 3)
    branch_20 = conv2d_bn(branch_2, 256, 1, 3)
    branch_21 = conv2d_bn(branch_2, 256, 3, 1)
    branch_2 = concatenate([branch_20, branch_21], axis=channel_axis)

    branch_3 = AveragePooling2D((3, 3), strides=(1, 1), padding='same')(input)
    branch_3 = conv2d_bn(branch_3, 256, 1, 1)

    x = concatenate([branch_0, branch_1, branch_2, branch_3], axis=channel_axis)
    return x
```

15.3.8 Creating Inception V4 Base

```
def inception_v4_base(input):
    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = -1

    # Input Shape is 299 x 299 x 3 (th) or 3 x 299 x 299 (th)
    net = conv2d_bn(input, 32, 3, 3, strides=(2,2), padding='valid')
    net = conv2d_bn(net, 32, 3, 3, padding='valid')
    net = conv2d_bn(net, 64, 3, 3)

    branch_0 = MaxPooling2D((3,3), strides=(2,2), padding='valid')(net)

    branch_1 = conv2d_bn(net, 96, 3, 3, strides=(2,2), padding='valid')

    net = concatenate([branch_0, branch_1], axis=channel_axis)

    branch_0 = conv2d_bn(net, 64, 1, 1)
    branch_0 = conv2d_bn(branch_0, 96, 3, 3, padding='valid')

    branch_1 = conv2d_bn(net, 64, 1, 1)
    branch_1 = conv2d_bn(branch_1, 64, 1, 7)
    branch_1 = conv2d_bn(branch_1, 64, 7, 1)
    branch_1 = conv2d_bn(branch_1, 96, 3, 3, padding='valid')

    net = concatenate([branch_0, branch_1], axis=channel_axis)

    branch_0 = conv2d_bn(net, 192, 3, 3, strides=(2,2), padding='valid')
```

```
    branch_1 = MaxPooling2D((3,3), strides=(2,2), padding='valid')(net)

    net = concatenate([branch_0, branch_1], axis=channel_axis)

    # 35 x 35 x 384
    # 4 x Inception-A blocks
    for idx in range(4):
      net = block_inception_a(net)

    # 35 x 35 x 384
    # Reduction-A block
    net = block_reduction_a(net)

    # 17 x 17 x 1024
    # 7 x Inception-B blocks
    for idx in range(7):
      net = block_inception_b(net)

    # 17 x 17 x 1024
    # Reduction-B block
    net = block_reduction_b(net)

    # 8 x 8 x 1536
    # 3 x Inception-C blocks
    for idx in range(3):
      net = block_inception_c(net)

    return net
```

15.3.9 Final Model Creation

```
def inception_v4(num_classes, dropout_keep_prob, weights, include_top):
    '''
    Creates the inception v4 network
    Args:
      num_classes: number of classes
      dropout_keep_prob: float, the fraction to keep before final layer.

    Returns:
      logits: the logits outputs of the model.
    '''

    # Input Shape is 299 x 299 x 3 (tf) or 3 x 299 x 299 (th)
    if K.image_data_format() == 'channels_first':
        inputs = Input((3, 299, 299))
    else:
        inputs = Input((299, 299, 3))
```

```
    # Make inception base
    x = inception_v4_base(inputs)


    # Final pooling and prediction
    if include_top:
        # 1 x 1 x 1536
        x = AveragePooling2D((8,8), padding='valid')(x)
        x = Dropout(dropout_keep_prob)(x)
        x = Flatten()(x)
        # 1536
        x = Dense(units=num_classes, activation='softmax')(x)

    model = Model(inputs, x, name='inception_v4')

    # load weights
    if weights == 'imagenet':
        if K.image_data_format() == 'channels_first':
            if K.backend() == 'tensorflow':
                warnings.warn('You are using the TensorFlow backend, yet you '
                              'are using the Theano '
                              'image data format convention '
                              '(`image_data_format="channels_first"`). '
                              'For best performance, set '
                              '`image_data_format="channels_last"` in '
                              'your Keras config '
                              'at ~/.keras/keras.json.')
        if include_top:
            weights_path = get_file(
                'inception-v4_weights_tf_dim_ordering_tf_kernels.h5',
                WEIGHTS_PATH,
                cache_subdir='models',
                md5_hash='9fe79d77f793fe874470d84ca6ba4a3b')
        else:
            weights_path = get_file(
                'inception-v4_weights_tf_dim_ordering_tf_kernels_notop.h5',
                WEIGHTS_PATH_NO_TOP,
                cache_subdir='models',
                md5_hash='9296b46b5971573064d12e4669110969')
        model.load_weights(weights_path, by_name=True)
    return model


def create_model(num_classes=1001, dropout_prob=0.2, weights=None, include_top=True
):
    return inception_v4(num_classes, dropout_prob, weights, include_top)
```

15.3.10 Adding the notop weight to the model

```
incept_model = create_model(num_classes=1001, dropout_prob=0.2, weights=None, inclu
de_top=False)
incept_model.load_weights('/content/drive/My Drive/inception-
v4_weights_tf_dim_ordering_tf_kernels_notop.h5')
```

15.3.11 Adding layers to the model

```
for l in incept_model.layers:
    if l is not None: l.trainable = True

x = incept_model.output
x = GlobalAveragePooling2D(data_format='channels_last')(x)
x = BatchNormalization()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(2048, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(5, activation='softmax')(x)

model = Model(inputs=incept_model.input, outputs=predictions)
model.summary()
```

15.3.12 Creating Training step size and Validation Step Size

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)
```

15.3.13 Creating callback list

```
es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE, restore_be
st_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE,
factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="mse",  metrics=['accuracy'])
model.summary()
```

15.3.14 Model Implementation

```
history_finetunning = model.fit_generator(generator=train_generator,
                                          steps_per_epoch=STEP_SIZE_TRAIN,
```

```
                                    validation_data=valid_generator,
                                    validation_steps=STEP_SIZE_VALID,
                                    epochs=50,
                                    callbacks=callback_list,
                                    verbose=1)
```

15.4 Building the Model ResNet-50

```
def create_model(input_shape, n_out):
    input_tensor = Input(shape=input_shape)
    base_model = applications.ResNet50(weights=None, include_top=False,input_tensor
=input_tensor)
    base_model.load_weights('/content/drive/My Drive/resnet50_weights_tf_dim_orderi
ng_tf_kernels_notop.h5')

    x = GlobalAveragePooling2D()(base_model.output)
    x = Dropout(0.5)(x)
    x = Dense(2048, activation='relu')(x)
    x = Dropout(0.5)(x)
    final_output = Dense(n_out, activation='softmax', name='final_output')(x)
    model = Model(input_tensor, final_output)
    return model

model = create_model(input_shape=(299, 299, 3), n_out=5)

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

for i in range(-5, 0):
    model.layers[i].trainable = True
model.summary()
```

15.4.1 Creating Training step size and validation step size

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)
```

15.4.2 Variables used during training

```
#EPOCHS = 40
WARMUP_EPOCHS = 2
LEARNING_RATE = 1e-4
WARMUP_LEARNING_RATE = 1e-3
#HEIGHT = 320
#WIDTH = 320
#CANAL = 3
```

```
#N_CLASSES = df_train_train['diagnosis'].nunique()
ES_PATIENCE = 5
RLROP_PATIENCE = 3
DECAY_DROP = 0.5
```

15.4.3 Compiling the model with warm-up epochs

```
model.compile(optimizer = optimizers.Adam(lr=WARMUP_LEARNING_RATE),loss = 'categori
cal_crossentropy',metrics = ['accuracy'])

history_warmup = model.fit_generator(generator=train_generator,
                                     steps_per_epoch=STEP_SIZE_TRAIN,
                                     validation_data=valid_generator,validation_ste
ps=STEP_SIZE_VALID,
                                     epochs=WARMUP_EPOCHS,
                                     verbose=1).history
```

15.4.4 Fine-Tuning the model and adding callback list

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


es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE, restore_be
st_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE,
factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="binary_crossentropy",  metrics=['accuracy'
])
model.summary()
```

15.4.5 Final Implementation after fine-tuning

```
history_finetunning = model.fit_generator(generator=train_generator,
                                          steps_per_epoch=STEP_SIZE_TRAIN,
                                          validation_data=valid_generator,
                                          validation_steps=STEP_SIZE_VALID,
                                          epochs=50,
                                          callbacks=callback_list,
                                          verbose=1).history
```

15.5 Building the model Inception-v3

```
def create_model(input_shape, n_out):
    input_tensor = Input(shape=input_shape)
```

```
    base_model = applications.InceptionV3(weights=None, include_top=False,input_ten
sor=input_tensor)
    base_model.load_weights('/content/drive/My Drive/inception_v3_weights_tf_dim_or
dering_tf_kernels_notop.h5')

    x = GlobalAveragePooling2D()(base_model.output)
    x = Dropout(0.5)(x)
    x = Dense(2048, activation='relu')(x)
    x = Dropout(0.5)(x)
    final_output = Dense(n_out, activation='softmax', name='final_output')(x)
    model = Model(input_tensor, final_output)
    return model

model = create_model(input_shape=(299, 299, 3), n_out=5)

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

for i in range(-5, 0):
    model.layers[i].trainable = True
model.summary()
```

15.5.1 Creating training step size and validation step size

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)
```

15.5.2 Variables used in the training

```
#EPOCHS = 40
WARMUP_EPOCHS = 2
LEARNING_RATE = 1e-4
WARMUP_LEARNING_RATE = 1e-3
#HEIGHT = 320
#WIDTH = 320
#CANAL = 3
#N_CLASSES = df_train_train['diagnosis'].nunique()
ES_PATIENCE = 5
RLROP_PATIENCE = 3
DECAY_DROP = 0.5
```

15.5.3 Implementing the model with warm up epochs

```
model.compile(optimizer = optimizers.Adam(lr=WARMUP_LEARNING_RATE),loss = 'categori
cal_crossentropy',metrics = ['accuracy'])

history_warmup = model.fit_generator(generator=train_generator,
```

```
                                        steps_per_epoch=STEP_SIZE_TRAIN,
                                        validation_data=valid_generator,validation_ste
ps=STEP_SIZE_VALID,
                                        epochs=WARMUP_EPOCHS,
                                        verbose=1).history
```

15.5.4 Fine tuning the model and adding callback list

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


es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE, restore_be
st_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE,
factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="binary_crossentropy",  metrics=['accuracy'
])
model.summary()
```

15.5.5 Final Implementation after Fine-Tuning

```
history_finetunning = model.fit_generator(generator=train_generator,
                                          steps_per_epoch=STEP_SIZE_TRAIN,
                                          validation_data=valid_generator,
                                          validation_steps=STEP_SIZE_VALID,
                                          epochs=50,
                                          callbacks=callback_list,
                                          verbose=1).history
```

15.6 Building the model Inception-ResNet-v2

```
def create_model(input_shape, n_out):

    pretrain_model = InceptionResNetV2(
        include_top=False,
        weights='imagenet',
        input_shape=input_shape)

    input_tensor = Input(shape=input_shape)
    x = GlobalAveragePooling2D()(pretrain_model.output)
    bn = BatchNormalization()(input_tensor)
    x = pretrain_model(bn)
    x = Conv2D(128, kernel_size=(1,1), activation='relu')(x)
```

```
    x = Flatten()(x)
    x = Dropout(0.5)(x)
    x = Dense(2048, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(n_out, activation='softmax')(x)
    model = Model(input_tensor, output)

    return model

model = create_model(
    input_shape=(299,299,3),
    n_out=5)


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

for i in range(-9, 0):
    model.layers[i].trainable = True
model.summary()
```

15.6.1 Creating training step size and validation step size

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)
```

15.6.2 Variables used during training

```
#EPOCHS = 40
WARMUP_EPOCHS = 2
LEARNING_RATE = 1e-4
WARMUP_LEARNING_RATE = 1e-3
#HEIGHT = 320
#WIDTH = 320
#CANAL = 3
#N_CLASSES = df_train_train['diagnosis'].nunique()
ES_PATIENCE = 5
RLROP_PATIENCE = 3
DECAY_DROP = 0.5
```

15.6.3 Implementing the model with warm up epochs

```
model.compile(optimizer = Adam(lr=WARMUP_LEARNING_RATE),loss = 'categorical_crossen
tropy',metrics = ['accuracy'])

history_warmup = model.fit_generator(generator=train_generator,
                                     steps_per_epoch=STEP_SIZE_TRAIN,
```

```
                                        validation_data=valid_generator,validation_ste
ps=STEP_SIZE_VALID,
                                        epochs=WARMUP_EPOCHS,
                                        verbose=1).history
```

15.6.4 Fine Tuning the model and creating callback List

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

es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE, restore_be
st_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE,
factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="binary_crossentropy",  metrics=['accuracy'
])
model.summary()
```

15.6.5 Final Implementation after fine - tuning the model

```
history_finetunning = model.fit_generator(generator=train_generator,
                                          steps_per_epoch=STEP_SIZE_TRAIN,
                                          validation_data=valid_generator,
                                          validation_steps=STEP_SIZE_VALID,
                                          epochs=50,
                                          callbacks=callback_list,
                                          verbose=1).history
```