National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Urun Gungor
Student ID: x20246404

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Urun Gungor		
Student ID:	X20246404		
Programme:	Data Analytics	**Year:**	2023
Module:	MSc Research Project		
Lecturer:	Dr. Catherine Mulwa		
Submission Due Date:	14/08/2023		
Project Title:	Leukaemia Cell Classification with using DC-GAN versus 3 Traditional Techniques		
Word Count: 47	**Page Count:** 3191		

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.
<u>ALL</u> 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: 14/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Urun Gungor
Student ID: x20246404

INTRODUCTION

This document presents research configuration steps. The architecture is Resnet-50 for each model.The configuration manual is structured as follows; data preprocessing, leukemia classification, generated data with DC-GAN and classification, produced data with ADASYN and classification, produced data with weighted random sampling with classification, It includes data produced with data augmentation and classification sections.

Code links :

Classification

https://www.kaggle.com/code/petssss/resnet-50-classification/notebook

Data generation with DC-GAN and Classification

https://www.kaggle.com/petssss/dc-gan-generated-images/edit

https://www.kaggle.com/petssss/resnet-50-with-gan-data/edit

Data Produced with ADASYN and Classification

https://www.kaggle.com/petssss/adasyn/edit

Data Produced with Weighted Random Sampling and Classification

https://www.kaggle.com/code/petssss/weighted-random/notebook?scriptVersionId=139507977

Data Produced with Data Augmentation and Classification

https://www.kaggle.com/code/petssss/data-aug/edit

Presentation link:

-From Youtube

-Presentation of Technical Report → https://www.youtube.com/watch?v=AX_5kvCzP20

-Presentation of Data and Code → https://drive.google.com/file/d/1lFf9demcIidl1qxG5-Usz36c3jAQipzy/view

-From Drive

-Presentation of Technical Report → https://www.youtube.com/watch?v=rdZyAVH1Vyk

-Presentation of Data and Code → https://drive.google.com/file/d/1S1OhqPXd2-PPLJ0nhvtDSYX1U8RKBbZs/view

SOFTWARE CONFIGURATION

1 Data Preprocessing

1.1 Import Libraries

Required libraries for all operations were imported.

```python
#Importing all the necessary libraries for image processing
import tensorflow as tf
from tensorflow.keras.layers import (
    BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense
)
#from tensorflow.keras.layers import Conv2D, Flatten, Dense, Dropout, MaxPooling2D, Batchbenignization
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.callbacks import EarlyStopping, Callback , ModelCheckpoint
from tensorflow.keras.metrics import Accuracy,binary_crossentropy, FalsePositives, FalseNegatives, TruePositives, TrueNegatives
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```python
#Importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import os
import shutil
import cv2
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.metrics import confusion_matrix, classification_report
import imgaug as ia
import imgaug.augmenters as iaa
import matplotlib.pyplot as plt
%matplotlib inline
import operator
import random
import skimage
from skimage.io import imread, imshow, imsave
```

To import numpy as np : To work with e multidimensional array-processing with high-performance.

import pandas as pd : To data analysis and manipulation tool, fast, powerfully.

import seaborn as sns : To statistical data visualizaiton.

import sklearn : To do ml operations easily.

import os : To connecting with the operating system, like creating files and directories, management of files and directories

import shutil : To make high-level operation on a file

import cv2 : To makes easier to find the package with search engines import imgaug as ia : To image augmentation in machine learning experiments.import matplotlib.pyplot as plt : To draw graps.

%matplotlib inline : To display plots inline.

import operator : To use operators.

import random : To create random numbers.

import skimage : To collection of algorithms for image processing and computer vision.

from skimage.io import imread, imshow, imsave

from PIL import Image: To add and manage for opening, manipulating, and saving many different image file formats.

Necessary libraries for image processing

import tensorflow as tf : To data automation, model tracking, performance monitoring, and model retraining.

from tensorflow.keras.layers import (BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense, MaxPool2D) :

from tensorflow.keras import layers : To make deep learning operations.

1.2 IMPORT DRIVE TO GET IMAGES

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

With using **file paths** fuction created a way to get data from Google drive.

1.3 TRAIN DATA

Data Preprocessing

```
import glob
File_paths = glob.glob("drive/content/My_Drive/MyDrive/data/C-NMC_Leukemia")
```

```
#Training dataset
train_data_0_all= glob.glob("/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/*.bmp")
train_data_0_hem= glob.glob("/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/hem/*.bmp")
train_data_1_all= glob.glob("/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_1/all/*.bmp")
train_data_1_hem= glob.glob("/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_1/hem/*.bmp")
train_data_2_all= glob.glob("/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_2/all/*.bmp")
train_data_2_hem= glob.glob("/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_3/hem/*.bmp")
```

```
#Training dataset paths
train_data_0_all_path = "/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/*.bmp"
train_data_0_hem_path = "/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/hem/*.bmp"
train_data_1_all_path = "/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_1/all/*.bmp"
train_data_1_hem_path= "/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_1/hem/*.bmp"
train_data_2_all_path = "/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_2/all/*.bmp"
train_data_2_hem_path = "/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_3/hem/*.bmp"
```

Glob is a function that used to search for files that match a specific file pattern or name to find my dataset in Google drive.

1.4 CREATE VALIDATION SET FOR ALL AND HEM

```
#Validation Dataset
valid_dataset = glob.glob('/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_test_prelim_phase_data/*.bmp')

#Validation Dataset (.CSV)
valid_data = pd.read_csv('/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_test_prelim_phase_data/C-NMC_test_prelim_phase_data_labels.csv')
```

1.5 GET DATASET DATA TYPE INFORMATION

```
valid_data.head(5)
```

	Patient_ID	new_names	labels
0	UID_57_29_1_all.bmp	1.bmp	1
1	UID_57_22_2_all.bmp	2.bmp	1
2	UID_57_31_3_all.bmp	3.bmp	1
3	UID_H49_35_1_hem.bmp	4.bmp	0
4	UID_58_6_13_all.bmp	5.bmp	1

1.6 CREATE TRAIN DATASET FOR ALL AND HEM CLASS

```
X_train = []
y_train = []
X_train_all = []
X_train_hem = []
all = [train_data_0_all, train_data_1_all, train_data_2_all]
hem = [train_data_0_hem, train_data_1_hem, train_data_2_hem]

for i in range(0, len(all)):
  for j in range(0, len(all[i])):
    X_train.append(all[i][j])
    y_train.append(0)
    X_train_all.append(all[i][j])

for i in range(0, len(hem)):
  for j in range(0, len(hem[i])):
    X_train.append(hem[i][j])
    X_train_hem.append(hem[i][j])
    y_train.append(1)

df = pd.DataFrame({'images' : X_train, 'labels' : y_train})
print(df)
```

Output :

```
                                              images  labels
0     /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       0
1     /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       0
2     /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       0
3     /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       0
4     /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       0
...                                                 ...     ...
9567  /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       1
9568  /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       1
9569  /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       1
9570  /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       1
9571  /content/drive/MyDrive/data/C-NMC_Leukemia/C-N...       1

[9572 rows x 2 columns]
```

1.7 GIVE NEW NAME VALIDATIN SET IMAGES IN DRIVE

```
X_val = []
y_val = []

for img_name in valid_data.new_names:
    X_val.append('/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_test_prelim_phase_data/' + img_name)
for label_name in valid_data.labels.values:
    y_val.append(label_name)
```

1.8 CROP FROM 410 TO 210 210 AND SAVE IMAGES

The initial size of the data images was 410. However, there was a black background around the cells. This black background was crop to most suitable size was 210.

```
def crop_center(img, bounding):
    start = tuple(map(lambda a, da: a//2-da//2, img.shape, bounding))
    end = tuple(map(operator.add, start, bounding))
    slices = tuple(map(slice, start, end))
    return img[slices]


'''Save Oversampled Cropped Images'''
def SaveOC_images(list_data, path, batch):
    for x in range(len(list_data)):
#           print(path)
        new_path = path + '/' + list_data[x]
        print(new_path)
#           break
        imsave(new_path ,batch[x])
```

1.9 IMAGE DATA TRANSFORMATION TO ARRAYS FOR 2 CLASS

```
train_all_batch = np.zeros((len(X_train_all), 210, 210, 3), dtype=np.uint8)
train_hem_batch = np.zeros((len(X_train_hem), 210, 210, 3), dtype=np.uint8)

print(train_all_batch.shape, train_hem_batch.shape)

(7272, 210, 210, 3) (2300, 210, 210, 3)
```

There are 7272 data for ALL and 2300 data for HEM in the train dataset. The array size is 210 i and has 3-dimensional.

1.10 TRAIN ALL CLASS

```
X_train_all
```

```
['/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_25_2_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_6_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_8_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_24_2_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_32_2_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_9_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_22_2_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_23_2_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_1_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_35_3_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_24_4_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_34_5_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_35_8_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_22_6_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_23_7_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_35_6_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_5_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_10_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_2_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_34_2_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_34_3_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_24_7_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_36_1_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_25_10_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_22_3_all.bmp',
...
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_5_6_1_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_5_5_2_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_5_5_3_all.bmp',
 '/content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_5_35_3_all.bmp',
 ...]
```

1.11 READ AND CROP TRAIN ALL AND HEM

```python
def Read_n_Crop(list_data, batch):
    i=0
    for x in list_data:
        image = imread(os.path.join(x))
        image = crop_center(image, (210,210,3))
        batch[i] = image
        i+=1
        print('i:',  i, 'x:',  x)

    print(type(batch), batch.shape, batch.dtype, batch[0].shape, batch[0].dtype)

Read_n_Crop(X_train_all, train_all_batch)
Read_n_Crop(X_train_hem, train_hem_batch)
```

Output

```
i: 1 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_25_2_all.bmp
i: 2 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_6_all.bmp
i: 3 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_8_all.bmp
i: 4 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_24_2_all.bmp
i: 5 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_32_2_all.bmp
i: 6 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_9_all.bmp
i: 7 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_22_2_all.bmp
i: 8 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_23_2_all.bmp
i: 9 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_1_all.bmp
i: 10 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_35_3_all.bmp
i: 11 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_24_4_all.bmp
i: 12 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_34_5_all.bmp
i: 13 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_35_8_all.bmp
i: 14 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_22_6_all.bmp
i: 15 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_23_7_all.bmp
i: 16 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_35_6_all.bmp
i: 17 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_5_all.bmp
i: 18 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_10_all.bmp
i: 19 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_33_2_all.bmp
i: 20 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_34_2_all.bmp
i: 21 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_34_3_all.bmp
i: 22 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_24_7_all.bmp
i: 23 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_36_1_all.bmp
i: 24 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_25_10_all.bmp
i: 25 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_0/all/UID_48_22_3_all.bmp
...
i: 3194 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_1/all/UID_51_63_4_all.bmp
i: 3195 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_1/all/UID_51_63_1_all.bmp
i: 3196 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_1/all/UID_51_61_2_all.bmp
i: 3197 x: /content/drive/MyDrive/data/C-NMC_Leukemia/C-NMC_training_data/fold_1/all/UID_51_61_9_all.bmp
```

2 CLASSIFICATION WITH RESNET-50

2.1 IMPORT LIBRARIES

```
#Importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import os
import shutil
import cv2
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.metrics import confusion_matrix, classification_report
import imgaug as ia
import imgaug.augmenters as iaa
import matplotlib.pyplot as plt
%matplotlib inline
import operator
import random
import skimage
from skimage.io import imread, imshow, imsave
from PIL import Image

#Importing all the necessary libraries for image processing
import tensorflow as tf
from tensorflow.keras.layers import (
    BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense , MaxPool2D
)
from tensorflow.keras import layers
```

2.2 IMPORT OTHER LIBRARIES FOR DEEP LEARNING

Sub-libraries imported such as metrics, layers optimization functions.

```
#from tensorflow.keras.layers import Conv2D, Flatten, Dense, Dropout, MaxPooling2D, Batchbenignization
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.callbacks import EarlyStopping, Callback , ModelCheckpoint
from tensorflow.keras.metrics import Accuracy,binary_crossentropy, FalsePositives, FalseNegatives, TruePositives, TrueNegatives
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.utils import to_categorical


import tqdm
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn import metrics

from skimage.transform import resize
#from skimage.color import grey2rgb

from keras import optimizers
from tensorflow.keras.optimizers.legacy import Adam

from keras.callbacks import Callback,ModelCheckpoint
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout
from keras.callbacks import ReduceLROnPlateau
from keras.wrappers.scikit_learn import KerasClassifier
import keras.backend as K
```

```
from typing import Optional


# Libraries for TensorFlow
from tensorflow.keras.preprocessing import image
from tensorflow.keras import models, layers

# Library for Transfer Learning
from tensorflow.keras.applications import resnet50
```

2.3 GET DATA FROM GOOGLE DRIVE

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

```
Mounted at /content/drive
```

```
import glob
File_paths = glob.glob("drive/content/My_Drive/data/")
```

2.4 CREATE TEST AND TRAIN SET

Train and test set assigned 2 class as represented ALL and HEM.

```
y_train = to_categorical(y_train, num_classes= 2)
y_test = to_categorical(y_test, num_classes= 2)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

Output :

```
(9572, 210, 210, 3)
(1867, 210, 210, 3)
(9572, 2)
(1867, 2)
```

2.5 INITIALIZATION OF RESNET-50 MODEL

```python
# Check properties of the model that we are going to use for Transfer Learning

print("Summary of default ResNet50 model.\n")

# we are using resnet50 for transfer learnin here. So we have imported it
from tensorflow.keras.applications import resnet50

# initializing model with weights='imagenet'i.e. we are carring its original weights
model_resnet=resnet50.ResNet50(weights='imagenet')

# display the summary to see the properties of the model
model_resnet.summary()
```

To print : print("Summary of default ResNet50 model.\n")

To import Resnet50 :from tensorflow.keras.applications import resnet50

To Initialize model with model_resnet=resnet50.ResNet50(weights='imagenet')

To display model summary : model_resnet.summary()

Output :

```
Summary of default ResNet50 model.

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102967424/102967424 [==============================] - 4s 0us/step
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
                                )

...
Total params: 25,636,712
Trainable params: 25,583,592
Non-trainable params: 53,120
```

2.6 DEFINE MODEL INNER LAYERS

```
print("Summary of Custom ResNet50 model.\n")
print("1) We setup input layer and 2) We removed top (last) layer. \n")

# let us prepare our input_layer to pass our image size which is (210,210,3)).
input_layer=layers.Input(shape=(210,210,3))

# initialize the transfer model ResNet50 with appropriate properties per our need.
# we are passing paramers as following
# 1) weights='imagenet' - Using this we are carring weights as of original weights.
# 2) input_tensor to pass the ResNet50 using input_tensor
# 3)  want to change the last layer so we are not including top layer
resnet_model=resnet50.ResNet50(weights='imagenet',input_tensor=input_layer,include_top=False)

# See the summary of the model with our properties.
resnet_model.summary()
```

To prepare input_layer to pass our image size which is (210,210,3) :
input_layer=layers.Input(shape=(210,210,3))

Change the last layer that are not including top layer:
resnet_model=resnet50.ResNet50(weights='imagenet',input_tensor=input_layer,include_top=False)

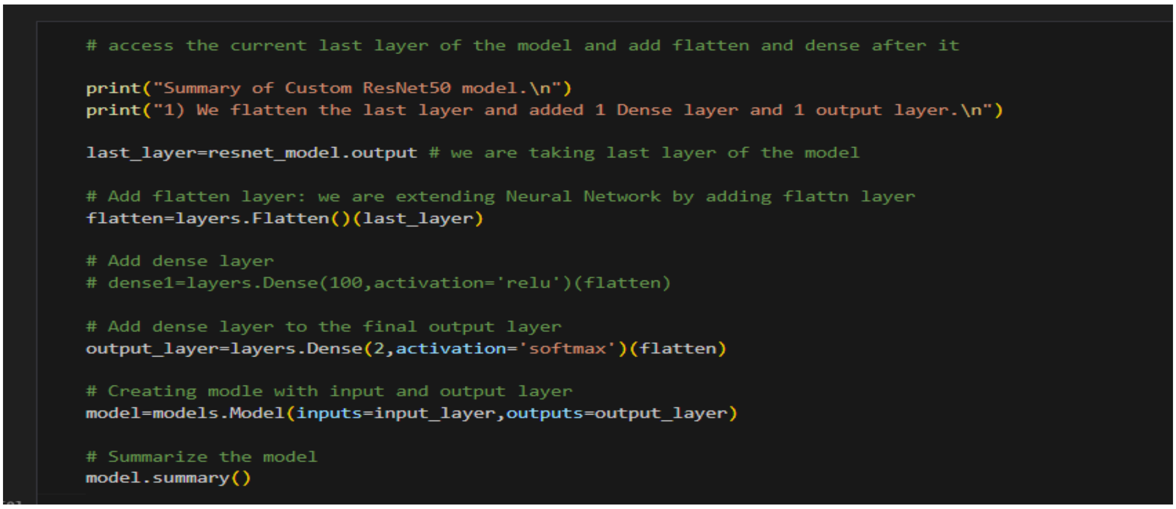
To see the summary of the model with our properties: resnet_model.summary()

Output :

```
Summary of Custom ResNet50 model.

1) We setup input layer and 2) We removed top (last) layer.

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 3s 0us/step
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 210, 210, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 216, 216, 3)  0           ['input_2[0][0]']

 conv1_conv (Conv2D)            (None, 105, 105, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 105, 105, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 105, 105, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 107, 107, 64  0           ['conv1_relu[0][0]']
...
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
```

A specific value (based on data preprocess) was assigned to the model layers as 210,210,3 and the aimed to get more effective results.

2.7 DEFINE INNER LAYER PARAMETERS



```
# access the current last layer of the model and add flatten and dense after it

print("Summary of Custom ResNet50 model.\n")
print("1) We flatten the last layer and added 1 Dense layer and 1 output layer.\n")

last_layer=resnet_model.output # we are taking last layer of the model

# Add flatten layer: we are extending Neural Network by adding flattn layer
flatten=layers.Flatten()(last_layer)

# Add dense layer
# dense1=layers.Dense(100,activation='relu')(flatten)

# Add dense layer to the final output layer
output_layer=layers.Dense(2,activation='softmax')(flatten)

# Creating modle with input and output layer
model=models.Model(inputs=input_layer,outputs=output_layer)

# Summarize the model
model.summary()
```

To define last layer : last_layer=resnet_model.output

To add flatten layer: flatten=layers.Flatten()(last_layer)

To add first dense layer : =layers.Dense(100,activation='relu')(flatten)

To add second dense layer : output_layer=layers.Dense(2,activation='softmax')(flatten)

To creating modle with input and output :
layermodel=models.Model(inputs=input_layer,outputs=output_layer)

To summarize the model :model.summary()

Relu was used as the activation function and two layes were added.

2.8 TRANSFER LEARNING and PARAMETERS

Transfer learning was applied to get fast and more effective results.Firstly, all inner layers freeze and model train with out layer.

base_model_resnet.trainable = False → represent freeze inner layer

Secondly model trained with all layers.

base_model_resnet.trainable = True → represent train model with all layer

a) Train with only out layer

```
base_model_resnet.trainable = False
```

```
# Modelling WITH Transfer Learning

#  prepare model as per requirements

print("Summary of Custom ResNet50 model.\n")
print("1) We setup input layer and 2) We removed top (last) layer. \n")

# let us prepare our input_layer to pass our image size. default is (224,224,3). we will change it to (100,100,3)
input_layer=layers.Input(shape=(210,210,3))

# by passing `training=False`. This is important for fine-tuning, as you will
# learn in a few paragraphs.
resnet_model = base_model_resnet(input_layer, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
resnet_model = layers.GlobalAveragePooling2D()(resnet_model)
# A Dense classifier with a single unit (binary classification)
outputs = layers.Dense(2)(resnet_model)
model = keras.Model(input_layer, outputs)
```

```
Summary of Custom ResNet50 model.

1) We setup input layer and 2) We removed top (last) layer.
```

```
metrics = ['accuracy', tf.keras.metrics.Precision() , tf.keras.metrics.FalseNegatives() ,tf.keras.metrics.FalsePositives(),
            tf.keras.metrics.TruePositives(), tf.keras.metrics.TrueNegatives()]

model.compile(optimizer=keras.optimizers.Adam(0.001),  # Very low learning rate
               loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
               metrics=metrics)

model.fit(xtrain,ytrain,epochs=5,batch_size=32,verbose=True,validation_data=(xtest,ytest))
```

b) Train with all layers

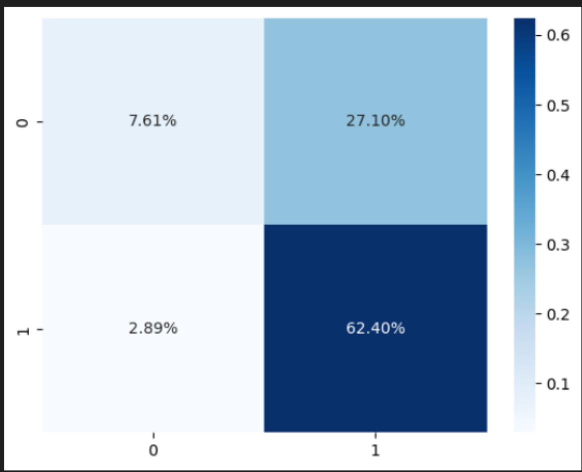
```
base_model_resnet.trainable = True

# It's important to recompile your model after you make any changes



# to the `trainable` attribute of any inner layer, so that your changes
# are take into account
model.compile(optimizer=keras.optimizers.Adam(0.0001),  # Very low learning rate
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=metrics)

# Train end-to-end. Be careful to stop before you overfit!
history =  model.fit(xtrain,ytrain,epochs=10,batch_size=32,verbose=True,validation_data=(xtest,ytest))
```

2.9 VISUALIZATION

a) Confusion matrix

```
sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')
```

<Axes: >



b) Metrics

```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.show()
```

c) Loss graph

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

d) Accuracy graph

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

3 GENERATED IMAGES WITH DC-GAN AND CLASSIFICATION

GAN is a machine learning algorithm that trains a generator and allocator simultaneously and generates new data in a loop. The main goal is to generate healthy, cancer-free synthetic data to balance the CNMC 2019 dataset with DCGAN.
Healthy cells are less than 4000 leukemia cells. For this reason, approximately 5028 healthy (HEM) synthetic data were produced. On the other hand, producing high quality synthetic data is the other goal. Therefore, the discriminator is not necessary in the first place. However, the discriminator controlling the similarity ratio between the original data and the synthetic data was not excluded, as it was not removed. Since the similarity ratio between the synthetic data and the original data was aimed to be high, the training time was ignored and the learning rate was kept small.
The next step was add generated images to original dataset and classification with Resnet-50.

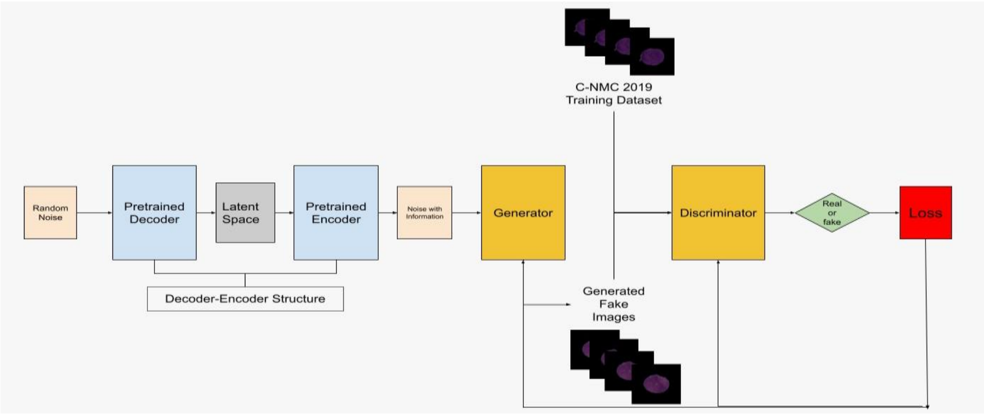


Figure : Architecture of DC-GAN

3.1 IMPORT LIBRARIES

```
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
import operator
import tensorflow as tf
import random
from keras.preprocessing.image import ImageDataGenerator

import sys, os, glob, time, imageio
import numpy as np, pandas as pd

import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML

from PIL import Image

import torch
import torchvision.utils as vutils
import torchvision.transforms as transforms

from keras import models, layers, optimizers
from keras.models import Sequential
from keras.utils import array_to_img, img_to_array, load_img

import tensorflow as tf
```

Import Numpy as Np : There are lists used instead of arrays in Python, but process is too slow. NumPy is, returning an array up to 50 times faster than traditional Python lists because most of the fast computational parts are written in C or C++.

Import Os : The OS module in Python is to interface with the Windows, Mac or Linux operating system on which Python runs, by making the operating system use its functionality.

Import Cv2: Opencv is a basic library for image analysis and has more than 2,500 optimized algorithms. It works easily in windows, Cv2 is last version.

Import Matplotlib.Pyplot As Plt : Matplotlip was used for 2D graphics and working with multiple graphics.

%Matplotlib Inline : It is a function that contribute to renders the figure in a notebook, instead of displaying a dump of the figure object for more fast.

Import Operator : Operator module be used for efficient functions for better comparison.

Import Tensorflow As Tf : Tensorflow is a fundamental library that used for all operation especially deep network algorithms.

Import Random : Used for generate random numbers.

From Keras.Preprocessing.Image Import Imagedatagenerator : For generating batches image data with real-time data augmentation.

Import Sys, Os, Glob, Time, Imageio : output script name with sys, recursively create folders in the current path, Directory tree generator with os, file search with glob.

Import Numpy As Np, Pandas As Pd : For data manipulation and analysis.

Import Matplotlib.Pyplot As Plt : Part of matplotlip for graphic and chart.

Import Matplotlib.Animation As Animation : Part of matplotlip for animation.

From Ipython.Display Import Html : To embed rendered HTML output into IPython output.

From Pil Import Image : To import images.

Import Torch : A Tensor library like NumPy, to support GPU.

Import Torchvision.Utils As Vutils : To effective visualization.

From Keras Import Models, Layers, Optimizers : To optimizate of model and layer.

From Keras.Models Import Sequential : To assign each layer has exactly one input tensor and one output tensor.

From Keras.Utils Import Array_To_Img, Img_To_Array, Load_Img : For array converted to image.

Import Tensorflow As Tf : Fundamental library.

3.2 GET VALIDATION AND TEST DATA ON KAGGLE

```
# Main Dataset
main_folder="/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data"
class_names=os.listdir(main_folder)
print(class_names)

# Validation Dataset
validation_folder="/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid"
val_class_names=os.listdir(validation_folder)
print(val_class_names)
```

```
['all', 'hem']
['all', 'hem']
```

3.3 CREATE TRAİN AND VALIDATION DATASET AND DEFINE A PATH TO GET TEST AND TRAINING SET

```
# Root directory for dataset
path_train= "/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/"
#path_test = '/kaggle/working/Cancer_test/'
path_val  = "/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/"



# Training images
train_all    = glob.glob(path_train+'all/*.bmp', recursive=True)
train_hem    = glob.glob(path_train+'hem/*.bmp', recursive=True)

# Testing images
test_all    = os.listdir(path_val+'all/')
test_hem    = os.listdir(path_val+'hem/')

# print('   - {:04d} all and {:04d} hem ==> {:04d} images in the training sample'\
#       .format(len(Ximg_all),
#               len(Ximg_hem),
#               len(glob.glob(path_root+'*/*.bmp'))))

print('   - {:04d} all and {:04d} hem ==> {:04d} images in the training sample'\
      .format(len(train_all),
              len(train_hem),
              len(glob.glob(path_train+'*/*.bmp'))))
print('   - {:04d} all and {:04d} hem ==> {:04d} images in the testing sample'\
      .format(len(test_all),
              len(test_hem),
              len(glob.glob(path_val+'*/*.bmp'))))
```

```
- 7272 all and 2300 hem ==> 9572 images in the training sample
- 1219 all and 0648 hem ==> 1867 images in the testing sample
```

import glob : File search for glob.

File_paths = With a path to reach and get data from Kaggle.

To root directory for dataset with used path_root, path_train, path_test, path_val.

To training images first step is, defined Ximg_all with glob and Ximg_hem.

To training images used train_all and train_hem with glob.

To test images test_all and test_hem with os.

To Print with .format(len(Ximg_all), len(Ximg_hem),

To print with format(len(train_all), len(train_hem),

To print with .format(len(test_all), len(test_hem).

Output

```
- 2397 all and 1137 hem ==> 3534 images in the training sample
- 7272 all and 2300 hem ==> 9572 images in the training sample
- 1219 all and 0648 hem ==> 1867 images in the testing sample
```

3.4 DEFINE TIME TYPES WİTH İF / ELSE

```python
# Time
def _time(start, end):
    # if in seconds
    if (end-start)<60:
        wall_time = f'{round((end-start),2)}sec'
    # if in minute(s)
    elif (end-start)>=3600:
        wall_time = f'{int((end-start)/3600)}h {int(((end-start)%3600)/60)}min {round((end-start)%60,2)}sec'
    # if in houre(s)
    else:
        wall_time = f'{int((end-start)/60)}min {round((end-start)%60,2)}sec'
    return wall_time
```

3.5 RESIZE IMAGES to 128 X 128 PIXEL SIZE AND PLOT

```python
nrows, ncols = 4, 7
plt.figure(figsize=(16,10))
for idx, name in enumerate(test_all[:nrows*ncols]):
    plt.subplot(nrows, ncols, idx+1)
    img = Image.open(path_val+'/all/'+name) # or use plt.imread(path_test+'benign/'+name)
    img = img.resize(size=(128, 128), resample=Image.ANTIALIAS, box=None)
    plt.imshow(img)
    plt.title(name[:-5], fontsize=9)
    plt.axis('off')
```

```
/tmp/ipykernel_28/5901719.py:6: DeprecationWarning: ANTIALIAS is deprecated and will be removed in Pillow 10 (2023-07-01). Use LANCZOS or Resampling.LANCZOS instead.
  img = img.resize(size=(128, 128), resample=Image.ANTIALIAS, box=None)
```

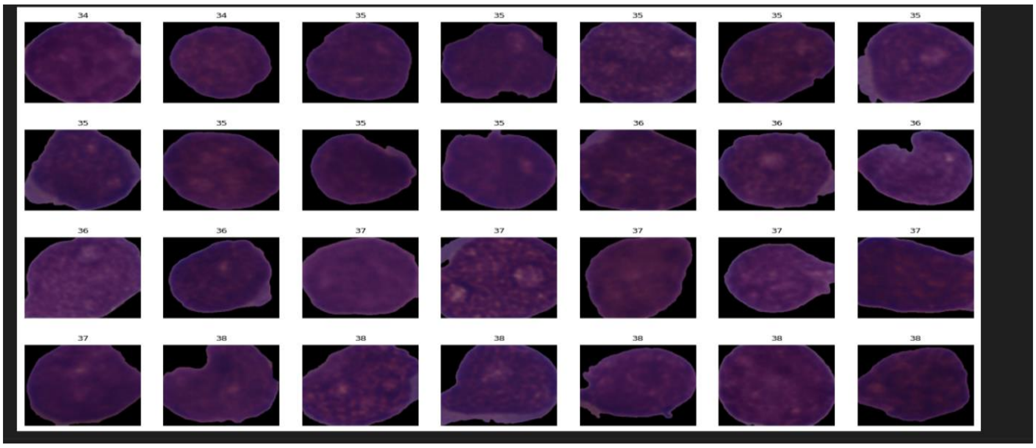
nrows, ncols = 4, 7 : To plot images with 4 rows and 7 colomns.

plt.figure : Define figure size 16 to 10.

img = img.resize :, resample 128 to 128 pixel size for processing data in dcgan simplier.

plt.imshow(img) and plt.title(name[:-5], fontsize=9) : Show and assign cell plot properties.

OUTPUT



3.6 CONVERT NO ARRAY AND NORMALIZED IMAGES

This part to resizes the pictures, converts them to no arrays, and normalizes them.

```python
def get_data(data_path, dim=(128, 128), rand_shuffle=True):
    start = time.time()
    imgs_data = []
    sample_size = len(data_path)
    for idx, im_path in enumerate(data_path):
        if idx%(sample_size//10)==0:
            print('Processing index {:05d} of {:05d} ==> {:03d}%'\
                  .format(idx, sample_size, round(100*idx/sample_size)))
        img = img_to_array(load_img(im_path, target_size = dim))
        imgs_data.append(img)

    # to float
    imgs_data = np.array(imgs_data).astype('float32')
    # scale to [0,1] (note the . after 255 - float)
    imgs_data = imgs_data/255. #for formalizing to [-1,1] ==> (imgs_data - 127.5)/127.5

    # shuffle the data
    if rand_shuffle:
        idx = np.arange(imgs_data.shape[0])
        np.random.shuffle(idx)
        imgs_data = imgs_data[idx,:,:,:]

    print(f"Hey! the calculations are done in {_time(start, time.time())}")
    return imgs_data
```

3.7 CREATE TRAIN DATASET

```python
print('Starting for all images ...')
#X_all = get_data(train_all)
print()
print('Starting for hem images ...')
X_hem = get_data(train_hem)
```

OUTPUT

```
Starting for all images ...

Starting for hem images ...
Processing index 00000 of 02300 ==> 000%
Processing index 00230 of 02300 ==> 010%
Processing index 00460 of 02300 ==> 020%
Processing index 00690 of 02300 ==> 030%
Processing index 00920 of 02300 ==> 040%
Processing index 01150 of 02300 ==> 050%
Processing index 01380 of 02300 ==> 060%
Processing index 01610 of 02300 ==> 070%
Processing index 01840 of 02300 ==> 080%
Processing index 02070 of 02300 ==> 090%
Hey! the calculations are done in 6min 43.71sec
```

()shape is to Get the dimensions of Pandas and NumPy type objects in Python.

```python
X_hem.shape[0]
```

```
2300
```

3.8 CREATE GRID GRAPH

```python
def define_grid(data_images, nrows=4, ncols=5, plot_grid=True):
    # save the started time
    start = time.time()
    # Number of GPUs available. Use 0 for CPU mode.
    ngpu = 1
    # Decide which device we want to run on
    device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")
    # Rearange the shaphe of the data
    data_transp = [np.transpose(data_images[i,:,:]) for i in range(data_images[:nrows*ncols].shape[0])]
    # From to torch type for the grid
    data_transp = torch.Tensor(data_transp)
    print(f'The shape is reordered from {data_images.shape[1:]} to {data_transp.shape[1:]} in {_time(start, time.time())}')

    # Make the grid
    grid_images = np.transpose(
        vutils.make_grid(
            data_transp.to(device)[:nrows*ncols],
            nrow=nrows,
            padding=2,
            benignize=True,
            scale_each=True,
            pad_value=1,
        ).cpu(), axes=(2,1,0))

    # Show the output grid
    if plot_grid:
        plt.figure(figsize=(12,12))
        plt.axis("off")
        plt.title(f'Grid of {nrows*ncols} real images', fontsize=27)
        plt.imshow(grid_images)

    return grid_images

#grid_X_benign = define_grid(X_all, plot_grid=False)
grid_X_hem = define_grid(X_hem, plot_grid=False)
```

The algorithm required to define, create and draw the grid function was created and the size of the array was adjusted accordingly.

OUTPUT

```
The shape is reordered from (128, 128, 3) to torch.Size([3, 128, 128]) in 0.36sec
```

3.9 MAIN PART / CREATE DC-GAN NETWORK ARCHITECTURE

```
# Number of training epochs
n_epoch = 1000

# Batch size during training
batch_size = 128

# Size of z latent vector (i.e. size of generator input)
latent_dim = 100

# Spatial size of training images. All images will be resized to this size
cols, rows = 128, 128

# Number of channels in the training images. For RGB color images this is 3
channels = 3
dim = cols, rows # height, width
in_shape = (cols, rows, channels) # height, width, color

# Learning rate for optimizers
lr=0.0001

# Beta1 hyperparam for Adam optimizers
beta1 = 0.5

# Number of GPUs available. Use 0 for CPU mode.
ngpu = 1

# plot ncols images in row and nrows images in colomn
nrows, ncols = 3, 4
```

n_epoch =# Number of training epochs assigned 1000

batch_size = Batch size during training assigned 128.

latent_dim = Size of z latent vector, generator imput assigned 100

cols, rows = 128, 128 (images resized)

Number of channels in the training images.

channels = Number of channels in the training images assigned 3. For RGB color images this is 3.

dim = define dimensions

in_shape = (cols, rows, channels) # height, width, color

lr = Learning rate for optimizers assigned 0.001

beta1 = Beta1 hyperparam assigned for Adam optimizers to 0.5..

ngpu = Number of GPUs available which means assigned 1. Use 0 for CPU mode.#

nrows, ncols = plot ncols images in row and nrows images in column assigned 3, and 4 respectively.

3.10 IMPORT TENSORFLOW AND KERAS OPTIMIZER ADAM FOR DC-GAN NETWORK ARCHITECTURE

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.optimizers.legacy import Adam
```

Adam used as optimization fuction.

3.11 DEFINE DISCRIMINATOR AND LAYERS

```
def define_discriminator(in_shape=(50,50,3)):
    model = models.Sequential()
    # benign
    model.add(layers.Conv2D(64, (5,5), padding='same', input_shape=in_shape))
    model.add(layers.LeakyReLU(alpha=0.2))
    # downsample to 64x64
    model.add(layers.Conv2D(32, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # downsample to 32x32
    model.add(layers.Conv2D(16, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # downsample to 16x16
    model.add(layers.Conv2D(8, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # downsample to 8x8
    model.add(layers.Conv2D(4, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # classifier
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.4))
    model.add(layers.Dense(1, activation='sigmoid'))
    # compile model
    opt = tf.keras.optimizers.legacy.Adam(learning_rate=0.0002,beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    print(model.summary())
    return model
```

define_discriminator   in_shape= images pixel size assined 50,50,3 for easier train.

model = models.Sequential()

model.add assigned (l**ayers.Conv2D =** (64, (5,5)

(layers.LeakyReLU assigned alpha=0.2)

 # downsample started to 64x64

 # first layer

model.add(layers.Conv2D(32, (5,5), strides=(2,2), padding='same'))

model.add(layers.LeakyReLU(alpha=0.2))

 # downsample to 32x32

 # second layer

model.add(layers.Conv2D(16, (5,5), strides=(2,2), padding='same'))

model.add(layers.LeakyReLU(alpha=0.2))

 # downsample to 16x16

 # third layer

model.add(layers.Conv2D(8, (5,5), strides=(2,2), padding='same'))

model.add(layers.LeakyReLU(alpha=0.2))

 # downsample to 8x8

 # fourth layer

model.add(layers.Conv2D(4, (5,5), strides=(2,2), padding='same'))

model.add(layers.LeakyReLU(alpha=0.2))

 # classifier function and parameters

model.add(layers.Flatten()), model.add(layers.Dropout(0.4)), model.add(layers.Dense(1, activation='sigmoid'))

 # compile model

Used adam for optimize fuction with 0.0002 learning rate beta1:0.5 and binary cross entropy loss function.(This parameter changed step by step while dcgan traninig better.)

3.12 DEFINE GENERATOR LAYERS

```python
def define_generator(latent_dim):
    model = models.Sequential()
    # foundation for 8x8 feature maps
    n_nodes = 128*8*8
    model.add(layers.Dense(n_nodes, input_dim=latent_dim))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Reshape((8, 8, 128)))
    # upsample to 16x16
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # upsample to 32x32
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # upsample to 64x64
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # upsample to 128x128
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # output layer 128x128x3
    model.add(layers.Conv2D(3, (5,5), activation='tanh', padding='same'))
    return model

#input of G
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = np.random.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

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(g_model, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)
    # create 'fake' class labels (0)
    y = np.zeros((n_samples, 1))
    return X, y
```

def define_generator(latent_dim): number of nodes used as input of the generator

model = models.Sequential() :To build a way for model and add layers

foundation for 8x8 feature maps

n_nodes = Assigned to 128*8*8

model.add(layers.Dense(n_nodes, input_dim=latent_dim))

(layers.LeakyReLU : assigned (alpha=0.2))

(layers.Reshape assigned ((8, 8, 128)))

With similar logic but opposite of generator

To upsample to 16x16 : model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))

(layers.LeakyReLU assigned (alpha=0.2)) for all layers.

To upsample to 32x32 : model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))

To upsample to 64x64 :model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))

To upsample to 128x128 : model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))

To output layer 128x128x3 : model.add(layers.Conv2D(3, (5,5), activation='tanh', padding='same'))

To get input of G :def generate_latent_points(latent_dim, n_samples):

To generate points in the latent space :x_input = np.random.randn(latent_dim*n_samples)

To reshape into a batch of inputs for the network : x_input = x_input.reshape(n_samples, latent_dim)

To get result : return x_input

To use the generator to generate n fake examples, with class labels : def generate_fake_samples(g_model, latent_dim, n_samples):

To generate points in latent space : x_input = generate_latent_points(latent_dim, n_samples)

To predict outputs : X = g_model.predict(x_input)

To create 'fake' class labels (0) :y = np.zeros((n_samples, 1))

3.13 DEFINE GENERAL MODEL AND METRICS

```
def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect them
    model = models.Sequential()
    # add generator
    model.add(g_model)
    # add the discriminator
    model.add(d_model)
    # compile model
    opt = tf.keras.optimizers.legacy.Adam(learning_rate=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model

# retrive real samples
def get_real_samples(dataset, n_samples):
    # choose random instances
    ix = np.random.randint(0, dataset.shape[0], n_samples)
    # retrieve selected images
    X = dataset[ix]
    # set 'real' class labels (1)
    y = np.ones((n_samples, 1))
    return X, y

# create and save a plot of generated images
def show_generated(generated, epoch, nrows=4, ncols=5):
    #[-1,1] -> [0,1]
    #generated = (generated+1)/2
    #generated = (generated[:ncols*nrows]*127.5)+127.5
    #generated = generated*255
    plt.figure(figsize=(10,10))
    for idx in range(nrows*ncols):
        plt.subplot(nrows, ncols, idx+1)
        plt.imshow(generated[idx])
        plt.axis('off')
    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch+1))
    plt.show()
```

To make weights in the discriminator not trainable : def define_gan(g_model, d_model):

d_model.trainable = False

To connect them : model = models.Sequential()

To add generator : model.add(g_model)

To add the discriminator : model.add(d_model)

To compile model with same parameters : tf.keras.optimizers.legacy.Adam(learning_rate=0.0002, beta_1=0.5)

To model.compile : (loss='binary_crossentropy', optimizer=opt)

To retrive real samples : def get_real_samples(dataset, n_samples):

To choose random samples : ix = np.random.randint(0, dataset.shape[0], n_samples)

To retrieve selected images : X = dataset[ix]

To set 'real' class labels (1) : y = np.ones((n_samples, 1))

return X, y

To create and save a plot of generated images : def show_generated(generated, epoch, nrows=4, ncols=5):

To plot figures : plt.figure(figsize=(10,10))

3.14 EVALUATE DISCRIMINATOR AND PLOT LOSS FUNCTION

```
# evaluate the discriminator and plot generated images
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples=100):
    # prepare real samples
    X_real, y_real = get_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 at epoch %d [real: %.0f%%, fake: %.0f%%]'%(epoch+1, acc_real*100, acc_fake*100))
    # show plot
    show_generated(x_fake, epoch)
    filename = 'generator_model_%03d.h5' % (epoch+1)
    g_model.save(filename)

def plot_loss(loss):
    plt.figure(figsize=(10,5))
    plt.title("Generator and Discriminator Loss During Training", fontsize=20)
    plt.plot(loss[0], label="D_real")
    plt.plot(loss[1], label="D_fake")
    plt.plot(loss[2], label="G")
    plt.xlabel("Iteration", fontsize=20); plt.ylabel("Loss", fontsize=20)
    plt.legend(); plt.show()
```

To evaluate the discriminator and plot generated images : def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples=100):

To prepare real samples : X_real, y_real = get_real_samples(dataset, n_samples)

To evaluate discriminator on real examples : _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)

To prepare fake examples : x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)

To evaluate discriminator on fake examples : _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)

To summarize discriminator performance : print('> Accuracy at epoch %d [real: %.0f%%, fake: %.0f%%]'%(epoch+1, acc_real*100, acc_fake*100))

To show plot : show_generated(x_fake, epoch)

filename = 'generator_model_%03d.h5' % (epoch+1)

To save : g_model.save(filename)

To define plot of loss function : def plot_loss(loss):

To draw : plt.figure(figsize=(10,5))

plt.title("Generator and Discriminator Loss During Training", fontsize=20)

3.15 TRAIN DISCRIMINATOR AND GENERATOR, AND DEFINE PARAMETERS

```
def train(g_model, d_model, gan_model, dataset, latent_dim=100, n_epochs=10000, n_batch=128):

    start = time.time()
    bat_per_epo = int(dataset.shape[0]/n_batch)
    half_batch = int(n_batch/2)
    loss1, loss2, loss3 = [], [], []
    fake_liste = []

    # manually enumerate epochs
    print('Training Start...')
    for i in range(n_epochs):
        start1 = time.time()
        # enumerate batches over the training set
        for j in range(bat_per_epo):
            # get randomly selected 'real' samples
            X_real, y_real = get_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 = np.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
            loss1.append(d_loss1); loss2.append(d_loss2); loss3.append(g_loss)

        print('Epoch: {:03d}/{:03d}, Loss: [D_real = {:2.3f}, D_fake = {:2.3f}, G = {:2.3f}], time: {:s}'\
              .format(i+1,n_epochs,d_loss1,d_loss2,g_loss, _time(start1,time.time())))
        # evaluate the model performance
        if (i+1)%(n_epochs//10) == 0:
            # Save and show generated images
            summarize_performance(i, g_model, d_model, dataset, latent_dim)

    print('Total time for training {} epochs is {} sec'.format(n_epochs, _time(start, time.time())))

    # Show loss curves
    loss = (loss1, loss2, loss3)
    plot_loss(loss)
```

To train model :def train(g_model, d_model, gan_model, dataset, latent_dim=100, n_epochs=10000, n_batch=128):

To manually enumerate epochs : print('Training Start...')

for i in range(n_epochs):

start1 = time.time()

To enumerate batches over the training set : for j in range(bat_per_epo):

To get randomly selected 'real' samples : X_real, y_real = get_real_samples(dataset, half_batch)

To update discriminator model weights : d_loss1, _ = d_model.train_on_batch(X_real, y_real)

To generate 'fake' examples : X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)

To update discriminator model weights : d_loss2, _ = d_model.train_on_batch(X_fake, y_fake)

To prepare points in latent space as input for the generator :X_gan = generate_latent_points(latent_dim, n_batch)

To create inverted labels for the fake samples :y_gan = np.ones((n_batch, 1))

To update the generator via the discriminator's error :g_loss = gan_model.train_on_batch(X_gan, y_gan)

To summarize loss on this batch :loss1.append(d_loss1); loss2.append(d_loss2); loss3.append(g_loss)

To evaluate the model performance :if (i+1)%(n_epochs//10) == 0:

To save and show generated images :summarize_performance(i, g_model, d_model, dataset, latent_dim)

To show loss curves :loss = (loss1, loss2, loss3)

3.16 DEFINE DISCRIMINATOR AND GENERATOR AND TRAIN MODEL

```
discriminator = define_discriminator()
generator = define_generator(latent_dim)

# create the gan
gan = define_gan(generator, discriminator)
```

```
# train model
train(generator, discriminator, gan, X_hem, latent_dim, n_epochs=10000, n_batch=batch_size)
```

3.17 SAVE IMAGES AS ZIP FILE

```python
# Assuming 'generator' is your compiled DCGAN generator model
# and 'num_images' is the number of images you want to generate
def generate_and_save_images(generator, num_images, output_folder):
    # Create the output folder if it doesn't exist
    os.makedirs(output_folder, exist_ok=True)

    # Generate images using the generator model
    noise = tf.random.normal([num_images, latent_dim])  # Assuming you have a latent_dim
    generated_images = generator.predict(noise)

    # Convert the images to the range [0, 255] and of data type 'uint8'
    generated_images = ((generated_images + 1) * 127.5).astype(np.uint8)

    # Save each generated image to the output folder
    for i in range(num_images):
        image = Image.fromarray(generated_images[i])
        image.save(os.path.join(output_folder, f"generated_image_{i}.bmp"))


output_folder = '/kaggle/working/Gan_Images'
num_images = 5028
generate_and_save_images(generator, num_images, output_folder)
```

```
158/158 [==============================] - 8s 50ms/step
```

```python
import shutil

shutil.make_archive('Gan_Generated_Images','zip','/kaggle/working/Gan_Images')
```

```
'/kaggle/working/Gan_Generated_Images.zip'
```

OUTPUT

Gan_Generated_Images.zip - ZIP arşivi, açılmış boyut 247,407,768 bayt

Ad	Boyut	Sıkı. boyut	Tür	Değişme	CRC32
..			File folder		
generated_imag...	49,206	21,855	BMP File	8/3/2023 9:10 P...	59FB194A
generated_imag...	49,206	20,948	BMP File	8/3/2023 9:10 P...	C031B9BA
generated_imag...	49,206	20,966	BMP File	8/3/2023 9:10 P...	615135C1
generated_imag...	49,206	15,414	BMP File	8/3/2023 9:10 P...	91B8757D
generated_imag...	49,206	22,748	BMP File	8/3/2023 9:10 P...	C57F73DC
generated_imag...	49,206	13,353	BMP File	8/3/2023 9:10 P...	5567D686
generated_imag...	49,206	18,677	BMP File	8/3/2023 9:10 P...	C4CCB4BE
generated_imag...	49,206	29,913	BMP File	8/3/2023 9:10 P...	FFFA3BF2
generated_imag...	49,206	21,901	BMP File	8/3/2023 9:10 P...	A847638B
generated_imag...	49,206	24,462	BMP File	8/3/2023 9:10 P...	A5F70B7A
generated_imag...	49,206	23,819	BMP File	8/3/2023 9:10 P...	452A67C9
generated_imag...	49,206	24,483	BMP File	8/3/2023 9:10 P...	93406EA7
generated_imag...	49,206	15,878	BMP File	8/3/2023 9:10 P...	D85D03AF
generated_imag...	49,206	20,623	BMP File	8/3/2023 9:10 P...	96F554D3
generated_imag...	49,206	23,024	BMP File	8/3/2023 9:10 P...	9224E2D7
generated_imag...	49,206	23,983	BMP File	8/3/2023 9:10 P...	386B5EEB
generated_imag...	49,206	23,773	BMP File	8/3/2023 9:10 P...	910B0890
generated_imag...	49,206	20,473	BMP File	8/3/2023 9:10 P...	6ADE2C2E
generated_imag...	49,206	20,830	BMP File	8/3/2023 9:10 P...	FD40CFA5
generated_imag...	49,206	23,599	BMP File	8/3/2023 9:10 P...	471BD70E

3.18 CLASSIFICATION WITH GENERATED DATA ADD TO DATASET

Generated images add to original dataset and classification did again. The model, used libraries, graphics are same as the first classification model. Therefore, only the different part in code are presented.

3.19 ADD GENERATED DATA TO DATASET

```
folder_all_train = '/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/all'
folder_hem_train = '/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/hem'

folder_all_test = '/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/all'
folder_hem_test = '/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/hem'

gan_generated_data = '/kaggle/input/gan-generated-data'
```

3.20 CREATE TEST AND TRAIN SET WITH NEW DATA

```
y_train = to_categorical(y_train, num_classes= 2)
y_test = to_categorical(y_test, num_classes= 2)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(10220, 210, 210, 3)
(1867, 210, 210, 3)
(10220, 2)
(1867, 2)
```

3.21 CLASSIFATION AND VISUALIZATION

The model, metrics, and visualization graphs are totally same with first classification therefore not presented here.

4 PRODUCED IMAGES WITH ADASYN and CLASSIFICATION

Traditional models run on kaggle TPU, synthetic data generation with DC-Gan was run on kaggle GPU because there was no RAM. However, Kagle's weekly quota of 12 hours and cloud queues were noted as hardware limitaitons.

TPUs are popular right now. You are #37 in the queue. You can wait, try connecting again later, or use another accelerator.

```
base_model_resnet.trainable = True

# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
# are take into account
model.compile(optimizer=keras.optimizers.Adam(1e-5),  # Very low learning rate
              loss='binary_crossentropy',
              metrics=metrics)

# Train end-to-end. Be careful to stop before you overfit!
model.fit(xtrain,ytrain,epochs=20,batch_size=64,verbose=True,validation_data=(xtest,ytest))
```

4.1 IMPORT LIBRARIES

```
%matplotlib inline
import operator
import random
import skimage
from skimage.io import imread, imshow, imsave
from PIL import Image
from imblearn.over_sampling import ADASYN

#Importing all the necessary libraries for image processing
import tensorflow as tf
from tensorflow.keras.layers import (
    BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense , MaxPool2D
)
from tensorflow.keras import layers

#from tensorflow.keras.layers import Conv2D, Flatten, Dense, Dropout, MaxPooling2D, Batchbenignization
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.callbacks import EarlyStopping, Callback , ModelCheckpoint
from tensorflow.keras.metrics import Accuracy,binary_crossentropy, FalsePositives, FalseNegatives, TruePositives, TrueNegatives
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.utils import to_categorical
```

adasyn 1/

from imblearn.over_sampling import ADASYN: The main library for imported ADASYN.

```
import tqdm
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn import metrics

from skimage.transform import resize
#from skimage.color import grey2rgb

from keras import optimizers
from tensorflow.keras.optimizers.legacy import Adam

from keras.callbacks import Callback,ModelCheckpoint
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout
from keras.callbacks import ReduceLROnPlateau
from keras.wrappers.scikit_learn import KerasClassifier
import keras.backend as K

#import tensorflow_addons as tfa
#from tensorflow.keras.metrics import Metric
#from tensorflow_addons.utils.types import AcceptableDTypes, FloatTensorLike
#from typeguard import typechecked
from typing import Optional


# Libraries for TensorFlow
from tensorflow.keras.preprocessing import image
from tensorflow.keras import models, layers

# Library for Transfer Learning
from tensorflow.keras.applications import resnet50
from tensorflow import keras
```

4.2 CREATE TRAIN AND TEST SET

```
folder_all_train = '/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/all'
folder_hem_train = '/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/hem'

folder_all_test = '/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/all'
folder_hem_test = '/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/hem'

read = lambda imname: np.asarray(Image.open(imname).convert("RGB"))
```

4.3 DEFINE CLASSES AND TRAIN MODEL

```
y_train = to_categorical(y_train, num_classes= 2)
y_test = to_categorical(y_test, num_classes= 2)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(9572, 210, 210, 3)
(1867, 210, 210, 3)
(9572, 2)
(1867, 2)
```

```
# Train the Model

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X_train,y_train,test_size=0.2,random_state=5)
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)

print("Splitting data for train and test completed.")
```

```
(7657, 210, 210, 3)
(1915, 210, 210, 3)
(7657, 2)
(1915, 2)
Splitting data for train and test completed.
```

4.4 MAIN PART / DEFINE ADASYN ALGORITHM

```
num_samples = xtrain.shape[0]
X_train_flat = xtrain.reshape(num_samples, -1)

adasyn = ADASYN(sampling_strategy='auto', random_state=42)
X_train_balanced, y_train_balanced = adasyn.fit_resample(X_train_flat, ytrain)
X_train_balanced = X_train_balanced.reshape(-1, 210, 210, 3)
y_train_balanced = to_categorical(y_train_balanced, num_classes= 2)
```

```
print(X_train_balanced.shape)
y_train_balanced.shape
```

```
(11470, 210, 210, 3)
(11470, 2)
```

4.5 MODEL WITH RESNET-50

```
# Check properties of the model that we are going to use for Transfer Learning

print("Summary of default ResNet50 model.\n")

# are using resnet50 for transfer learnin here. So we have imported it
from tensorflow.keras.applications import resnet50

# initializing model with weights='imagenet'i.e. we are carring its original weights
base_model_resnet=resnet50.ResNet50(weights='imagenet',input_shape=(210, 210, 3), include_top=False)

# display the summary to see the properties of the model
base_model_resnet.summary()
```

```
Summary of default ResNet50 model.

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 210, 210, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 216, 216, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 105, 105, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 105, 105, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 105, 105, 64  0           ['conv1_bn[0][0]']
                                )
```

4.6 TRANSFER LEARNING

a) Freeze inner layers

```
base_model_resnet.trainable = False
```

```
# Modelling WITH Transfer Learning

# Here we will prepare model as per our requirements
print("Summary of Custom ResNet50 model.\n")
print("1) We setup input layer and 2) We removed top (last) layer. \n")

input_layer=layers.Input(shape=(210,210,3))
# resnet_model = data_augmentation(input_layer)
# scale_layer = keras.layers.Rescaling(scale=1 / 127.5, offset=-1)
# resnet_model = scale_layer(resnet_model)

# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as you will
# learn in a few paragraphs.
resnet_model = base_model_resnet(input_layer, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
resnet_model = layers.GlobalAveragePooling2D()(resnet_model)
# A Dense classifier with a single unit (binary classification)
x = keras.layers.Dropout(0.2)(resnet_model)  # Regularize with dropout
outputs = layers.Dense(2)(resnet_model)
model = keras.Model(input_layer, outputs)
```

```
Summary of Custom ResNet50 model.

1) We setup input layer and 2) We removed top (last) layer.
```

Define parameters

```
# model.compile(optimizer=keras.optimizers.Adam(),
#               loss=keras.losses.BinaryCrossentropy(from_logits=True),
#               metrics=[keras.metrics.BinaryAccuracy()])
metrics = ['accuracy', tf.keras.metrics.Precision() , tf.keras.metrics.FalseNegatives() ,tf.keras.metrics.FalsePositives(),
          tf.keras.metrics.TruePositives(), tf.keras.metrics.TrueNegatives()]

model.compile(optimizer=keras.optimizers.Adam(0.000001),  # Very low learning rate
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=metrics)


model.fit(X_train_balanced,y_train_balanced,epochs=3,batch_size=32,verbose=True,validation_data=(xtest,ytest))
```

```
Epoch 1/3
359/359 [==============================] - 216s 590ms/step - loss: 0.9292 - accuracy: 0.3836 - precision: 0.4136 - false_negatives: 11123.0000 - false_positives: 492.0000
itives: 347.0000 - true_negatives: 10978.0000 - val_loss: 0.8843 - val_accuracy: 0.4204 - val_precision: 1.0000 - val_false_negatives: 1913.0000 - val_false_positives: 0.(
val_true_positives: 2.0000 - val_true_negatives: 1915.0000
```

b) Train with all layers

```
base_model_resnet.trainable = True

# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
# are take into account
model.compile(optimizer=keras.optimizers.Adam(0.0001),  # Very low learning rate
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=metrics)

# Train end-to-end. Be careful to stop before you overfit!
history = model.fit(X_train_balanced,y_train_balanced,epochs=9,batch_size=32,verbose=True,validation_data=(xtest,ytest))
```

4.7 VISUALIZATON

This step code is same for every classification therefore not presented here.

5 WEIGHTED RANDOM SAMPLING

In this traditional technique, classes are assigned numbers based on the amount of data they have. In my test dataset, ALL has 7272 and HEM has 2300 data.

5.1 IMPORT LIBRARIES

The necessary libraries were imported. Since the libraries with this step are the same, they are not explained one by one.

```
#Importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import os
import shutil
import cv2
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
%matplotlib inline
import operator
import random
import skimage
from skimage.io import imread, imshow, imsave
from PIL import Image

#Importing all the necessary libraries for image processing
import tensorflow as tf
from tensorflow.keras.layers import (
    BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense , MaxPool2D
)
from tensorflow.keras import layers
```

```python
import tqdm
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn import metrics

from skimage.transform import resize
#from skimage.color import grey2rgb

from keras import optimizers
from tensorflow.keras.optimizers.legacy import Adam

from keras.callbacks import Callback,ModelCheckpoint
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout
from keras.callbacks import ReduceLROnPlateau
from keras.wrappers.scikit_learn import KerasClassifier
import keras.backend as K

#import tensorflow_addons as tfa
from tensorflow.keras.metrics import Metric
#from tensorflow_addons.utils.types import AcceptableDTypes, FloatTensorLike
#from typeguard import typechecked
from typing import Optional


# Libraries for TensorFlow
from tensorflow.keras.preprocessing import image
from tensorflow.keras import models, layers

# Library for Transfer Learning
from tensorflow.keras.applications import resnet50
```

5.2 CREATE TRAIN AND TEST SET

```python
folder_all_train = '/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/all'
folder_hem_train = '/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/hem'

folder_all_test = '/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/all'
folder_hem_test = '/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/hem'

read = lambda imname: np.asarray(Image.open(imname).convert("RGB"))
```

```python
# Load in training pictures
ims_all = [read(os.path.join(folder_all_train, filename)) for filename in os.listdir(folder_all_train)]
X_all = np.array(ims_all, dtype='uint8')
ims_hem = [read(os.path.join(folder_hem_train, filename)) for filename in os.listdir(folder_hem_train)]
X_hem = np.array(ims_hem, dtype='uint8')

# Load in testing pictures
ims_all_test = [read(os.path.join(folder_all_test, filename)) for filename in os.listdir(folder_all_test)]
X_all_test = np.array(ims_all_test, dtype='uint8')
ims_hem_test = [read(os.path.join(folder_hem_test, filename)) for filename in os.listdir(folder_hem_test)]
X_hem_test = np.array(ims_hem_test, dtype='uint8')

# Create labels
y_all = np.ones(X_all.shape[0])
y_hem = np.zeros(X_hem.shape[0])

y_all_test = np.ones(X_all_test.shape[0])
y_hem_test = np.zeros(X_hem_test.shape[0])

# Merge data
X_train = np.concatenate((X_all, X_hem), axis = 0)
y_train = np.concatenate((y_all, y_hem), axis = 0)

X_test = np.concatenate((X_all_test, X_hem_test), axis = 0)
y_test = np.concatenate((y_all_test, y_hem_test), axis = 0)

# Shuffle data
s = np.arange(X_train.shape[0])
np.random.shuffle(s)
X_train = X_train[s]
y_train = y_train[s]
```

5.3 DEFINE CLASSES

```python
y_train = to_categorical(y_train, num_classes= 2)
y_test = to_categorical(y_test, num_classes= 2)
```

```python
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(9572, 210, 210, 3)
(1867, 210, 210, 3)
(9572, 2)
(1867, 2)
```

5.4 SPLIT DATA AS TRAIN AND TEST SET

```
# Train the Model

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X_train,y_train,test_size=0.2,random_state=5)
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)

print("Splitting data for train and test completed.")
```

```
(7657, 210, 210, 3)
(1915, 210, 210, 3)
(7657, 2)
(1915, 2)
Splitting data for train and test completed.
```

```
print(X_hem.shape)
print(X_all.shape)
```

```
(2300, 210, 210, 3)
(7272, 210, 210, 3)
```

5.5 MAIN PART / DEFINE CLASS WEIGHTS

HEM images are 2300, all images are 7272.Therefore assign 7 coefficient for HEM images and assign 2 coefficient for ALL class.To balanced these 2 class in this step.

```
]:  class_weights = {0: 7., 1: 2.}
```

5.6 MODEL WITH RESNET-50

```
]:
# Check properties of the model that we are going to use for Transfer Learning

print("Summary of default ResNet50 model.\n")

# we are using resnet50 for transfer learnin here. So we have imported it
from tensorflow.keras.applications import resnet50

# initializing model with weights='imagenet'i.e. we are carring its original weights
base_model_resnet=resnet50.ResNet50(weights='imagenet',input_shape=(210, 210, 3), include_top=False)

# display the summary to see the properties of the model
base_model_resnet.summary()
```

```
Summary of default ResNet50 model.

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_ker
nels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 210, 210, 3  0           []
                                )]
```

5.7 TRANSFER LEARNING

a) Freeze inner layer

```
# Modelling WITH Transfer Learning

# Here we will prepare model as per our requirements

print("Summary of Custom ResNet50 model.\n")
print("1) We setup input layer and 2) We removed top (last) layer. \n")

# let us prepare our input_layer to pass our image size. default is (224,224,3). we will change it to (100,100,3)
input_layer=layers.Input(shape=(210,210,3))
# resnet_model = data_augmentation(input_layer)
# scale_layer = keras.layers.Rescaling(scale=1 / 127.5, offset=-1)
# resnet_model = scale_layer(resnet_model)

# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as you will
# learn in a few paragraphs.
resnet_model = base_model_resnet(input_layer, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
resnet_model = layers.GlobalAveragePooling2D()(resnet_model)
# A Dense classifier with a single unit (binary classification)
x = keras.layers.Dropout(0.2)(resnet_model)  # Regularize with dropout
outputs = layers.Dense(2)(resnet_model)
model = keras.Model(input_layer, outputs)
```

```
Summary of Custom ResNet50 model.

1) We setup input layer and 2) We removed top (last) layer.
```

Define parameters

```
metrics = ['accuracy', tf.keras.metrics.Precision() , tf.keras.metrics.FalseNegatives() ,tf.keras.metrics.FalsePositives(),
          tf.keras.metrics.TruePositives(), tf.keras.metrics.TrueNegatives()]

model.compile(  loss=tf.keras.losses.BinaryCrossentropy(from_logits=True), optimizer=keras.optimizers.Adam(0.001) ,metrics = metrics
)

model.fit(xtrain,ytrain,epochs=5,batch_size=32,verbose=True,validation_data=(xtest,ytest),class_weight=class_weights)
```

```
Epoch 1/5
240/240 [==============================] - 42s 121ms/step - loss: 1.4617 - accuracy: 0.8023 - precision: 0.8376 - false_negative
s: 2339.0000 - false_positives: 1031.0000 - true_positives: 5318.0000 - true_negatives: 6626.0000 - val_loss: 0.4110 - val_accura
cy: 0.8266 - val_precision: 0.8695 - val_false_negatives: 489.0000 - val_false_positives: 214.0000 - val_true_positives: 1426.000
0 - val_true_negatives: 1701.0000
```

b) Train with all layers

```
base_model_resnet.trainable = True

# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
# are take into account
model.compile(optimizer=keras.optimizers.Adam(0.00001),  # Very low learning rate
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=metrics)

# Train end-to-end. Be careful to stop before you overfit!
history = model.fit(xtrain,ytrain,epochs=10,batch_size=32,verbose=True,validation_data=(xtest,ytest),class_weight=class_weights)
```

```
Epoch 1/10
240/240 [==============================] - 124s 336ms/step - loss: 1.1136 - accuracy: 0.8659 - precision: 0.9038 - false_negative
s: 1925.0000 - false_positives: 814.0000 - true_positives: 7647.0000 - true_negatives: 8758.0000 - val_loss: 0.2341 - val_accurac
y: 0.9107 - val_precision: 0.9356 - val_false_negatives: 230.0000 - val_false_positives: 116.0000 - val_true_positives: 1685.0000
- val_true_negatives: 1799.0000
```

5.8 VISUALIZATION

This step is totally same with others.Therefore not present here.

6 DATA AUGMENTATION AND CLASSIFICATION

6.1 IMPORT LIBRARIES

Main library for data generation : from tensorflow.keras.preprocessing.image import ImageDataGenerator

```
#Importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import os
import shutil
import cv2
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
%matplotlib inline
import operator
import random
import skimage
from skimage.io import imread, imshow, imsave
from PIL import Image

#Importing all the necessary libraries for image processing
import tensorflow as tf
from tensorflow.keras.layers import (
    BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense , MaxPool2D
)
from tensorflow.keras import layers

#from tensorflow.keras.layers import Conv2D, Flatten, Dense, Dropout, MaxPooling2D, Batchbenignization
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.callbacks import EarlyStopping, Callback , ModelCheckpoint
from tensorflow.keras.metrics import Accuracy,binary_crossentropy, FalsePositives, FalseNegatives, TruePositives, TrueNegatives
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.utils import to_categorical
```

```
import tqdm
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn import metrics

from skimage.transform import resize
#from skimage.color import grey2rgb

from keras import optimizers
from tensorflow.keras.optimizers.legacy import Adam

from keras.callbacks import Callback,ModelCheckpoint
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout
from keras.callbacks import ReduceLROnPlateau
from keras.wrappers.scikit_learn import KerasClassifier
import keras.backend as K

#import tensorflow_addons as tfa
#from tensorflow.keras.metrics import Metric
#from tensorflow_addons.utils.types import AcceptableDTypes, FloatTensorLike
#from typeguard import typechecked
from typing import Optional


# Libraries for TensorFlow
from tensorflow.keras.preprocessing import image
from tensorflow.keras import models, layers

# Library for Transfer Learning
from tensorflow.keras.applications import resnet50
from tensorflow import keras
```

6.2 CREATE TRAIN SET AND TEST SET

```
folder_all_train = '/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/all'
folder_hem_train = '/kaggle/input/dataset/train_data-20230722T190126Z-001/train_data/hem'

folder_all_test = '/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/all'
folder_hem_test = '/kaggle/input/dataset/new_data_valid-20230722T190129Z-001/new_data_valid/hem'

read = lambda imname: np.asarray(Image.open(imname).convert("RGB"))
```

```
# Load in training pictures
ims_all = [read(os.path.join(folder_all_train, filename)) for filename in os.listdir(folder_all_train)]
X_all_train = np.array(ims_all, dtype='uint8')
ims_hem = [read(os.path.join(folder_hem_train, filename)) for filename in os.listdir(folder_hem_train)]
X_hem_train = np.array(ims_hem, dtype='uint8')

# Load in testing pictures
ims_all_test = [read(os.path.join(folder_all_test, filename)) for filename in os.listdir(folder_all_test)]
X_all_test = np.array(ims_all_test, dtype='uint8')
ims_hem_test = [read(os.path.join(folder_hem_test, filename)) for filename in os.listdir(folder_hem_test)]
X_hem_test = np.array(ims_hem_test, dtype='uint8')

# Create labels
y_all_train = np.ones(X_all_train.shape[0])
y_hem_train= np.zeros(X_hem_train.shape[0])

y_all_test = np.ones(X_all_test.shape[0])
y_hem_test = np.zeros(X_hem_test.shape[0])

# Merge data
X_all = np.concatenate((X_all_train, X_all_test), axis = 0)
X_hem = np.concatenate((X_hem_train, X_hem_test), axis = 0)

y_all = np.concatenate((y_all_train, y_all_test), axis = 0)
y_hem = np.concatenate((y_hem_train, y_hem_test), axis = 0)
```

+ Code + Markdown

```
print(X_all.shape)
print(y_all.shape)
print()
print(X_hem.shape)
print(y_hem.shape)
```

6.3 DEFINE CLASSES

```
y_hem_categorical = to_categorical(y_hem)
y_all_categorical = to_categorical(y_all)
```

```
(8491, 210, 210, 3)
(8491,)

(2948, 210, 210, 3)
(2948,)
```

6.4 MAIN PART / DATA GENERATOR

```
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)

gen_data_for_hem = datagen.flow(
    X_hem,
    y_hem_categorical,
    batch_size=max_class_samples-min_class_samples,
    shuffle=True
)
```

6.5 BALANCED TWO CLASSES

```
X_balanced = np.concatenate((X_all, gen_data_for_hem.x), axis=0)
y_balanced = np.concatenate((y_all_categorical, y_all_categorical), axis=0)

# Shuffle the balanced data
shuffle_indices = np.arange(len(X_balanced))
np.random.shuffle(shuffle_indices)
X_balanced = X_balanced[shuffle_indices]
y_balanced = y_balanced[shuffle_indices]

# Train the Model

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_balanced,y_balanced,test_size=0.2,random_state=5)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

print("Splitting data for train and test completed.")
```

```
(9151, 210, 210, 3)
(2288, 210, 210, 3)
(9151, 2)
(2288, 2)
Splitting data for train and test completed.
```

6.6 MODEL WITH RESNET-50

```
# Check properties of the model that we are going to use for Transfer Learning

print("Summary of default ResNet50 model.\n")

# we are using resnet50 for transfer learnin here. So we have imported it
from tensorflow.keras.applications import resnet50

# initializing model with weights='imagenet'i.e. we are carring its original weights
base_model_resnet=resnet50.ResNet50(weights='imagenet',input_shape=(210, 210, 3), include_top=False)

# display the summary to see the properties of the model
base_model_resnet.summary()
```

```
Summary of default ResNet50 model.

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 210, 210, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 216, 216, 3)  0           ['input_1[0][0]']
```

6.7 TRANSFER LEARNING

a) Freeze inner layers

```
base_model_resnet.trainable = False
```

```
# Modelling WITH Transfer Learning

# Here we will prepare model as per our requirements

print("Summary of Custom ResNet50 model.\n")
print("1) We setup input layer and 2) We removed top (last) layer. \n")

input_layer=layers.Input(shape=(210,210,3))

# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as you will
# learn in a few paragraphs.
resnet_model = base_model_resnet(input_layer, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
resnet_model = layers.GlobalAveragePooling2D()(resnet_model)
# A Dense classifier with a single unit (binary classification)
x = keras.layers.Dropout(0.1)(resnet_model)  # Regularize with dropout
outputs = layers.Dense(2)(resnet_model)
model = keras.Model(input_layer, outputs)
```

```
Summary of Custom ResNet50 model.

1) We setup input layer and 2) We removed top (last) layer.
```

Define parametes

```
metrics = ['accuracy', tf.keras.metrics.Precision() , tf.keras.metrics.FalseNegatives() ,tf.keras.metrics.FalsePositives(),
          tf.keras.metrics.TruePositives(), tf.keras.metrics.TrueNegatives()]

model.compile(optimizer=keras.optimizers.Adam(0.000001),  # Very low learning rate
             loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
             metrics=metrics)

model.fit(X_train,y_train,epochs=5,batch_size=32,verbose=True,validation_data=(X_test,y_test))
```

```
Epoch 1/5
286/286 [==============================] - 187s 641ms/step - loss: 0.8487 - accuracy: 0.5884 - precision_5: 1.0000 - false_negatives_5: 9143.0000 - false_posit
rue_positives_5: 8.0000 - true_negatives_5: 9151.0000 - val_loss: 0.7940 - val_accuracy: 0.7255 - val_precision_5: 1.0000 - val_false_negatives_5: 2284.0000 -
0.0000e+00 - val_true_positives_5: 4.0000 - val_true_negatives_5: 2288.0000
```

b) Train with all layers

```
base_model_resnet.trainable = True

# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
# are take into account
model.compile(optimizer=keras.optimizers.Adam(0.00001),  # Very low learning rate
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True), metrics=metrics)


# Train end-to-end. Be careful to stop before you overfit!
history = model.fit(X_train,y_train,epochs=8,batch_size=32,verbose=True,validation_data=(X_test,y_test))
```

```
Epoch 1/8
286/286 [==============================] - 558s 2s/step - loss: 0.0030 - accuracy: 1.0000 - precision_5: 1.0000 - false_negativ
_positives_5: 9292.0000 - true_negatives_5: 11439.0000 - val_loss: 1.3539e-07 - val_accuracy: 1.0000 - val_precision_5: 1.0000 -
ves_5: 0.0000e+00 - val_true_positives_5: 2288.0000 - val_true_negatives_5: 2288.0000
```

6.8 VISUALIZATION

This step is totally same with all tecniqus therefore not presented here.

HARDWARE CONFIGURATION

The hardware to be used during the project are indicated in Table 1.

Table 1: Specifications of the hardware to be used in the project

Hardware	Components
Computer Model	LENOVO 82EY IdeaPad Gaming 3 15ARH05
CPU	AMD Ryzen 5 4600H with Radeon Graphics
Memory (RAM)	8GB
Motherboard	LENOVO LNVNB161216 SDK0J40700 WIN
GPU	NVIDIA GeForce GTX 1650
Storage	510 GB

Ethical Considerations of the Research

In order to evaluate an ethical study the publicly available C-NMC 2019 dataset is used during this research.

References

TensorFlow. (n.d.). Module: tf.keras | TensorFlow Core v2.4.1. [online] Available at: https://www.tensorflow.org/api_docs/python/tf/keras

TensorFlow. (n.d.). tf.keras.applications.resnet50.ResNet50 | TensorFlow Core v2.6.0. [online] Available at: https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50.

TensorFlow. (n.d.). tf.keras.preprocessing.image.ImageDataGenerator. [online] Available at: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.

TensorFlow. (n.d.). Classification on imbalanced data | TensorFlow Core. [online] Available at: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data.

TensorFlow. (n.d.). Deep Convolutional Generative Adversarial Network | TensorFlow Core. [online] Available at: https://www.tensorflow.org/tutorials/generative/dcgan.

TensorFlow. (n.d.). Introduction to graphs and tf.function | TensorFlow Core. [online] Available at: https://www.tensorflow.org/guide/intro_to_graphs.

Kaggle (2022). Kaggle: Your Home for Data Science. [online] Kaggle.com. Available at: https://www.kaggle.com/.