Configuration Manual

MSc Research Project
MSc in Data Analytics

Ifeoma Oduntan
Student ID: X18191274

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ifeoma Oduntan
Student ID:	X18191274
Programme:	MSc in Data Analytics **Year:** 2022
Module:	Research Project
Lecturer:	Jorge Basilio
Submission Due Date:	15/08/2022
Project Title:	Analysis of Blood Image in Sickle Cell Classification Using Deep Learning Algorithm
Word Count:	**1 Page Count: 48**

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: ………

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

Ifeoma Oduntan
Student ID: X18191274

1 Introduction

The configuration manual provides information that supports the implementation of this research project. This includes comprehensive information on software tools used on step by step 'Classification of Sickle Cell Disease' using Generative Adversarial Networks (GAN) to synthesize additional images and implementation of seven deep learning models. From equipment and environment set up to downloading the dataset, mounting the drive, installing, and importing the required libraries to reading the data into Google Colaboratory (Google Colab). It shows the execution of the project to arrive at the produced results.

It is divided into sections as follows: Section one shows the setting up of the environment, while section two shows the loading of the required libraries. Section three is the overview of the dataset, while section four shows the loading of the data into Google Colab from Google drive where the data is stored and exploratory data analysis, augmentation and synthesizing of additional images using GAN (Generative Adversarial Network) Then section five shows the implementation of the different models.

2 Hardware Details

A Python 3.8 environment programming language in Google Colab, using Intel(R) Core (TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz windows laptop
The hardware specifications used in this research are listed below.in subsection 2.1

2.1 Device Specification

- Device name LAPTOP-F6DVNM05
- Processor Intel(R) Core (TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
- Installed RAM16.0 GB (15.8 GB usable)
- Device ID B4FC212C-5AB0-4E4C-BCF7-F22FC2E85CFD
- Product ID 00325-96694-08330-AAOEM
- System type 64-bit operating system, x64-based processor
- Pen and touch Touch support with 10 touch points.

2.2 Windows Specifications

- EditionWindows 11 Home
- Version 21H2
- Installed on 19/05/2022
- OS build 22000.795
- Experience Windows Feature Experience Pack 1000.22000.795.0

2.3 Software Specification

- ❖ Google Colaboratory Pro – Upgrading from the standard Colab to Colab Pro by purchasing a monthly subscription from Google Colab to increase the memory size if not the system continues to crash.
- ❖ Jupyter Notebook

2.4 Programming Requisites

- ❖ Python (Version 3.8)
- ❖ TensorFlow (Version 2 8 2)
- ❖ Google Colab has most of the requirements preinstalled, but some uninstalled required libraries can be installed using! pip install (required library name)

3 Required Libraries

The following required libraries used in the implementation of this research project are listed below:

```
## Import the required libraries
from numpy.random import seed
seed(42)
import tensorflow
tensorflow.random.set_seed(42)
seed = 42
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from tqdm import tqdm
from PIL import Image
from sklearn.utils import shuffle
import os
import glob
import keras
import random
import math
import seaborn as sns
import sys
import datetime
import glob as glob
import matplotlib.cm as cm
from sklearn import metrics
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import applications
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Dropout, Flatten,GlobalAveragePooling2D
```

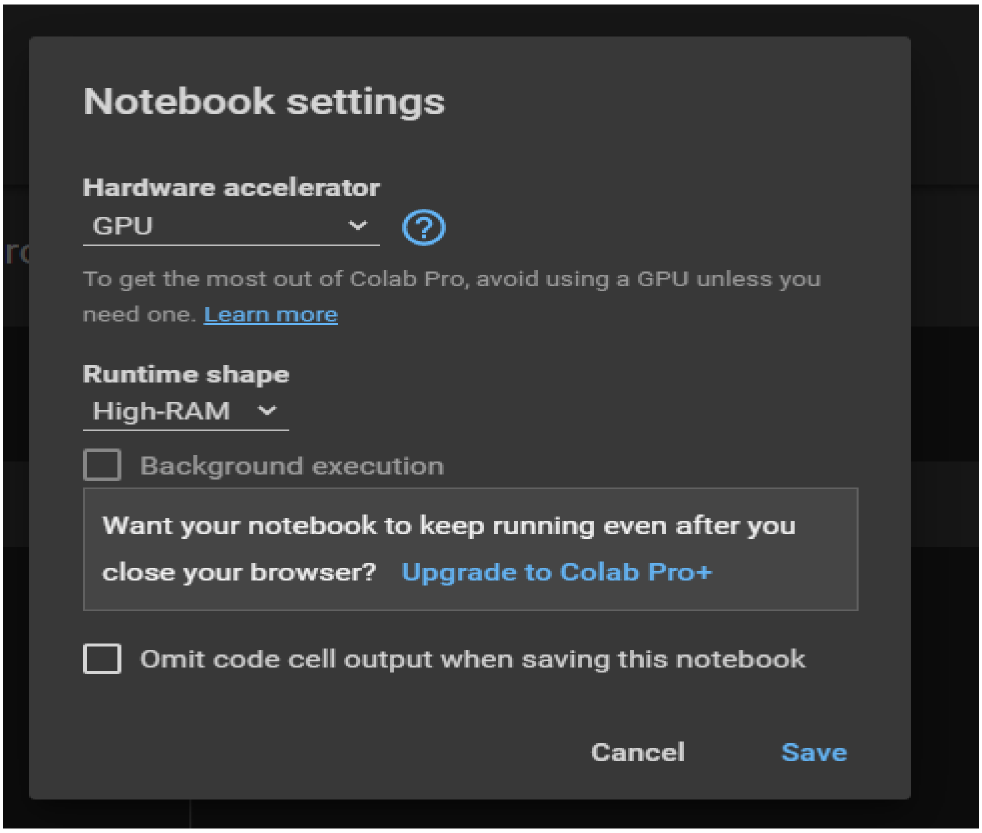
```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Lambda
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.applications.densenet import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,img_to_array, load_img
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping, Callback

import warnings
warnings.filterwarnings("ignore")
```

4 Dataset

The dataset used in this research project is the erythrocytesIDB1. This dataset is a privately owned dataset and only released on request from the following website http://erythrocytesidb.uib.es/ by emailing a filled form and agreeing to the terms and conditions of the data owner. The dataset was first downloaded from the provided link by the owner through email from the cloud and saved to google drive. The dataset has a total 196 full size images and 626 individual images in 3 classes in jpg format. The notebook is compatible as files and folders can be uploaded to the Colab using the file drop down menu and select upload and then select the file or folder upload option which opens a browser to select where you want to upload the files from. If the data is saved in google drive, then the drive should be mounted in google Colab to load the data.

5 Loading The Data

The data was unzipped using zip extractor and saved in the google drive. The files and folders can be uploaded to the Colab using the file drop down menu and select upload and then select the file or folder upload option which opens a browser to select where you want to upload the files from. If the data is saved in google drive, then the drive should be mounted in google Colab to load the data as shown below. Also go to runtime on the ribbon bar and change runtime from None to GPU, and below it, change the Standard to High RAM.



The image folder was first read into Google Colab by first mounting the google drive on the Colab environment. The first time you use the Colab to mount the drive, you will be directed to a URL in a browser to get the authorization code which is copied and pasted in Colab before the drive is mounted.

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

```
from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive
```

The version of the TensorFlow that was used for running the code is shown below.

```
import tensorflow as tf
print(tf.__version__)

2.8.2
```

6 Exploratory Data Analysis

The first thing is to load all the necessary libraries for exploring the dataset and visualizing some of the images in each class. The dataset is processed, and the models were used in classifying the images first.

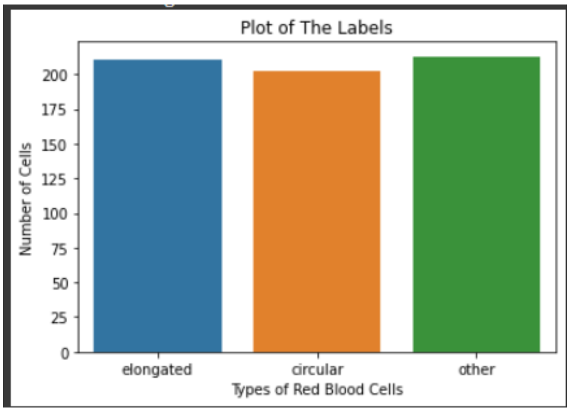
```
#import the libraries
import seaborn as sns
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
```

```
dataset_path = os.listdir('/content/drive/MyDrive/erythrocytes')

print (dataset_path)  #what kinds of red blood cells are in this dataset

print("Types of classes labels found: ", len(dataset_path))

class_labels = []

for item in dataset_path:
 # Get all the file names
 all_classes = os.listdir('/content/drive/MyDrive/erythrocytes' + '/' +item)
 #print(all_classes)

 # Add them to the list
 for cell in all_classes:
    class_labels.append((item, str('dataset_path' + '/' +item) + '/' + cell))
    print(class_labels[:3])


# Build a dataframe
df = pd.DataFrame(data=class_labels, columns=['Labels', 'images'])
print(df.head())
print(df.tail())
```

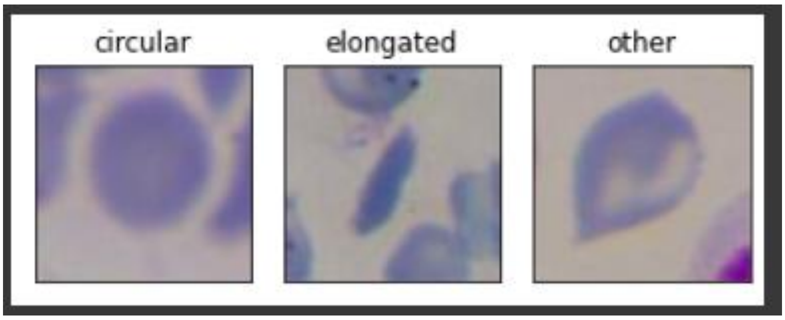
```
[('elongated', 'dataset_path/elongated/e0082.jpg'), ('elongated', 'dataset_path/elongated/
[('elongated', 'dataset_path/elongated/e0082.jpg'), ('elongated', 'dataset_path/elongated/
[('elongated', 'dataset_path/elongated/e0082.jpg'), ('elongated', 'dataset_path/elongated/
[('elongated', 'dataset_path/elongated/e0082.jpg'), ('elongated', 'dataset_path/elongated/
```

```
# Checking how many samples are present in each category
print("Total number of images in the dataset: ", len(df))

label_count = df['Labels'].value_counts()
print(label_count)
```

```
Total number of images in the dataset:  626
other         213
elongated     211
circular      202
Name: Labels, dtype: int64
```



Bar plot of the classes



Sample image of each class.

7 Traditional Data Augmentation

The required libraries were loaded and the **erythrocytes** dataset folder which comprised of the original images were used for generating more images through traditional data augmentation. Each image was augmented 7 times using rotation, vertical and horizontal flips, zoom range, shear range, width, and height shift. The generated images were saved in a folder named **erythrocytes2** which already contains the original images. A copy of the original folder named **erythrocytes3** was made. The reason behind making a copy of this folder is because 2000 GAN generated images of each class are added to the **erythrocytes3** folder.

```
# Constructing an instance of the ImageDataGenerator class
# Pass the augmentation parameters through the constructor.

datagen = ImageDataGenerator(
        rotation_range=30,     #Random rotation between 0 and 45
        width_shift_range=0.2,   #% shift
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        vertical_flip=True,
        horizontal_flip=True,
        fill_mode='constant', cval=125)    #Also try nearest, constant, reflect, wrap
```

```
# Generating and saving augmented samples
# using the above defined parameters with datagen.flow which generates batches of randomly circular augmented images

i = 0
for batch in datagen.flow(x, batch_size=202,
                          save_to_dir='/content/drive/MyDrive/erythrocytes2/circular/',
                          save_prefix='c',
                          save_format='jpg'):
    i += 1
    if i > 6:
        break  # otherwise the generator would loop indefinitely
```

Generating the elongated class

```
# Generating and saving augmented samples
# using the above defined parameters with datagen . flow generates batches of randomly elongated augmented images

i = 0
for batch in datagen.flow(x, batch_size=211,
                          save_to_dir='/content/drive/MyDrive/erythrocytes2/elongated/',
                          save_prefix='e',
                          save_format='jpg'):
    i += 1
    if i > 6:
        break  # otherwise the generator would loop indefinitely
```

The augmented other class

```
# Generating and saving augmented samples
# using the above defined parameters with datagen . flow generates batches of randomly other augmented images

i = 0
for batch in datagen.flow(x, batch_size=213,
                          save_to_dir='/content/drive/MyDrive/erythrocytes2/other/',
                          save_prefix='o',
                          save_format='jpg'):
    i += 1
    if i > 6:
        break  # otherwise the generator would loop indefinitely
```

8 Implementation of Generative Adversarial Network (GAN) for Image Synthesis.

The dataset used for reading the images is from the **erythrocytes2** folder (which consists of the original images and traditional augmented images). 2000 generated images of each class are saved in **erythrocytes3** folder.

Each class takes between 20 to 30 minutes to train and generate the images.

Importing the required libraries

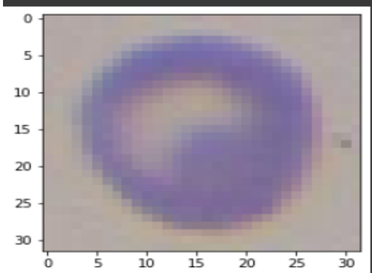
```python
# dcgan on erythrocytes dataset
from numpy import expand_dims
from numpy import zeros
from numpy import ones
from numpy import vstack
import os
import cv2
from tqdm import tqdm
import random
import pickle
from numpy.random import randn
from numpy.random import randint
import matplotlib.pyplot as plt
from keras.layers import Input
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dropout
from matplotlib import pyplot
```

GENERATING THE CIRCULAR CLASS IMAGES

```python
# define the standalone discriminator model
def define_discriminator(in_shape=(32,32,3)):
  model = Sequential()
  # normal
  model.add(Conv2D(64, (3,3), padding='same', input_shape=in_shape))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(256, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # classifier
  model.add(Flatten())
  model.add(Dropout(0.4))
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(learning_rate=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return model
```

```python
# define the standalone generator model
def define_generator(latent_dim):
  model = Sequential()
  # foundation for 4x4 image
  n_nodes = 256 * 4 * 4
  model.add(Dense(n_nodes, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Reshape((4, 4, 256)))
  # upsample to 8x8
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 16x16
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 32x32
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # output layer
  model.add(Conv2D(3, (3,3), activation='tanh', padding='same'))
  return model
```

```python
# load dataset
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
DATADIR = "/content/drive/MyDrive/erythrocytes2/"

CATEGORIES = [ "circular" ]

for category in CATEGORIES:  # elongated, circular and other
    path = os.path.join(DATADIR,category)  # create path to circular, elongated or other
    for img in os.listdir(path):  # iterate over each image for circular, elongated or other
        img_array = cv2.imread(os.path.join(path,img))
        img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB) # convert bgr to rgb
        img_array = cv2.resize(img_array, (32,32))
        plt.imshow(img_array)  # graph it
        plt.show()  # display!

        break  # we just want one for now so break

    break  #...and one more!
```




```python
# evaluate the discriminator, plot generated images, save generator model
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples=1200):
  # prepare real samples
  X_real, y_real = generate_real_samples(dataset, n_samples)
  # evaluate discriminator on real examples
  _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
  # prepare fake examples
  x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)
  # evaluate discriminator on fake examples
  _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
  # summarize discriminator performance
  print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real*100, acc_fake*100))
  # save plot
  save_plot(x_fake, epoch)
  # save the generator model tile file
  filename = 'generator_model_%03d.h5' % (epoch+1)
  g_model.save(filename)
```

```python
# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=500, n_batch=128):
  bat_per_epo = int(dataset.shape[0] / n_batch)
  half_batch = int(n_batch / 2)
  # manually enumerate epochs
  for i in range(n_epochs):
    # enumerate batches over the training set
    for j in range(bat_per_epo):
      # get randomly selected 'real' samples
      X_real, y_real = generate_real_samples(dataset, half_batch)
      # update discriminator model weights
      d_loss1, _ = d_model.train_on_batch(X_real, y_real)
      # generate 'fake' examples
      X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
      # update discriminator model weights
      d_loss2, _ = d_model.train_on_batch(X_fake, y_fake)
      # prepare points in latent space as input for the generator
      X_gan = generate_latent_points(latent_dim, n_batch)
      # create inverted labels for the fake samples
      y_gan = ones((n_batch, 1))
      # update the generator via the discriminator's error
      g_loss = gan_model.train_on_batch(X_gan, y_gan)
      # summarize loss on this batch
      print('>%d, %d/%d, d1=%.3f, d2=%.3f g=%.3f' %
        (i+1, j+1, bat_per_epo, d_loss1, d_loss2, g_loss))
    # evaluate the model performance, sometimes
    if (i+1) % 10 == 0:
      summarize_performance(i, g_model, d_model, dataset, latent_dim)
```

```python
# size of the latent space
latent_dim = 100
# create the discriminator
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
# train model
train(g_model, d_model, gan_model, dataset, latent_dim)
```

```
>1, 1/10, d1=0.694, d2=0.696 g=0.692
>1, 2/10, d1=0.631, d2=0.697 g=0.690
>1, 3/10, d1=0.572, d2=0.702 g=0.685
>1, 4/10, d1=0.492, d2=0.716 g=0.673
>1, 5/10, d1=0.391, d2=0.748 g=0.646
>1, 6/10, d1=0.264, d2=0.819 g=0.606
>1, 7/10, d1=0.180, d2=0.915 g=0.575
>1, 8/10, d1=0.146, d2=0.943 g=0.591
>1, 9/10, d1=0.117, d2=0.861 g=0.656
>1, 10/10, d1=0.116, d2=0.732 g=0.753
>2, 1/10, d1=0.076, d2=0.634 g=0.861
>2, 2/10, d1=0.056, d2=0.573 g=0.960
>2, 3/10, d1=0.033, d2=0.538 g=1.023
>2, 4/10, d1=0.035, d2=0.528 g=1.015
>2, 5/10, d1=0.037, d2=0.549 g=0.961
```

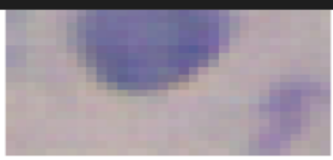
```
# example of loading the generator model and generating images
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[i, :, :,:])
  pyplot.show()

# load model
model = load_model('generator_model_500.h5')
# generate images
latent_points = generate_latent_points(100, 100)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
create_plot(X, 10)
```

```
#loading the generator model to generate some images and save in a folder
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n, temp):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[temp, :, :,:])
    filename2 = '/content/drive/MyDrive/erythrocytes3/circular/c'+str(temp)+'.jpg'
    pyplot.savefig(filename2)
  pyplot.show()
# load model
model = load_model('generator_model_500.h5')
```

```
# generate images
latent_points = generate_latent_points(100, 2000)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
for i in range(2000):
  create_plot(X,1, i)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
```

```
#loading the generator model to generate some images and save in GanImages folder
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n, temp):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[temp, :, :,:])
    filename2 = '/content/drive/MyDrive/GanImages1/circular/c'+str(temp)+'.jpg'
    pyplot.savefig(filename2)
  pyplot.show()
# load model
model = load_model('generator_model_500.h5')
# generate images
latent_points = generate_latent_points(100, 2000)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
for i in range(2000):
  create_plot(X,1, i)
```



2000 images were generated for the ‘circular’ class and saved in the ‘circular’ folder within the **erythrocytes3** and another 2000 images were saved in **GanImages1** folder for evaluation of the GAN generated images.

GENERATING THE ELONGATED CLASS IMAGES

```
# define the standalone discriminator model
def define_discriminator(in_shape=(32,32,3)):
  model = Sequential()
  # normal
  model.add(Conv2D(64, (3,3), padding='same', input_shape=in_shape))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(256, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # classifier
  model.add(Flatten())
  model.add(Dropout(0.4))
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(learning_rate=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return model
```

```
#loading the generator model to generate some images and save in a folder
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n, temp):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[temp, :, :,:])
    filename2 = '/content/drive/MyDrive/erythrocytes3/circular/c'+str(temp)+'.jpg'
    pyplot.savefig(filename2)
  pyplot.show()
# load model
model = load_model('generator_model_500.h5')
# generate images
latent_points = generate_latent_points(100, 2000)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
for i in range(2000):
  create_plot(X,1, i)
```

```
# size of the latent space
latent_dim = 100
# create the discriminator
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
# train model
train(g_model, d_model, gan_model, dataset, latent_dim)
```

```
>1, 1/10, d1=0.699, d2=0.696 g=0.691
>1, 2/10, d1=0.622, d2=0.698 g=0.689
>1, 3/10, d1=0.552, d2=0.706 g=0.682
>1, 4/10, d1=0.456, d2=0.724 g=0.665
>1, 5/10, d1=0.341, d2=0.763 g=0.638
>1, 6/10, d1=0.237, d2=0.828 g=0.612
>1, 7/10, d1=0.160, d2=0.876 g=0.610
>1, 8/10, d1=0.119, d2=0.880 g=0.638
>1, 9/10, d1=0.100, d2=0.858 g=0.685
>1, 10/10, d1=0.084, d2=0.822 g=0.757
>2, 1/10, d1=0.124, d2=0.763 g=0.849
>2, 2/10, d1=0.178, d2=0.693 g=0.965
```

```
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n, temp):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[temp, :, :,:])
    filename2 = '/content/drive/MyDrive/erythrocytes3/elongated/e'+str(temp)+'.jpg'
    pyplot.savefig(filename2)
  pyplot.show()
# load model
model = load_model('generator_model_500.h5')
# generate images
latent_points = generate_latent_points(100, 2000)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
for i in range(2000):
  create_plot(X,1, i)
```

```
#loading the generator model to generate some images and save in a folder
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n, temp):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[temp, :, :,:])
    filename2 = '/content/drive/MyDrive/GanImages1/elongated/e'+str(temp)+'.jpg'
    pyplot.savefig(filename2)
  pyplot.show()
# load model
model = load_model('generator_model_500.h5')
# generate images
latent_points = generate_latent_points(100, 2000)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
```

GENERATING THE OTHER CLASS IMAGES

```
[ ]
# define the standalone discriminator model
def define_discriminator(in_shape=(32,32,3)):
  model = Sequential()
  # normal
  model.add(Conv2D(64, (3,3), padding='same', input_shape=in_shape))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(256, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  # classifier
  model.add(Flatten())
  model.add(Dropout(0.4))
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(learning_rate=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return model
```

```
# size of the latent space
latent_dim = 100
# create the discriminator
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
# train model
train(g_model, d_model, gan_model, dataset, latent_dim)
```

```
>1, 1/10, d1=0.711, d2=0.696 g=0.692
>1, 2/10, d1=0.645, d2=0.698 g=0.689
>1, 3/10, d1=0.588, d2=0.703 g=0.684
>1, 4/10, d1=0.495, d2=0.718 g=0.671
>1, 5/10, d1=0.391, d2=0.753 g=0.642
>1, 6/10, d1=0.290, d2=0.829 g=0.599
```

```
# loading the generator model and generating images
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[i, :, :,:])
  pyplot.show()

# load model
model = load_model('generator_model_500.h5')
# generate images
latent_points = generate_latent_points(100, 100)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
create_plot(X, 10)
```

```
#loading the generator model to generate some images
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n, temp):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[temp, :, :,:])
    filename2 = '/content/drive/MyDrive/erythrocytes3/other/o'+str(temp)+'.jpg'
    pyplot.savefig(filename2)
  pyplot.show()
# load model
model = load_model('generator_model_500.h5')
# generate images
latent_points = generate_latent_points(100, 2000)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
```

```
#loading the generator model to generate some images and save in a folder
from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# plot the generated images
def create_plot(examples, n, temp):
  # plot images
  for i in range(n * n):
    # define subplot
    pyplot.subplot(n, n, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(examples[temp, :, :,:])
    filename2 = '/content/drive/MyDrive/GanImages1/other/o'+str(temp)+'.jpg'
    pyplot.savefig(filename2)
  pyplot.show()
# load model
model = load_model('generator_model_500.h5')
# generate images
latent_points = generate_latent_points(100, 2000)
# generate images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
```

8.1 Evaluating The GAN Generated Images and the Augmented Images

Implementation of Inception Score for GAN Evaluation

```python
#Evaluating first the augmented images
# loading the erythrocytes2(Augmented Images/Original Images)
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
#DATADIR = "C:\\Users\\iffyd\\Masters Project\\erythrocytes"
DATADIR = "/content/drive/MyDrive/erythrocytes2/"

CATEGORIES = [ "circular", "elongated", "other" ]

for category in CATEGORIES:  # elongated, circular and other
    path = os.path.join(DATADIR,category)  # create path to circular, elongated or other
    for img in os.listdir(path):  # iterate over each image for circular, elongated or other
        img_array = cv2.imread(os.path.join(path,img))
        img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB) # convert bgr to rgb
        img_array = cv2.resize(img_array, (32,32))
        plt.imshow(img_array)  # graph it
        plt.show()  # display!

        break  # we just want one for now so break

    break  #...and one more!
```


```python
training_data = []

def create_training_data():
    for category in CATEGORIES:  # for the classes (circular, elongated or other)

        path = os.path.join(DATADIR,category)  # create path to circular, elongated or other
        class_num = CATEGORIES.index(category)  # get the classification  (0, 1 or 2). 0=circular, 1=elongated, 2=other

        for img in tqdm(os.listdir(path)):  # iterate over each image for circular, elongated or other
            try:
                img_array = cv2.imread(os.path.join(path,img))  # convert to array
                img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB) # convert bgr to rgb
                new_array = cv2.resize(img_array, (32, 32))  # resize images
                training_data.append([new_array, class_num])  # add this to our training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()

print(len(training_data))
```

```
100%|██████████| 1616/1616 [00:03<00:00, 427.59it/s]
100%|██████████| 1688/1688 [00:03<00:00, 424.66it/s]
100%|██████████| 1697/1697 [00:03<00:00, 472.15it/s]5001
```

```python
#  prepare the training images
def load_real_samples():
  # the dataset
  dataset = (trainX, _), (_, _)
  # convert from unsigned ints to floats
  X = trainX.astype('float32')
  # scale from [0,255] to [-1,1]
  X = (X - 127.5) / 127.5
  #return X
```

```python
###Loading the data and shuffling to ensure each split covers a diverse set of classes.
# load the erythrocytes images
dataset = (trainX, _), (_, _)
# shuffle the images
shuffle(trainX)
```

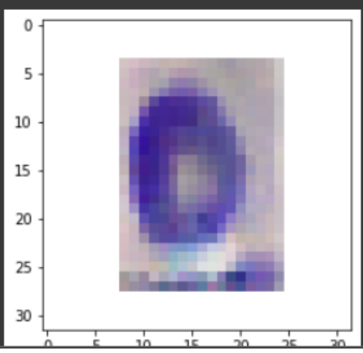
```
        [[ 50,  55, 157],
         [184, 174, 198],
         [226, 188, 199],
         ...,
         [ 27,  27,  27],
         [ 27,  27,  27],
         [ 27,  27,  27]]],


       [[[239, 223, 233],
         [220, 204, 215],
         [218, 199, 218],
         ...,
         [ 54,  54,  54],
         [ 54,  54,  54],
         [ 54,  54,  54]],
```

```python
# calculating the  inception score for evaluating the Augmented images/Original Images used in generating GAN images
from math import floor
from numpy import ones
from numpy import expand_dims
from numpy import log
from numpy import mean
from numpy import std
from numpy import exp
from numpy.random import shuffle
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input
from skimage.transform import resize
from numpy import asarray

# scale an array of images to a new size
def scale_images(trainX, new_shape):
  images_list = list()
  for image in trainX:
    # resize with nearest neighbor interpolation
    new_image = resize(image, new_shape, 0)
    # store
    images_list.append(new_image)
  return asarray(images_list)

# assumes images have any shape and pixels in [0,255]
def calculate_inception_score(images, n_split=10, eps=1E-16):
  # load inception v3 model
  model = InceptionV3()
  # enumerate splits of images/predictions
  scores = list()
  n_part = floor(images.shape[0] / n_split)
  for i in range(n_split):
    # retrieve images
    ix_start, ix_end = i * n_part, (i+1) * n_part
    subset = images[ix_start:ix_end]
```

```
        # convert from uint8 to float32
        subset = subset.astype('float32')
        # scale images to the required size
        subset = scale_images(subset, (299,299,3))
        # pre-process images, scale to [-1,1]
        subset = preprocess_input(subset)
        # predict p(y|x)
        p_yx = model.predict(subset)
        # calculate p(y)
        p_y = expand_dims(p_yx.mean(axis=0), 0)
        # calculate KL divergence using log probabilities
        kl_d = p_yx * (log(p_yx + eps) - log(p_y + eps))
        # sum over classes
        sum_kl_d = kl_d.sum(axis=1)
        # average over images
        avg_kl_d = mean(sum_kl_d)
        # undo the log
        is_score = exp(avg_kl_d)
        # store
        scores.append(is_score)
    # average across images
    is_avg, is_std = mean(scores), std(scores)
    return is_avg, is_std

dataset = (trainX, _), (_, _)
# shuffle the images
shuffle(trainX)
print('loaded', trainX.shape)
# calculate inception score
is_avg, is_std = calculate_inception_score(trainX)
print('score', is_avg, is_std)
```

```
loaded (4000, 32, 32, 3)
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96116736/96112376 [==============================] - 0s 0us/step
96124928/96112376 [==============================] - 0s 0us/step
score 3.109748 0.11674382
```

```
# loading the GAN generated Images for calculating the Inception Score (IS)
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
#DATADIR = "C:\\Users\\iffyd\\Masters Project\\erythrocytes"
DATADIR = "/content/drive/MyDrive/GanImages1/"

CATEGORIES = [ "circular", "elongated", "other" ]

for category in CATEGORIES:  # elongated, circular and other
    path = os.path.join(DATADIR,category)  # create path to circular, elongated or other
    for img in os.listdir(path):  # iterate over each image for circular, elongated or other
        img_array = cv2.imread(os.path.join(path,img))
        img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB) # convert bgr to rgb
        img_array = cv2.resize(img_array, (32,32))
        plt.imshow(img_array)  # graph it
        plt.show()  # display!

        break  # we just want one for now so break

    break  #...and one more!
```



```python
training_data = []

def create_training_data():
    for category in CATEGORIES:  # for the classes (circular, elongated or other)

        path = os.path.join(DATADIR,category)  # create path to circular, elongated or other
        class_num = CATEGORIES.index(category)  # get the classification  (0, 1 or 2). 0=circular, 1=elongated, 2=other

        for img in tqdm(os.listdir(path)):  # iterate over each image for circular, elongated or other
            try:
                img_array = cv2.imread(os.path.join(path,img))  # convert to array
                img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB) # convert bgr to rgb
                new_array = cv2.resize(img_array, (32, 32))  # resize images
                training_data.append([new_array, class_num])  # add this to our training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()

print(len(training_data))
```

```
100%|██████████| 2000/2000 [00:16<00:00, 122.63it/s]
100%|██████████| 2000/2000 [00:16<00:00, 122.44it/s]
100%|██████████| 2000/2000 [00:16<00:00, 123.47it/s]6000
```

```python
###Loading the data and shuffling to ensure each split covers a diverse set of classes.
# load the erythrocytes images
dataset = (trainX, _), (_, _)
# shuffle the images
shuffle(trainX)
```

```python
# calculating the  inception score for evaluating GAN generated images
from math import floor
from numpy import ones
from numpy import expand_dims
from numpy import log
from numpy import mean
from numpy import std
from numpy import exp
from numpy.random import shuffle
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input
from skimage.transform import resize
from numpy import asarray

# scale an array of images to a new size
def scale_images(trainX, new_shape):
  images_list = list()
  for image in trainX:
    # resize with nearest neighbor interpolation
    new_image = resize(image, new_shape, 0)
    # store
    images_list.append(new_image)
  return asarray(images_list)

# assumes images have any shape and pixels in [0,255]
def calculate_inception_score(images, n_split=10, eps=1E-16):
  # load inception v3 model
  model = InceptionV3()
  # enumerate splits of images/predictions
  scores = list()
  n_part = floor(images.shape[0] / n_split)
  for i in range(n_split):
    # retrieve images
    ix_start, ix_end = i * n_part, (i+1) * n_part
    subset = images[ix_start:ix_end]
```

```
        # convert from uint8 to float32
        subset = subset.astype('float32')
        # scale images to the required size
        subset = scale_images(subset, (299,299,3))
        # pre-process images, scale to [-1,1]
        subset = preprocess_input(subset)
        # predict p(y|x)
        p_yx = model.predict(subset)
        # calculate p(y)
        p_y = expand_dims(p_yx.mean(axis=0), 0)
        # calculate KL divergence using log probabilities
        kl_d = p_yx * (log(p_yx + eps) - log(p_y + eps))
        # sum over classes
        sum_kl_d = kl_d.sum(axis=1)
        # average over images
        avg_kl_d = mean(sum_kl_d)
        # undo the log
        is_score = exp(avg_kl_d)
        # store
        scores.append(is_score)
    # average across images
    is_avg, is_std = mean(scores), std(scores)
    return is_avg, is_std

dataset = (trainX, _), (_, _)
# shuffle the images
shuffle(trainX)
print('loaded', trainX.shape)
# calculate inception score
is_avg, is_std = calculate_inception_score(trainX)
print('score', is_avg, is_std)
```

```
loaded (4800, 32, 32, 3)
score 1.8279059 0.03621714
```

9 Implementing The Classification Models

The six deep learning models were used in the modelling of each of the three image datasets (**erythrocytes, erythrocytes2 and erythrocytes3**). The images were processed by resizing to the input shape of each model and then converted to numpy array and normalized to between 0 and 1 for all the models. Each model was fine-tuned based on the specification design, and was implemented using Keras API function, TensorFlow and Python

```
from google.colab import drive
drive.mount('/content/gdrive')

```

```
Mounted at /content/gdrive
```

```
## Import the required libraries
from numpy.random import seed
seed(42)
import tensorflow
tensorflow.random.set_seed(42)
seed = 42
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from tqdm import tqdm
from PIL import Image
from sklearn.utils import shuffle
import os
import glob
import keras
import random
import math
import seaborn as sns
import sys
import datetime
import glob as glob
import matplotlib.cm as cm
from sklearn import metrics
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import applications
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Dropout, Flatten,GlobalAveragePooling2D
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Lambda
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.applications.densenet import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,img_to_array, load_img
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping, Callback

import warnings
warnings.filterwarnings("ignore")
```

```
import tensorflow as tf
print(tf.__version__)
```

```
2.8.2
```

9.1 Modelling The Original Images

Each model takes between 2 to 4 minutes to run

DENSENET121 MODEL

```
model_d=DenseNet121(weights='imagenet',include_top=False, input_shape=(224, 224, 3))

x=model_d.output

x= GlobalAveragePooling2D()(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
x= Dense(1024,activation='relu')(x)
x= Dense(512,activation='relu')(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)

preds=Dense(3,activation='softmax')(x) #FC-layer
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121
29089792/29084464 [==============================] - 0s 0us/step
29097984/29084464 [==============================] - 0s 0us/step
```

```
[ ]
model=Model(inputs=model_d.input,outputs=preds)
model.summary()
```

```
conv5_block15_1_bn (BatchNorma  (None, 7, 7, 128)   512        ['conv5_block15_1_conv[0][0]']
lization)

conv5_block15_1_relu (Activati  (None, 7, 7, 128)   0          ['conv5_block15_1_bn[0][0]']
```

```
for layer in model.layers[:-8]:
    layer.trainable=False

for layer in model.layers[-8:]:
    layer.trainable=True
```

```
data=[]
labels=[]
random.seed(42)
imagePaths = sorted(list(os.listdir("/content/drive/MyDrive/erythrocytes/")))
random.shuffle(imagePaths)
print(imagePaths)
#/content/drive/MyDrive/augment1erythrocytes
for img in imagePaths:
    path=sorted(list(os.listdir("/content/drive/MyDrive/erythrocytes/"+img)))
    for i in path:
        image = cv2.imread("/content/drive/MyDrive/erythrocytes/"+img+'/'+i)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, (224,224))
        image = img_to_array(image)
        data.append(image)
        l = label = img
        labels.append(l)
```

```
['elongated', 'circular', 'other']
```

```
optimizer = Adam(learning_rate=0.0002)
#early stopping to monitor the validation loss and avoid overfitting
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5, restore_best_weights=True)

#reducing learning rate on plateau
#rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience= 5, factor= 0.5, min_lr= 1e-6, verbose=1)
```

```
#model compiling
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=30, horizontal_flip=True, vertical_flip=True, shear_range=0.2,
#datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horzontal_flip=False, shear_range=0.0)



datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
3/3 - 4s - loss: 0.7071 - accuracy: 0.7460 - val_loss: 0.6264 - val_accuracy: 0.8772 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 6/20

Epoch 6: val_loss improved from 0.62642 to 0.57655, saving model to model.h5
3/3 - 4s - loss: 0.5350 - accuracy: 0.7989 - val_loss: 0.5765 - val_accuracy: 0.8947 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 7/20
```

```
ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 63     accurately-predicted-data: 54     wrongly-predicted-data:  9
Accuracy: 85.714 %
```

INCEPTION V3 MODEL

```
[ ] InceptionV3_model = tf.keras.applications.InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 4s 0us/step
87924736/87910968 [==============================] - 4s 0us/step
```

```
[ ] from tensorflow.keras import Model
from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Dropout, Flatten,GlobalAveragePooling2D
from tensorflow.keras.models import Sequential

# The last 15 layers fine tune
for layer in InceptionV3_model.layers[:-15]:
    layer.trainable = False

num_class = 3
# Base model without Fully connected Layers
base_model = InceptionV3(include_top=False, weights='imagenet', input_shape=(224,224,3))
x=base_model.output
# Add some new Fully connected layers to
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x = Dropout(0.25)(x)
x=Dense(512,activation='relu')(x)
x = Dropout(0.25)(x)
preds=Dense(num_class, activation='softmax')(x) #final layer with softmax activation

model=Model(inputs=base_model.input,outputs=preds)
model.summary()
```

```
 Normalization)

 batch_normalization_187 (Batch  (None, 5, 5, 384)   1152        ['conv2d_185[0][0]']
```

```
optimizer = Adam(learning_rate=0.0002)
#early stopping to monitor the validation loss and avoid overfitting
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10, restore_best_weights=True)

#reducing learning rate on plateau
#rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience= 5, factor= 0.5, min_lr= 1e-6, verbose=1)
```

```
#model compiling
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=30, horizontal_flip=True, vertical_flip=True, shear_range=0.2,height_shift_range=0.2,width_shift_range=0.2
#datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)


datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
3/3 - 5s - loss: 0.1863 - accuracy: 0.9259 - val_loss: 0.4536 - val_accuracy: 0.8947 - lr: 2.0000e-04 - 5s/epoch - 2s/step
Epoch 6/20

Epoch 6: val_loss did not improve from 0.45360
3/3 - 4s - loss: 0.1087 - accuracy: 0.9656 - val_loss: 0.5921 - val_accuracy: 0.8772 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 7/20

Epoch 7: val_loss did not improve from 0.45360
3/3 - 4s - loss: 0.1674 - accuracy: 0.9312 - val_loss: 0.6400 - val_accuracy: 0.8947 - lr: 2.0000e-04 - 4s/epoch - 1s/step
```

```
ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 63      accurately-predicted-data: 60     wrongly-predicted-data:  3
Accuracy: 95.238 %
```

MOBILENET MODEL

```
[ ]  mobile = tf.keras.applications.mobilenet.MobileNet()
     mobile.summary()
```

```
conv_dw_11_relu (ReLU)       (None, 14, 14, 512)        0

conv_pw_11 (Conv2D)          (None, 14, 14, 512)        262144

conv_pw_11_bn (BatchNormali  (None, 14, 14, 512)        2048
zation)

conv_pw_11_relu (ReLU)       (None, 14, 14, 512)        0

conv_pad_12 (ZeroPadding2D)  (None, 15, 15, 512)        0

conv_dw_12 (DepthwiseConv2D  (None, 7, 7, 512)          4608
)

conv_dw_12_bn (BatchNormali  (None, 7, 7, 512)          2048
```

```
base_model=MobileNet(weights='imagenet',include_top=False, input_shape=(224, 224, 3))  #imports the mobilenet model and discards the last 1000 neuron layer.

```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf_no_top.h5
17227776/17225924 [==============================] - 0s 0us/step
17235968/17225924 [==============================] - 0s 0us/step
```

```
x=base_model.output

x= GlobalAveragePooling2D()(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
x= Dense(1024,activation='relu')(x)
x= Dense(1024,activation='relu')(x)
x= Dense(512,activation='relu')(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)

preds=Dense(3,activation='softmax')(x) #FC-layer
```

```
model=Model(inputs=base_model.input,outputs=preds)
model.summary()
```

```
conv_pad_12 (ZeroPadding2D)  (None, 15, 15, 512)      0

conv_dw_12 (DepthwiseConv2D  (None, 7, 7, 512)        4608
)

conv_dw_12_bn (BatchNormali  (None, 7, 7, 512)        2048
zation)

conv_dw_12_relu (ReLU)      (None, 7, 7, 512)         0
```

```
model.compile(optimizer=Adam(learning_rate=0.0002), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=30, horizontal_flip=True, vertical_flip=True, shear_range=0.2,height_shift_range=0.2,width_shift_range=0.2
#datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)


datagen.fit(xtrain)
# Fits-the-model
history = model.fit(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 6: val_loss did not improve from 0.74408
3/3 - 3s - loss: 0.5435 - accuracy: 0.7963 - val_loss: 0.7604 - val_accuracy: 0.6667 - lr: 2.0000e-04 - 3s/epoch - 1s/step
Epoch 7/20

Epoch 7: val_loss did not improve from 0.74408
3/3 - 4s - loss: 0.5038 - accuracy: 0.8280 - val_loss: 0.7785 - val_accuracy: 0.6491 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 8/20
```

```
##The test data is used to predict the performance of the model on unseen data and the correct prediction and wrong prediction are collected in a list with test accur

ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 63     accurately-predicted-data: 39     wrongly-predicted-data:   24
Accuracy: 61.905 %
```

VGG16 MODEL

```
[ ]
from tensorflow.keras.applications import VGG16 #For Transfer Learning
```

```
##Building Model
IMAGE_SIZE = [224, 224]
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
#here [3] denotes for RGB images(3 channels)

#don't train existing weights
for layer in vgg.layers:
 layer.trainable = False

x = Flatten()(vgg.output)
prediction = Dense(3, activation='softmax')(x)
model = Model(inputs=vgg.input, outputs=prediction)

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
58900480/58889256 [==============================] - 1s 0us/step
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=30, horizontal_flip=True, vertical_flip=True, shear_range=0.2,height_shift_range=0.2,width_shift_range=0.2
#datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=2, horizontal_flip=True, shear_range=0.2)


datagen.fit(xtrain)
# Fits-the-model
history = model.fit(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 6: val_loss improved from 1.05558 to 1.05403, saving model to model.h5
3/3 - 4s - loss: 1.0734 - accuracy: 0.4021 - val_loss: 1.0540 - val_accuracy: 0.3684 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 7/20

Epoch 7: val_loss did not improve from 1.05403
3/3 - 4s - loss: 1.0617 - accuracy: 0.4312 - val_loss: 1.0579 - val_accuracy: 0.3684 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 8/20

Epoch 8: val_loss improved from 1.05403 to 1.03464, saving model to model.h5
3/3 - 4s - loss: 1.0535 - accuracy: 0.3836 - val_loss: 1.0346 - val_accuracy: 0.4386 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 9/20

Epoch 9: val_loss improved from 1.03464 to 1.01180, saving model to model.h5
3/3 - 4s - loss: 1.0229 - accuracy: 0.4815 - val_loss: 1.0118 - val_accuracy: 0.6140 - lr: 2.0000e-04 - 4s/epoch - 1s/step
```

```
##The test data is used to predict the performance of the model on unseen data and the correct prediction and wrong prediction are collected in a list with test accurac

ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 63     accurately-predicted-data: 31     wrongly-predicted-data:  32
Accuracy: 49.206 %
```

VGG19 MODEL

```
[ ]
# Base model without Fully connected Layers
VGG_model = VGG19(include_top=False, weights='imagenet', input_shape=(224,224,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 0s 0us/step
80150528/80134624 [==============================] - 0s 0us/step
```

```
[ ]
VGG_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
[ ]
# dataset has 3 classes
num_class = 3
x=VGG_model.output
# Add some new Fully connected layers to
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x = Dropout(0.25)(x)
x=Dense(512,activation='relu')(x)
x = Dropout(0.25)(x)
preds=Dense(num_class, activation='softmax')(x) #final layer with softmax activation

model=Model(inputs=VGG_model.input,outputs=preds)
```

```
[ ]
for layer in VGG_model.layers:
  layer.trainable = False

VGG_model.summary()  #Trainable parameters will be 0
```

```
Model: "vgg19"

 Layer (type)                Output Shape              Param #
```

```
anne = 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)

#datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)
datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=30, horizontal_flip=True, vertical_flip=True, shear_range=0.2,height_shift_range=0.2,width_shift_range=0.2

datagen.fit(xtrain)
# Fits-the-model
history = model.fit(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 6: val_loss improved from 1.04612 to 1.01765, saving model to model.h5
3/3 - 5s - loss: 1.0548 - accuracy: 0.4471 - val_loss: 1.0177 - val_accuracy: 0.5614 - lr: 2.0000e-04 - 5s/epoch - 2s/step
Epoch 7/20

Epoch 7: val_loss improved from 1.01765 to 0.98702, saving model to model.h5
3/3 - 5s - loss: 1.0206 - accuracy: 0.4815 - val_loss: 0.9870 - val_accuracy: 0.5789 - lr: 2.0000e-04 - 5s/epoch - 2s/step
Epoch 8/20

Epoch 8: val_loss improved from 0.98702 to 0.96826, saving model to model.h5
3/3 - 5s - loss: 1.0035 - accuracy: 0.4444 - val_loss: 0.9683 - val_accuracy: 0.4386 - lr: 2.0000e-04 - 5s/epoch - 2s/step
Epoch 9/20
```

```
##The test data is used to predict the performance of the model on unseen data and the correct prediction and wrong prediction are collected in a list with test accurac

ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 63	accurately-predicted-data: 35	wrongly-predicted-data:  28
Accuracy: 55.556 %
```

RESNET50 MODEL

```
base_model = applications.resnet50.ResNet50(weights= None, include_top=False, input_shape= (224,224,3))
```

```
x=base_model.output

x= GlobalAveragePooling2D()(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
x= Dense(1024,activation='relu')(x)
x= Dense(1024,activation='relu')(x)
x= Dense(512,activation='relu')(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)

preds=Dense(3,activation='softmax')(x)
```

```
model=Model(inputs=base_model.input,outputs=preds)
model.summary()
```

```
conv5_block2_add (Add)          (None, 7, 7, 2048)   0          ['conv5_block1_out[0][0]',
                                                                 'conv5_block2_3_bn[0][0]']

conv5_block2_out (Activation)   (None, 7, 7, 2048)   0          ['conv5_block2_add[0][0]']

conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)    1049088    ['conv5_block2_out[0][0]']
```

```
for layer in model.layers[:-8]:
    layer.trainable=False

for layer in model.layers[-8:]:
    layer.trainable=True
```

```
model.compile(optimizer=Adam(learning_rate=0.0002),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
batch_size = 128
epochs = 20
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=30, horizontal_flip=True, vertical_flip=True, shear_range=0.2,height_shift_range=0.2,width_shift_range=0.2
#datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)


datagen.fit(xtrain)
# Fits-the-model
history = model.fit(datagen.flow(xtrain, ytrain, batch_size=batch_size),
               steps_per_epoch=xtrain.shape[0] //batch_size,
               epochs=epochs,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 6: val_loss did not improve from 1.09823
3/3 - 4s - loss: 1.4010 - accuracy: 0.3651 - val_loss: 1.0990 - val_accuracy: 0.3158 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 7/20

Epoch 7: val_loss did not improve from 1.09823
3/3 - 4s - loss: 1.4320 - accuracy: 0.3386 - val_loss: 1.0984 - val_accuracy: 0.3158 - lr: 2.0000e-04 - 4s/epoch - 1s/step
Epoch 8/20
```

```
ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 63	accurately-predicted-data: 24	 wrongly-predicted-data:  39
Accuracy: 38.095 %
```

9.1.1 Modelling Traditional Augmented Images

Each model takes between 2 to 6 minutes to run

DENSENET121 MODEL

```
model_d=DenseNet121(weights='imagenet',include_top=False, input_shape=(224, 224, 3))

x=model_d.output

x= GlobalAveragePooling2D()(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
x= Dense(1024,activation='relu')(x)
x= Dense(512,activation='relu')(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)

preds=Dense(3,activation='softmax')(x) #FC-layer
```

```
model=Model(inputs=model_d.input,outputs=preds)
model.summary()
```

```
conv5_block15_1_bn (BatchNorma  (None, 7, 7, 128)   512        ['conv5_block15_1_conv[0][0]']
lization)

conv5_block15_1_relu (Activati  (None, 7, 7, 128)   0          ['conv5_block15_1_bn[0][0]']
on)

conv5_block15_2_conv (Conv2D)  (None, 7, 7, 32)    36864      ['conv5_block15_1_relu[0][0]']

conv5_block15_concat (Concaten  (None, 7, 7, 992)   0          ['conv5_block14_concat[0][0]',
ate)                                                            'conv5_block15_2_conv[0][0]']
```

```
optimizer = Adam(learning_rate=0.0002)
#early stopping to monitor the validation loss and avoid overfitting
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5, restore_best_weights=True)

#reducing learning rate on plateau
#rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience= 5, factor= 0.5, min_lr= 1e-6, verbose=1)
```

```
#model compiling
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)


datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[es,anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 6: val_loss improved from 0.24179 to 0.23312, saving model to model.h5
31/31 - 7s - loss: 0.2828 - accuracy: 0.8993 - val_loss: 0.2331 - val_accuracy: 0.9156 - lr: 2.0000e-04 - 7s/epoch - 239ms/step
Epoch 7/20

Epoch 7: val_loss improved from 0.23312 to 0.22519, saving model to model.h5
31/31 - 7s - loss: 0.2415 - accuracy: 0.9057 - val_loss: 0.2252 - val_accuracy: 0.9133 - lr: 2.0000e-04 - 7s/epoch - 239ms/step
Epoch 8/20
```

```
ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 501     accurately-predicted-data: 470    wrongly-predicted-data:  31
Accuracy: 93.812 %
```

INCEPTION V3 MODEL

```
[ ]  InceptionV3_model = tf.keras.applications.InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
[ ]  from tensorflow.keras import Model
from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Dropout, Flatten,GlobalAveragePooling2D
from tensorflow.keras.models import Sequential

# The last 15 layers fine tune
for layer in InceptionV3_model.layers[:-15]:
    layer.trainable = False

num_class = 3
# Base model without Fully connected Layers
base_model = InceptionV3(include_top=False, weights='imagenet', input_shape=(224,224,3))
x=base_model.output
# Add some new Fully connected layers to
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x = Dropout(0.25)(x)
x=Dense(512,activation='relu')(x)
x = Dropout(0.25)(x)
preds=Dense(num_class, activation='softmax')(x) #final layer with softmax activation

model=Model(inputs=base_model.input,outputs=preds)
model.summary()
```

```
Normalization)

batch_normalization_571 (Batch  (None, 5, 5, 384)   1152        ['conv2d_561[0][0]']
Normalization)
```

```
optimizer = Adam(learning_rate=0.0002)
#early stopping to monitor the validation loss and avoid overfitting
#es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10, restore_best_weights=True)

#reducing learning rate on plateau
#rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience= 5, factor= 0.5, min_lr= 1e-6, verbose=1)
```

```
#model compiling
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)
#datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=2, horizontal_flip=True, shear_range=0.2)

datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
                steps_per_epoch=xtrain.shape[0] //128,
                epochs=20,
                verbose=2,
                callbacks=[anne, checkpoint],
                validation_data=(xval, yval))
```

```
Epoch 6: val_loss improved from 0.16079 to 0.08826, saving model to model.h5
31/31 - 16s - loss: 0.0080 - accuracy: 0.9972 - val_loss: 0.0883 - val_accuracy: 0.9822 - lr: 2.0000e-04 - 16s/epoch - 524ms/step
Epoch 7/20

Epoch 7: val_loss did not improve from 0.08826
31/31 - 15s - loss: 0.0078 - accuracy: 0.9972 - val_loss: 0.1241 - val_accuracy: 0.9800 - lr: 2.0000e-04 - 15s/epoch - 481ms/step
Epoch 8/20
```

```
ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 501     accurately-predicted-data: 492    wrongly-predicted-data:  9
Accuracy: 98.204 %
```

MOBILENET MODEL

```
[ ]
mobile = tf.keras.applications.mobilenet.MobileNet()
mobile.summary()
```

```
conv_dw_11_relu (ReLU)      (None, 14, 14, 512)       0

conv_pw_11 (Conv2D)         (None, 14, 14, 512)       262144

conv_pw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)

conv_pw_11_relu (ReLU)      (None, 14, 14, 512)       0

conv_pad_12 (ZeroPadding2D)  (None, 15, 15, 512)      0

conv_dw_12 (DepthwiseConv2D  (None, 7, 7, 512)        4608
)
```

```
base_model=MobileNet(weights='imagenet',include_top=False, input_shape=(224, 224, 3))  #imports the mobilenet model and discards the last 1000 neuron layer.
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf_no_top.h5
17227776/17225924 [==============================] - 0s 0us/step
17235968/17225924 [==============================] - 0s 0us/step
```

```
x=base_model.output

x= GlobalAveragePooling2D()(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
x= Dense(1024,activation='relu')(x)
x= Dense(1024,activation='relu')(x)
x= Dense(512,activation='relu')(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)

preds=Dense(3,activation='softmax')(x) #FC-layer
```

```
model=Model(inputs=base_model.input,outputs=preds)
model.summary()
```

```
conv_pad_12 (ZeroPadding2D)  (None, 15, 15, 512)      0

conv_dw_12 (DepthwiseConv2D  (None, 7, 7, 512)        4608
)

conv_dw_12_bn (BatchNormali  (None, 7, 7, 512)        2048
zation)
```

```python
optimizer = Adam(learning_rate=0.0002)
#early stopping to monitor the validation loss and avoid overfitting
#es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10, restore_best_weights=True)

#reducing learning rate on plateau
#rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience= 5, factor= 0.5, min_lr= 1e-6, verbose=1)
```

```python
#model compiling
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)
#datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=2, horizontal_flip=True, shear_range=0.2)

datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 6/20

Epoch 6: val_loss did not improve from 0.06902
31/31 - 15s - loss: 0.0064 - accuracy: 0.9980 - val_loss: 0.0702 - val_accuracy: 0.9889 - lr: 2.0000e-04 - 15s/epoch - 484ms/step
Epoch 7/20

Epoch 7: val_loss did not improve from 0.06902
31/31 - 15s - loss: 0.0093 - accuracy: 0.9969 - val_loss: 0.1054 - val_accuracy: 0.9800 - lr: 2.0000e-04 - 15s/epoch - 482ms/step
Epoch 8/20
```

```python
##The test data is used to predict the performance of the model on unseen data and the correct prediction and wrong prediction are collected in a list with test accuracy score

ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 501    accurately-predicted-data: 489   wrongly-predicted-data:  12
Accuracy: 97.605 %
```

VGG16 MODEL

```
[ ]  from tensorflow.keras.applications import VGG16 #For Transfer Learning
```

```
[ ]  ##Building Model
IMAGE_SIZE = [224, 224]
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
#here [3] denotes for RGB images(3 channels)

#don't train existing weights
for layer in vgg.layers:
 layer.trainable = False

x = Flatten()(vgg.output)
prediction = Dense(3, activation='softmax')(x)
model = Model(inputs=vgg.input, outputs=prediction)

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)


datagen.fit(xtrain)
# Fits-the-model
history = model.fit(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[es,anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 6: val_loss improved from 0.57048 to 0.54704, saving model to model.h5
31/31 - 10s - loss: 0.5354 - accuracy: 0.8322 - val_loss: 0.5470 - val_accuracy: 0.8311 - lr: 1.0000e-04 - 10s/epoch - 324ms/step
Epoch 7/20

Epoch 7: val_loss improved from 0.54704 to 0.52931, saving model to model.h5
31/31 - 10s - loss: 0.5050 - accuracy: 0.8417 - val_loss: 0.5293 - val_accuracy: 0.7956 - lr: 1.0000e-04 - 10s/epoch - 325ms/step
Epoch 8/20

Epoch 8: val_loss improved from 0.52931 to 0.50517, saving model to model.h5
31/31 - 10s - loss: 0.4835 - accuracy: 0.8460 - val_loss: 0.5052 - val_accuracy: 0.8111 - lr: 1.0000e-04 - 10s/epoch - 326ms/step
Epoch 9/20

Epoch 9: val_loss improved from 0.50517 to 0.49254, saving model to model.h5
31/31 - 10s - loss: 0.4597 - accuracy: 0.8536 - val_loss: 0.4925 - val_accuracy: 0.8378 - lr: 1.0000e-04 - 10s/epoch - 324ms/step
Epoch 10/20

Epoch 10: val_loss improved from 0.49254 to 0.47931, saving model to model.h5
```

```
##The test data is used to predict the performance of the model on unseen data and the correct prediction and wrong prediction are collected in a list with test accuracy score

ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 501    accurately-predicted-data: 423    wrongly-predicted-data:  78
Accuracy: 84.431 %
```

VGG19 MODEL

```
[ ] VGG_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 0s 0us/step
80150528/80134624 [==============================] - 0s 0us/step
```

```
[ ] for layer in VGG_model.layers:
      layer.trainable = False

    VGG_model.summary()  #Trainable parameters will be 0

Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)


datagen.fit(xtrain)
# Fits-the-model
history = model.fit(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[es,anne, checkpoint],
               validation_data=(xval, yval))
Epoch 7/20

Epoch 7: val_loss did not improve from 0.22278
31/31 - 12s - loss: 0.1828 - accuracy: 0.9276 - val_loss: 0.2243 - val_accuracy: 0.8889 - lr: 1.0000e-04 - 12s/epoch - 392ms/step
Epoch 8/20

Epoch 8: val_loss improved from 0.22278 to 0.17943, saving model to model.h5
31/31 - 12s - loss: 0.1568 - accuracy: 0.9360 - val_loss: 0.1794 - val_accuracy: 0.9289 - lr: 1.0000e-04 - 12s/epoch - 402ms/step
Epoch 9/20

Epoch 9: val_loss did not improve from 0.17943
```

```
##The test data is used to predict the performance of the model on unseen data and the correct prediction and wrong prediction are collected in a list with test accuracy score

ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')

Total-test-data; 501    accurately-predicted-data: 462    wrongly-predicted-data:  39
Accuracy: 92.216 %
```

RESNET50 MODEL

```
[ ]  base_model = applications.resnet50.ResNet50(weights= None, include_top=False, input_shape= (224,224,3))
```

```
[ ]  x=base_model.output

     x= GlobalAveragePooling2D()(x)
     x= BatchNormalization()(x)
     x= Dropout(0.5)(x)
     x= Dense(1024,activation='relu')(x)
     x= Dense(1024,activation='relu')(x)
     x= Dense(512,activation='relu')(x)
     x= BatchNormalization()(x)
     x= Dropout(0.5)(x)

     preds=Dense(3,activation='softmax')(x)
```

```
[ ]  model=Model(inputs=base_model.input,outputs=preds)
     model.summary()
```

```
conv5_block2_add (Add)          (None, 7, 7, 2048)   0          ['conv5_block1_out[0][0]',
                                                                  'conv5_block2_3_bn[0][0]']

conv5_block2_out (Activation)   (None, 7, 7, 2048)   0          ['conv5_block2_add[0][0]']

conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)    1049088    ['conv5_block2_out[0][0]']

conv5_block3_1_bn (BatchNormal  (None, 7, 7, 512)    2048       ['conv5_block3_1_conv[0][0]']
```

```
==================================================================================================
Total params: 27,272,067
Trainable params: 27,213,827
Non-trainable params: 58,240
__________________________________________________________________________________________________
```

```
]  for layer in model.layers[:-8]:
       layer.trainable=False

   for layer in model.layers[-8:]:
       layer.trainable=True
```

```
]  model.compile(optimizer=Adam(learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
]  batch_size = 128
   epochs = 20
```

```
]  anne = 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)

   datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)


   datagen.fit(xtrain)
   # Fits-the-model
   history = model.fit(datagen.flow(xtrain, ytrain, batch_size=batch_size),
                  steps_per_epoch=xtrain.shape[0] //batch_size,
                  epochs=epochs,
                  verbose=2,
                  callbacks=[anne, checkpoint],
                  validation_data=(xval, yval))
```

```
Epoch 6: val_loss improved from 1.09037 to 1.08739, saving model to model.h5
31/31 - 8s - loss: 0.9796 - accuracy: 0.5686 - val_loss: 1.0874 - val_accuracy: 0.4222 - lr: 1.0000e-04 - 8s/epoch - 250ms/step
```

```
ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 501     accurately-predicted-data: 344    wrongly-predicted-data:  157
Accuracy: 68.663 %
```

9.1.2 GAN Generated Images/Original Images

Each model took between 4mins to 6mins to run.
Result of Modelling GAN Generated Images/Original Images Dataset

```
## Import the required libraries
from numpy.random import seed
seed(42)
import tensorflow
tensorflow.random.set_seed(42)
seed = 42
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from tqdm import tqdm
from PIL import Image
from sklearn.utils import shuffle
import os
import glob
import keras
import random
import math
import seaborn as sns
import sys
import datetime
import glob as glob
import matplotlib.cm as cm
from sklearn import metrics
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import applications
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Dropout, Flatten,GlobalAveragePooling2D
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Lambda
```

```
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications import EfficientNetB5
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.applications.densenet import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,img_to_array, load_img
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping, Callback

import warnings
warnings.filterwarnings("ignore")
```

```
import tensorflow as tf
print(tf.__version__)
```

```
2.8.2
```

DENSENET121 MODEL

```
model_d=DenseNet121(weights='imagenet',include_top=False, input_shape=(224, 224, 3))

x=model_d.output

x= GlobalAveragePooling2D()(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)
x= Dense(1024,activation='relu')(x)
x= Dense(512,activation='relu')(x)
x= BatchNormalization()(x)
x= Dropout(0.5)(x)

preds=Dense(3,activation='softmax')(x) #FC-layer
```

```
[ ]
model=Model(inputs=model_d.input,outputs=preds)
model.summary()
```

```
Model: "model_7"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 input_12 (InputLayer)          [(None, 224, 224, 3  0           []
                                )]

 zero_padding2d_2 (ZeroPadding2  (None, 230, 230, 3)  0          ['input_12[0][0]']
 D)
```

```
optimizer = Adam(learning_rate=0.0002)
#early stopping to monitor the validation loss and avoid overfitting
#es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5, restore_best_weights=True)

#reducing learning rate on plateau
#rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience= 5, factor= 0.5, min_lr= 1e-6, verbose=1)
```

```
#model compiling
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
anne = 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)

datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)


datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 1/20

Epoch 1: val_loss improved from inf to 0.38326, saving model to model.h5
41/41 - 18s - loss: 0.4138 - accuracy: 0.8506 - val_loss: 0.3833 - val_accuracy: 0.9330 - lr: 2.0000e-04 - 18s/epoch - 427ms/step
Epoch 2/20
```

```
ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 663     accurately-predicted-data: 656    wrongly-predicted-data:   7
Accuracy: 98.944 %
```

RESNET50 MODEL

```
[ ]
```

```
[ ]  base_model = applications.resnet50.ResNet50(weights= None, include_top=False, input_shape= (224,224,3))
```

```
[ ]  x=base_model.output

     x= GlobalAveragePooling2D()(x)
     x= BatchNormalization()(x)
     x= Dropout(0.5)(x)
     x= Dense(1024,activation='relu')(x)
     x= Dense(1024,activation='relu')(x)
     x= Dense(512,activation='relu')(x)
     x= BatchNormalization()(x)
     x= Dropout(0.5)(x)

     preds=Dense(3,activation='softmax')(x)
```

```
[ ]  model=Model(inputs=base_model.input,outputs=preds)
     model.summary()
```

```
Model: "model_5"

 Layer (type)                  Output Shape         Param #     Connected to
==================================================================================================
 input_9 (InputLayer)          [(None, 224, 224, 3  0           []
                               )]

 conv1_pad (ZeroPadding2D)     (None, 230, 230, 3)  0           ['input_9[0][0]']
```

```
for layer in model.layers[:-8]:
    layer.trainable=False

for layer in model.layers[-8:]:
    layer.trainable=True
```

```
model.compile(optimizer=Adam(learning_rate=0.0002),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
batch_size = 128
epochs = 20
```

```
anne = 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)

#datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=2, horizontal_flip=True, shear_range=0.2)
datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)

datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 1/20

Epoch 1: val_loss improved from inf to 1.09659, saving model to model.h5
41/41 - 16s - loss: 1.2696 - accuracy: 0.4845 - val_loss: 1.0966 - val_accuracy: 0.3333 - lr: 2.0000e-04 - 16s/epoch - 383ms/step
Epoch 2/20

Epoch 2: val_loss improved from 1.09659 to 1.09069, saving model to model.h5
```

INCEPTION V3 MODEL

```
[ ]  InceptionV3_model = tf.keras.applications.InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
[ ]  from tensorflow.keras import Model
     from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Dropout, Flatten,GlobalAveragePooling2D
     from tensorflow.keras.models import Sequential

     # The last 15 layers fine tune
     for layer in InceptionV3_model.layers[:-15]:
         layer.trainable = False

     num_class = 3
     # Base model without Fully connected Layers
     base_model = InceptionV3(include_top=False, weights='imagenet', input_shape=(224,224,3))
     x=base_model.output
     # Add some new Fully connected layers to
     x=GlobalAveragePooling2D()(x)
     x=Dense(1024,activation='relu')(x)
     x = Dropout(0.25)(x)
     x=Dense(512,activation='relu')(x)
     x = Dropout(0.25)(x)
     preds=Dense(num_class, activation='softmax')(x) #final layer with softmax activation

     model=Model(inputs=base_model.input,outputs=preds)
     model.summary()
```

```
Model: "model_6"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_11 (InputLayer)          [(None, 224, 224, 3  0           []
                                )]
```

```
anne = 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)

#datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=2, horizontal_flip=True, shear_range=0.2)
datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)

datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 1/20

Epoch 1: val_loss improved from inf to 1.69041, saving model to model.h5
41/41 - 29s - loss: 0.1846 - accuracy: 0.9219 - val_loss: 1.6904 - val_accuracy: 0.7504 - lr: 2.0000e-04 - 29s/epoch - 710ms/step
Epoch 2/20
```

```
ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 663	accurately-predicted-data: 660	wrongly-predicted-data:  3
Accuracy: 99.548 %
```

MOBILENET MODEL

```
[ ]
mobile = tf.keras.applications.mobilenet.MobileNet()
mobile.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf.h5
17227776/17225924 [==============================] - 0s 0us/step
17235968/17225924 [==============================] - 0s 0us/step
Model: "mobilenet_1.00_224"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3)]     0

 conv1 (Conv2D)              (None, 112, 112, 32)      864

 conv1_bn (BatchNormalizatio  (None, 112, 112, 32)     128
```

```python
model.compile(optimizer=Adam(learning_rate=0.0002), loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
anne = 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)

#datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=2, horizontal_flip=True, shear_range=0.2)
datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)

datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 1/20

Epoch 1: val_loss improved from inf to 0.41577, saving model to model.h5
41/41 - 14s - loss: 0.3328 - accuracy: 0.8877 - val_loss: 0.4158 - val_accuracy: 0.8559 - lr: 2.0000e-04 - 14s/epoch - 331ms/step
Epoch 2/20

Epoch 2: val_loss improved from 0.41577 to 0.20842, saving model to model.h5
41/41 - 9s - loss: 0.0566 - accuracy: 0.9800 - val_loss: 0.2084 - val_accuracy: 0.9481 - lr: 2.0000e-04 - 9s/epoch - 220ms/step
```

```python
##The test data is used to predict the performance of the model on unseen data and the correct prediction and wrong prediction are collected in a list with test accura

ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 663    accurately-predicted-data: 661   wrongly-predicted-data:  2
Accuracy: 99.698 %
```

VGG16 MODEL

```python
from tensorflow.keras.applications import VGG16 #For Transfer Learning
```

```python
##Building Model
IMAGE_SIZE = [224, 224]
vgg = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
#here [3] denotes for RGB images(3 channels)

#don't train existing weights
for layer in vgg.layers:
  layer.trainable = False

x = Flatten()(vgg.output)
prediction = Dense(3, activation='softmax')(x)
model = Model(inputs=vgg.input, outputs=prediction)

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
```

```
Total params: 20,099,651
Trainable params: 75,267
Non-trainable params: 20,024,384
_________________________________________________________________
```

```
anne = 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)

#datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=2, horizontal_flip=True, shear_range=0.2)
datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)

datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
                steps_per_epoch=xtrain.shape[0] //128,
                epochs=20,
                verbose=2,
                callbacks=[anne, checkpoint],
                validation_data=(xval, yval))
```

```
Epoch 1/20

Epoch 1: val_loss improved from inf to 0.84060, saving model to model.h5
41/41 - 25s - loss: 0.9900 - accuracy: 0.5571 - val_loss: 0.8406 - val_accuracy: 0.8141 - lr: 2.0000e-04 - 25s/epoch - 601ms/step
Epoch 2/20

Epoch 2: val_loss improved from 0.84060 to 0.66969, saving model to model.h5
41/41 - 16s - loss: 0.7691 - accuracy: 0.7919 - val_loss: 0.6697 - val_accuracy: 0.8576 - lr: 2.0000e-04 - 16s/epoch - 379ms/step
Epoch 3/20

Epoch 3: val_loss improved from 0.66969 to 0.57597, saving model to model.h5
41/41 - 16s - loss: 0.6343 - accuracy: 0.8454 - val_loss: 0.5760 - val_accuracy: 0.8610 - lr: 2.0000e-04 - 16s/epoch - 379ms/step
Epoch 4/20

Epoch 4: val_loss improved from 0.57597 to 0.49390, saving model to model.h5
```

```
##The test data is used to predict the performance of the model on unseen data and the correct prediction and wrong prediction are collected in a lis

ypred = model.predict(xtest)

total = 0
accurate = 0
accurateindex = []
wrongindex = []

for i in range(len(ypred)):
    if np.argmax(ypred[i]) == np.argmax(ytest[i]):
        accurate += 1
        accurateindex.append(i)
    else:
        wrongindex.append(i)

    total += 1

print('Total-test-data;', total, '\taccurately-predicted-data:', accurate, '\t wrongly-predicted-data: ', total - accurate)
print('Accuracy:', round(accurate/total*100, 3), '%')
```

```
Total-test-data; 663    accurately-predicted-data: 638    wrongly-predicted-data:  25
Accuracy: 96.229 %
```

VGG19 MODEL

```
[ ]
```

```
[ ]  VGG_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
[ ]  for layer in VGG_model.layers:
         layer.trainable = False

     VGG_model.summary()  #Trainable parameters will be 0
```

Model: "vgg19"

Layer (type)	Output Shape	Param #
input_7 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928

```
# dataset has 3 classes
num_class = 3

# Base model without Fully connected Layers
VGG_model = VGG19(include_top=False, weights='imagenet', input_shape=(224,224,3))
x=VGG_model.output
# Add some new Fully connected layers to
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x = Dropout(0.25)(x)
x=Dense(512,activation='relu')(x)
x = Dropout(0.25)(x)
preds=Dense(num_class, activation='softmax')(x) #final layer with softmax activation

model=Model(inputs=VGG_model.input,outputs=preds)
```

```
anne = 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)

#datagen = ImageDataGenerator(zoom_range = 0.2, rotation_range=2, horizontal_flip=True, shear_range=0.2)
datagen = ImageDataGenerator(zoom_range = 0.0, rotation_range=0.0, horizontal_flip=False, shear_range=0.0)

datagen.fit(xtrain)
# Fits-the-model
history = model.fit_generator(datagen.flow(xtrain, ytrain, batch_size=128),
               steps_per_epoch=xtrain.shape[0] //128,
               epochs=20,
               verbose=2,
               callbacks=[anne, checkpoint],
               validation_data=(xval, yval))
```

```
Epoch 1/20

Epoch 1: val_loss improved from inf to 0.42524, saving model to model.h5
41/41 - 18s - loss: 0.9242 - accuracy: 0.5269 - val_loss: 0.4252 - val_accuracy: 0.8090 - lr: 2.0000e-04 - 18s/epoch - 435ms/step
Epoch 2/20

Epoch 2: val_loss improved from 0.42524 to 0.21441, saving model to model.h5
41/41 - 16s - loss: 0.2816 - accuracy: 0.8839 - val_loss: 0.2144 - val_accuracy: 0.9179 - lr: 2.0000e-04 - 16s/epoch - 399ms/step
Epoch 3/20
```

References

Bailo, O., Ham, D. and Shin, Y.M. (2019) 'Red Blood Cell Image Generation for Data Augmentation Using Conditional Generative Adversarial Networks', in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1039–1048. Available at: https://doi.org/10.1109/CVPRW.2019.00136.

Bang, D. and Shim, H. (2018) 'Improved Training of Generative Adversarial Networks Using Representative Features'. arXiv. Available at: http://arxiv.org/abs/1801.09195 (Accessed: 15 May 2022).

Bowles, C. *et al.* (2018) 'GAN Augmentation: Augmenting Training Data using Generative Adversarial Networks'. Available at: https://doi.org/10.48550/ARXIV.1810.10863.

Bressem, K.K. *et al.* (2020) 'Comparing different deep learning architectures for classification of chest radiographs', *Scientific Reports*, 10(1), p. 13590. Available at: https://doi.org/10.1038/s41598-020-70479-z.

Ding, J. *et al.* (2019) ‘A Case Study of the Augmentation and Evaluation of Training Data for Deep Learning’, *Journal of Data and Information Quality (JDIQ)*, 11(4), pp. 1–22. Available at: https://doi.org/10.1145/3317573.

Salimans, T. *et al.* (2016) ‘Improved Techniques for Training GANs’. arXiv. Available at: http://arxiv.org/abs/1606.03498 (Accessed: 13 August 2022).

Wieczorek, G. *et al.* (2021) ‘Multiclass Image Classification Using GANs and CNN Based on Holes Drilled in Laminated Chipboard’, *Sensors*, 21(23), p. 8077. Available at: https://doi.org/10.3390/s21238077.

Erythrocytes Dataset: http://erythrocytesidb.uib.es/

https://www.tensorflow.org/api_docs/python/tf

TensorFlow API Versions | TensorFlow Core v2.9.1

https://github.com/bnsreenu/python_for_microscopists/tree/master/248_keras_implementation_of_GAN

https://machinelearningmastery.com/reproducible-results-neural-networks-keras/

https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/

https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-cifar-10-small-object-photographs-from-scratch/