

Configuration Manual

MSc Research Project
Data Analytics

Mohit Teotia
Student ID: x20144041

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Mohit Teotia
Student ID:	x20144041
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	21

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:	Mohit Teotia
Date:	17th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Mohit Teotia
x20144041

1 Introduction

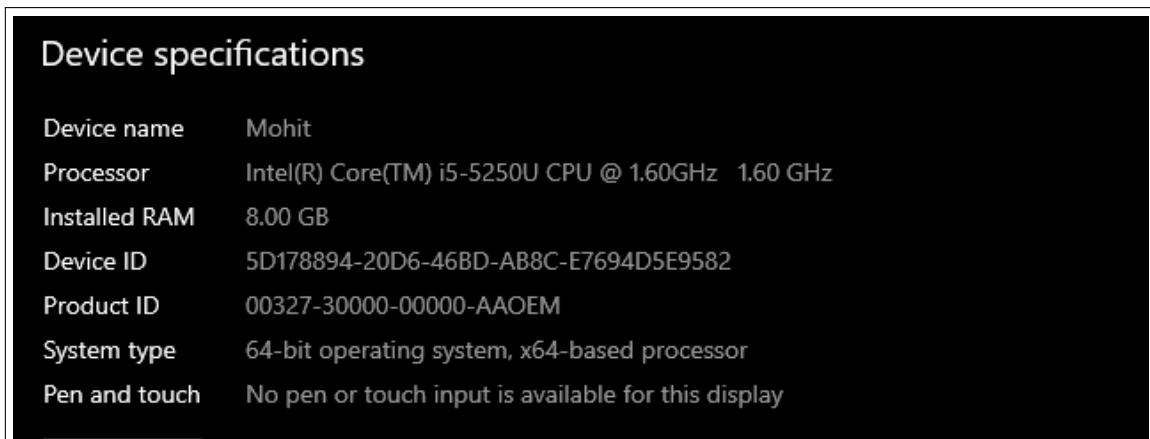
The hardware and software used in the project for the prediction of cervical cancer using deep learning architectures are described in this configuration manual. To replicate the findings and create a cervical cancer prediction model, follow the instructions in this handbook.

2 System Specification

This section will provide the hardware and software requirements necessary to carry out this project.

2.1 Hardware Requirement

Below is a discussion of the system specification used to implement all of the experiments. Figure 1 shows the hardware specification of the system.



Device specifications	
Device name	Mohit
Processor	Intel(R) Core(TM) i5-5250U CPU @ 1.60GHz 1.60 GHz
Installed RAM	8.00 GB
Device ID	5D178894-20D6-46BD-AB8C-E7694D5E9582
Product ID	00327-30000-00000-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Device Specification

- Device name: Mohit.
- System Processor: Intel(R) Core(TM) i5-5250U CPU @ 1.60GHz 1.60 GHz.
- Installed RAM in System: 8.00GB.
- System Type: 64-bit operating system, x64-based processor.

2.2 Software Requirement

Some of the basic software requirements for this project are listed below.

- Windows Edition: Windows 10.
- Scripting Language: Python 3.9.13.
- Tools: Jupyter notebook.

2.3 Resources and Specification

Figure 2 it is clearly observed the resources and their specifications used during the project.

Resources	Specification
Operating System (OS)	Windows 10
Main Memory (RAM)	8GB
Hard disk	256GB SSD and 1TB HDD
Programming Language	Python
Platform	Jupyter Notebook
Python Libraries	Numpy, Pandas, matplotlib, Tensorflow, OpenCV, Sklearn, Plotly

Figure 2: Resources and Specification

3 System Specification

3.1 Python

Python is being used to carry out this study endeavour. It has an impressive and noteworthy amount of Deep Learning and Machine Learning supporting models. It also features several libraries and some modules that aid in efficient pre-processing, image alterations, usability, and implementation. The most recent version of Python must therefore be downloaded to run the script on the PC. The software installer for the selected version can be downloaded by going to the Python websites. Figure 3 shows a screenshot of the Python version installed. Figure 4 shows the website from where you can download the desired version of python.

```
C:\Users\pc>python -V
Python 3.9.13

C:\Users\pc>
```

Figure 3: Python Version

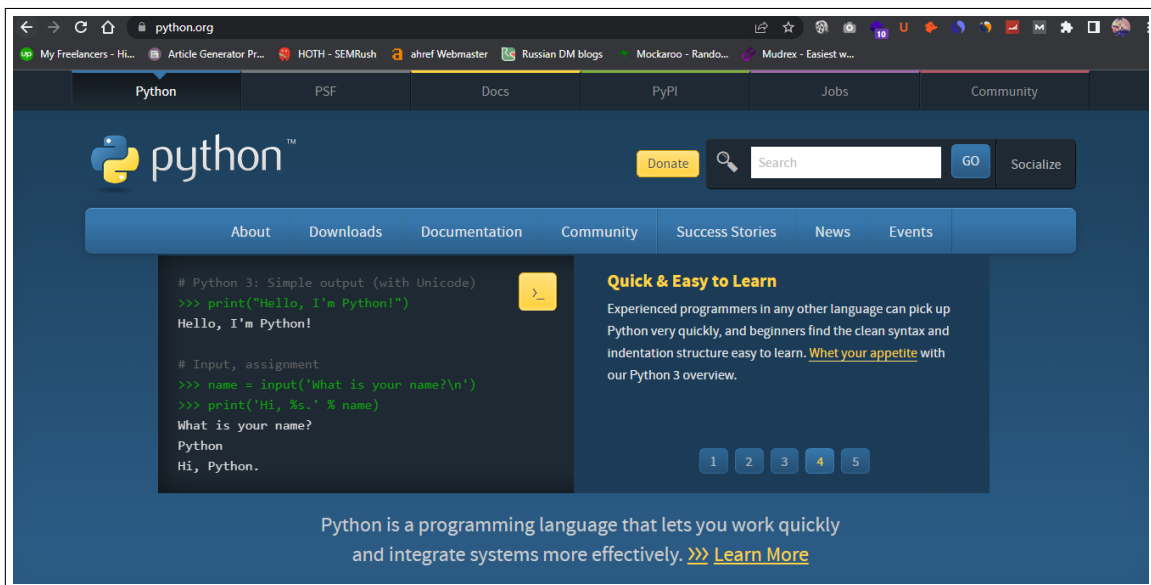


Figure 4: Python Website

3.2 Anaconda

The Anaconda package will be downloaded next. It offers a variety of intuitive Python-based IDEs that may be used for code development and result viewing. Jupyter Notebook and Spyder are the two most used IDEs that are pre-installed in Anaconda Navigator. After successfully downloading and installing Anaconda Navigator, a number of IDEs are displayed so that the developer can choose one that best suits their needs. Jupyter IDE, one of the accessible IDEs, is utilized in this study project. In figure 5 home page of the anaconda can be viewed. You can download anaconda from its official website as shown in figure 6. Anaconda is free to download and use.

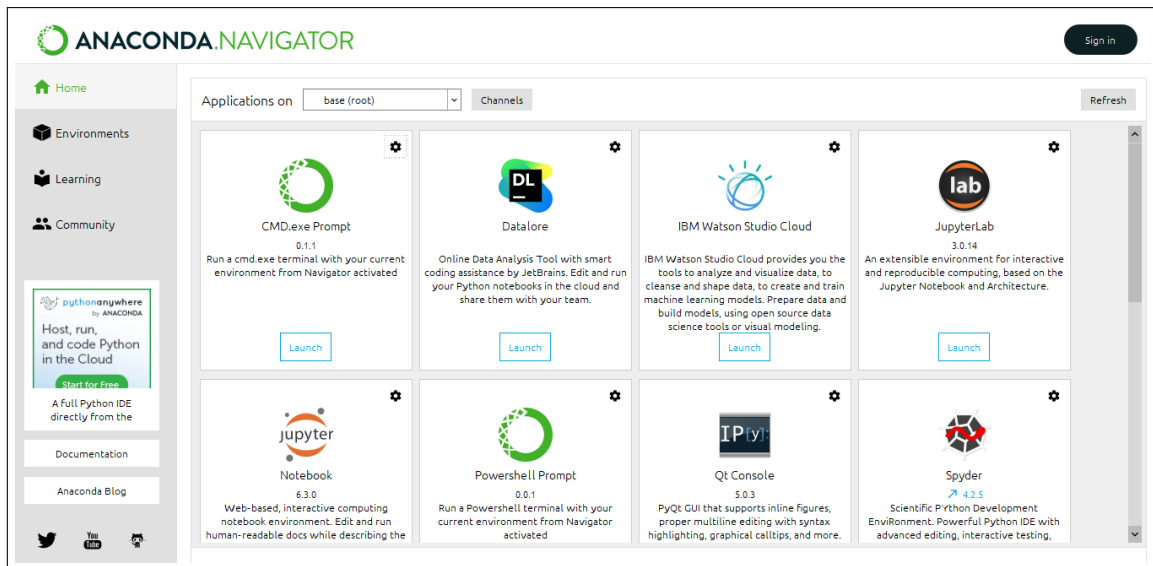


Figure 5: Anaconda

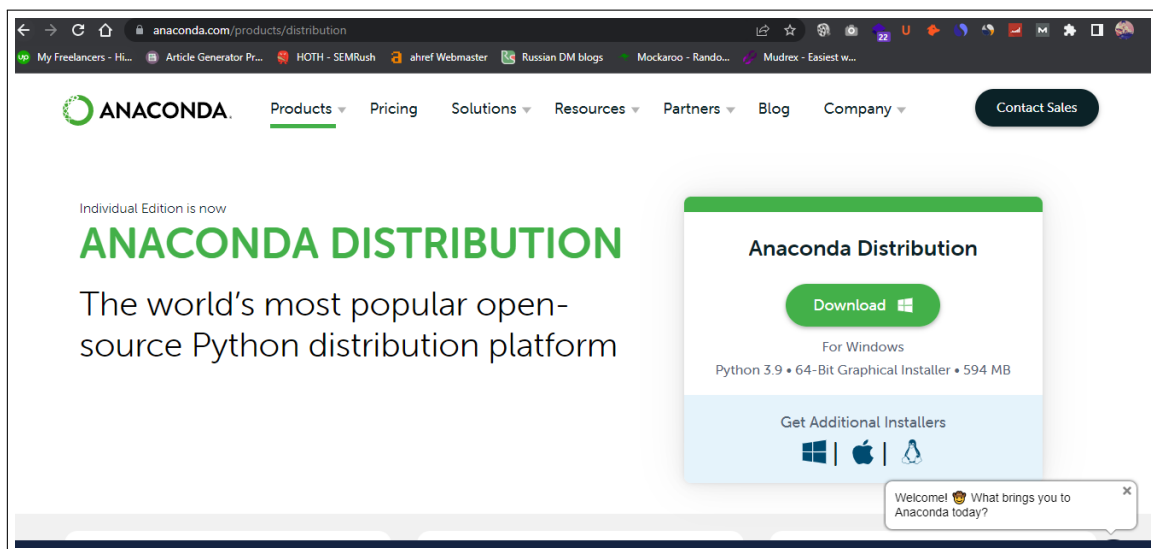


Figure 6: Anaconda Website

3.3 Data Source

The data used in this project is downloaded from GitHub shown in figure 7 which is publicly accessible by everyone. There are 93 EDF pictures in this dataset. According to the Bethesda method, these images are stacked from the slides and divided into three test categories: LSIL (Low-grade Squamous Intraepithelial Lesions), HSIL (Highgrade Squamous Intraepithelial Lesions), and negative. In this set of data, 2705 nuclei have been manually designated, and CSV files with the associated labels are included. This data distributes 16 negatives, 46 LSIL, and 31 HSIL pictures, with 238 indicated nuclei in each group and 1536, 931, and 931, respectively.

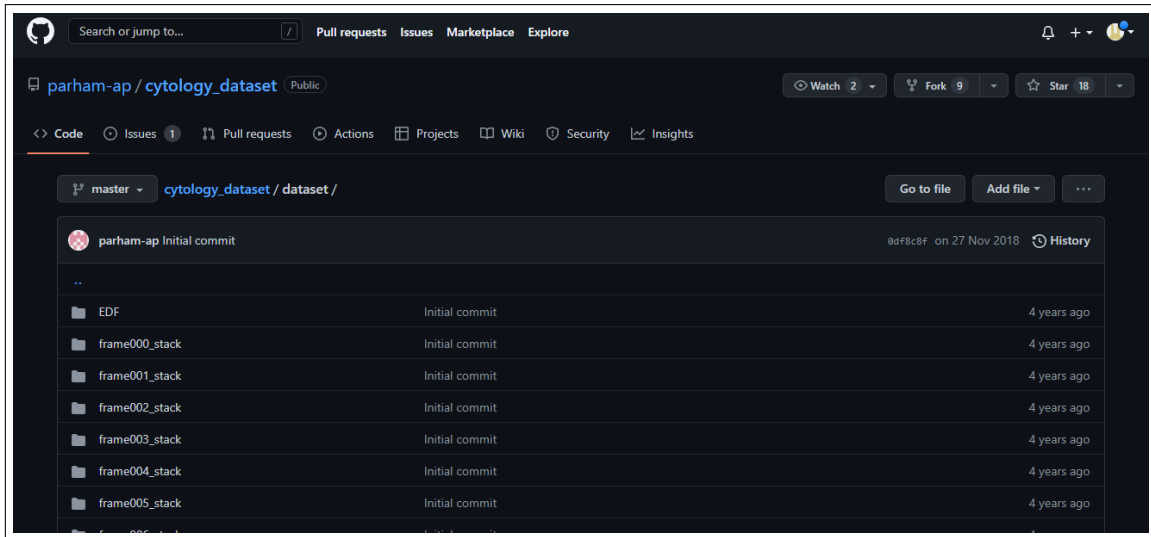


Figure 7: Data Set

4 Implementation of Cervical Cancer Prediction Project

Jupyter Notebook should be started from the installed navigator, as seen in Figure 8. A new tab in the browser is opened as you start the Jupyter IDE.



Figure 8: Jupyter Notebook Home

4.1 Step 1

The first step is to install important libraries before executing the programs that are going to be used in the models.

4.2 Step 2

Secondly, In this project import all the required libraries as shown in figure 9.

```
Importing libraries  
  
In [42]: 1 import os  
2 import cv2  
3 import numpy as np  
4 import pandas as pd  
5 import pickle  
6 import matplotlib.pyplot as plt  
7 import plotly.graph_objects as go  
8 from sklearn.preprocessing import OneHotEncoder  
9 from sklearn.model_selection import train_test_split  
10 from tqdm import tqdm  
11 import plotly.express as px  
12 import matplotlib.pyplot as plt
```

Figure 9: Libraries Used

5 Import Data

In this step of the project, import the data to perform the required tasks for the result as observed in figure 10.

```
1 # Directory where labels are stored  
2 label_dir = "dataset/labels.csv"  
3 #directory for dataset  
4 datadir = "dataset/"
```

Figure 10: Import Data

6 Data Loader

After importing data, Data Load is loaded for further processing and stored in two different arrays images and labels as shown in figure 11.

```
Data Loader  
  
1 labels_file = pd.read_csv(label_dir,index_col="frame")  
2 all_files = os.listdir(datadir)  
  
1 images = []  
2 labels = []
```

Figure 11: Data Loader

7 Iterating through all files

In this part of the project, we check all the files if they belong to CSV then we load, reshape and resize Images after this we store the images and labels as shown in figure 12.

```
1 # Iterating through all files
2 for file in all_files:
3     # check wether it is not csv file
4     if not file.endswith(".csv"):
5         # go through all images -> laod image-> reshape image
6         for image_name in os.listdir(os.path.join(datadir,file)):
7             #Load image
8             image_path = os.path.join(datadir,file) + "/" + image_name
9             img = cv2.imread(image_path)
10            # Resize Image
11            img = cv2.resize(img, (height,width))
12
13            # storing images and Labels
14            images.append(img)
15            labels.append(labels_file.loc[file[:-6]]["label"])
```

Figure 12: Load, Reshape, resize and Store

8 Converting list to NumPy Array

Now convert the list of images to a NumPy array as shown in figure 13.

```
1 # Converting list to numpy array|
2 images = np.array(images)
3 labels = np.array(labels)
```

Figure 13: Coveting list to NumPy

9 Shuffling Images and Labels

After converting the list of images to NumPy array the next step is to shuffle the images and labels as shown in figure 14.

```
1 #Shuffling indices
2 indices = np.arange(images.shape[0])
3 np.random.shuffle(indices)
4 # Shuffling images and Labels
5 images = images[indices]
6 labels = labels[indices]
7 labels_arr = labels_arr[indices]
```

Figure 14: Shuffling Images and Labels

10 Splitting Data in Categories

Now the data is split into three test categories which are LSIL (Low-grade Squamous Intraepithelial Lesions), HSIL(Highgrade Squamous Intraepithelial Lesions), and negative based on the Bethesda method. In figure 15 we can see the images which are categorised L, H, N as LSIL , HSIL and negative respectively. (Zhang et al.; 2020).In figure 16 code for this splitting of data can be seen.

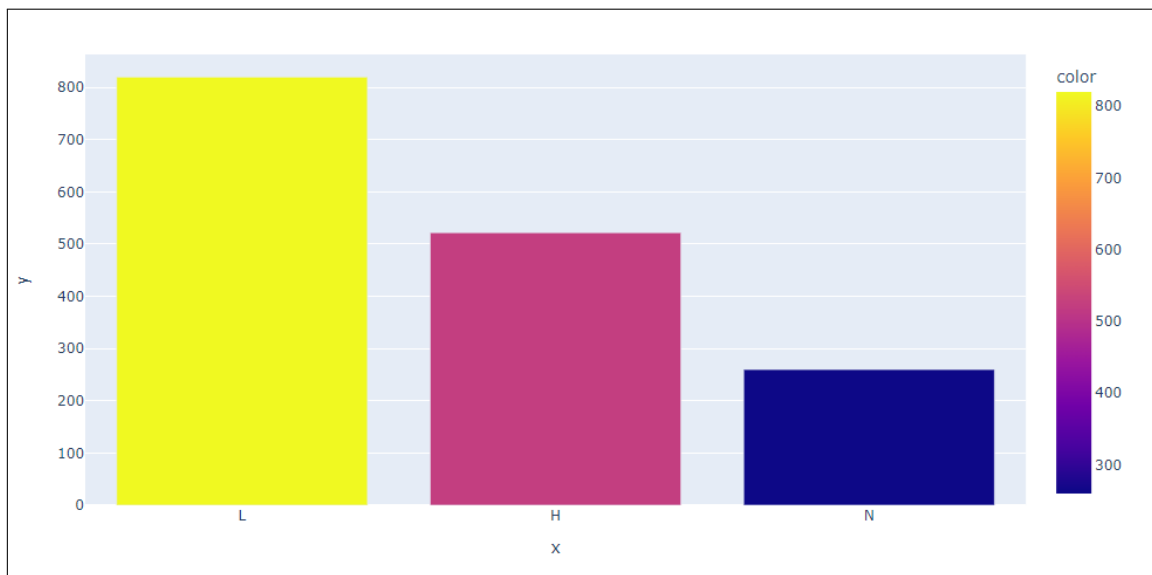


Figure 15: Data Category

```
1 label_count = pd.DataFrame(labels)[0].value_counts()
2 px.bar(x = label_count.keys(), y = label_count.values,color=label_count.values )
```

Figure 16: Data Category Code

11 Apply Gaussian Blur on SRC Image

Now we apply Gaussian blur filter on SRC images to produce an output in 3 sections that is the original image, Gaussian image and the sharp image as shown in figure 17.

```

1 img = cv2.imread(r"dataset\frame000_stack\fov000.png")
2 # apply gaussian blur on src image
3 dst = cv2.GaussianBlur(img,(5,5),cv2.BORDER_DEFAULT)
4 kernel3 = np.array([[0, 0, 0],
5                    [0, 0.9, 0],
6                    [0, 0, 0]])
7 sharp_img = cv2.filter2D(src=img, ddepth=-1, kernel=kernel3)
8 plt.figure(figsize=(20,15))
9 plt.subplot(1,3,1)
10 plt.imshow(img,)
11 plt.title("Original Image")
12 plt.subplot(1,3,2)
13 plt.imshow(dst,)
14 plt.title("Gaussian Image")
15 plt.subplot(1,3,3)
16 plt.imshow(sharp_img,)
17 plt.title("Sharp Image")
18 plt.show()

```

Figure 17: Gaussian Blur on SRC Image

After applying Gaussian blur sample data is been collect as shown in figure 18.

```

1 nrows = 3
2 ncols = 4
3
4 fig, axes = plt.subplots(nrows=nrows,ncols = ncols)
5 random_idx = np.random.randint(0,len(images),nrows*ncols)
6 fig.set_size_inches(16,12)
7
8
9 for idx,i in enumerate(random_idx):
10     image = images[i]
11
12     plt.subplot(nrows,ncols,idx+1)
13     plt.imshow(image)
14     plt.title(labels[i])
15     plt.axis('off')
16 plt.subplots_adjust(left = 0,wspace = 0.5,hspace = 0)
17 plt.suptitle("Sample Data")
18 plt.show()

```

Figure 18: Sample Data

12 data Pre-processing

Now data pre-processing is conducted for the data so that model fits in the real-world scenario (William et al.; 2018). In this step need to set the rotation range to 40, horizontal flip as true, shear range, zoom range and validation split to 0.2 respectively also rascal to 1/255. As shown in figure 19.

```

1 # Data with preprocessing
2 img_gen = tf.keras.preprocessing.image.ImageDataGenerator(rotation_range = 40,
3                                                         horizontal_flip=True,
4                                                         shear_range = 0.2,
5                                                         zoom_range = 0.2,
6                                                         rescale=1/255,
7                                                         validation_split=0.2)
8
9 # Refining train and validation datasets
10 train_data = img_gen.flow(images,labels_arr, batch_size=train_batch_size,subset='training')
11
12 val_data = img_gen.flow(images,labels_arr,batch_size=val_batch_size, subset='validation')

```

Figure 19: Data Pre-processing

13 Data Augmentation

Anyone can use data augmentation to develop faster, more accurate machine learning models while reducing their dependency on the preparation and acquisition of training data. In figure 20 it is observed the script used for data augmentation (Zhang and Liu; n.d.).

```

1 nrows = 2
2 ncols = 2
3
4 fig, axes = plt.subplots(nrows=nrows,ncols = ncols)
5 fig.set_size_inches(10,12)
6
7 temp_img = train_data.next()
8 for i in range(nrows*ncols):
9     image = temp_img[0][i]
10
11     plt.subplot(nrows,ncols,i+1)
12     plt.imshow(image)
13     plt.title(temp_img[1][i])
14     plt.axis('off')
15 plt.subplots_adjust(left = 0,wspace = 0,hspace = 0.2)
16 plt.suptitle("Data with Augmentation")
17 plt.show()

```

Figure 20: Data Augmentation Code

After applying the data augmentation over the dataset in figure 21 result can be seen.

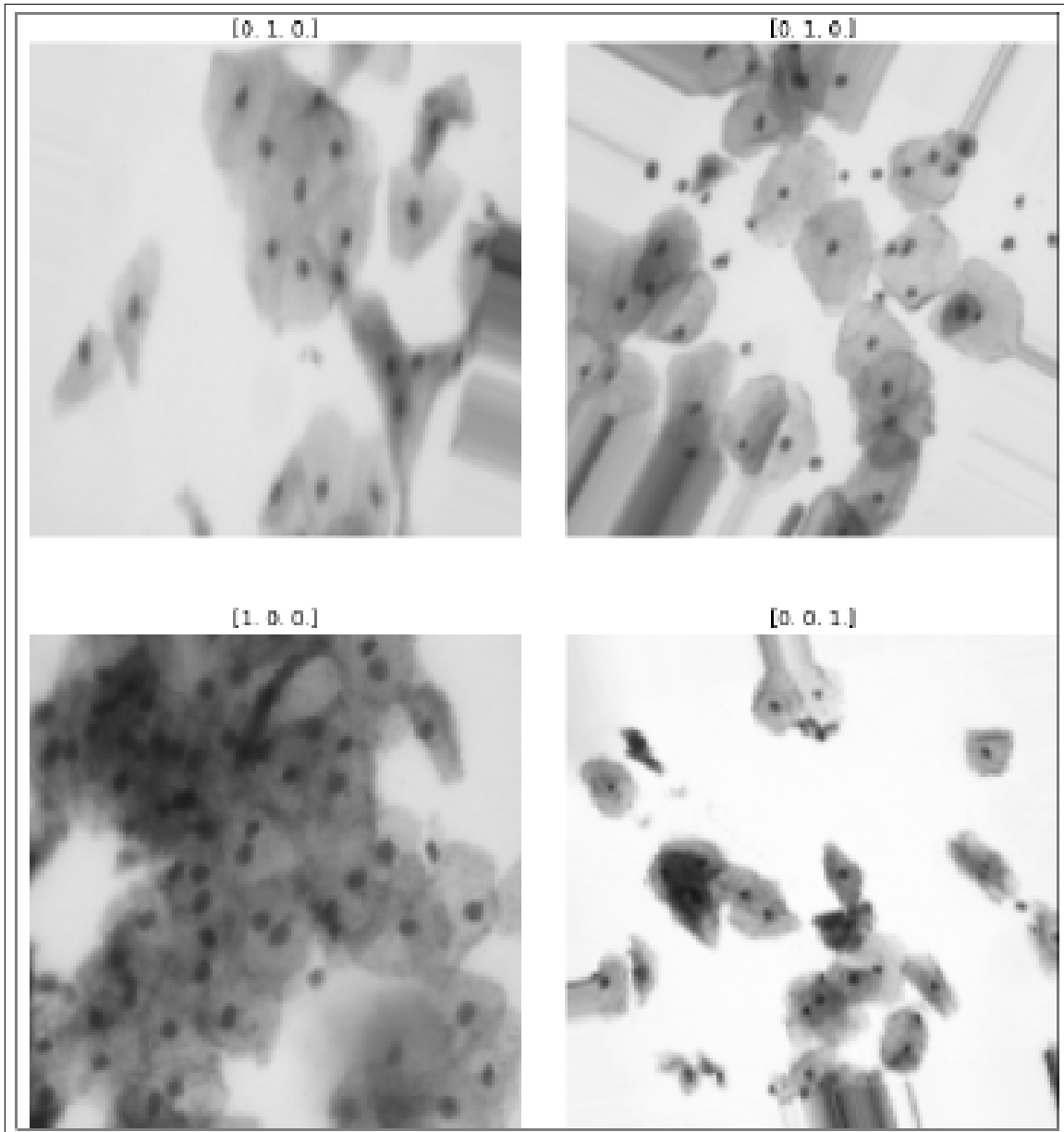


Figure 21: Augmented Data

14 Model Development & Training for Prediction of Cervical Cancer

Here all the models developed and trained in the project are explained.

14.1 InceptionV3 Model

In figure 22 and figure 23 the process of development and training of the inceptionV3 model step can be seen (Szegedy et al.; 2015). The model is trained on 50 epochs.

```

1 # Defining model
2 # We will be training from scratch
3 model = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet', input_shape=target_shape)
4 # Adding classifier layers at bottom
5 model_InceptionV3 = tf.keras.models.Sequential()
6 model_InceptionV3.add(model)
7 model_InceptionV3.add(tf.keras.layers.Flatten())
8 model_InceptionV3.add(tf.keras.layers.BatchNormalization())
9 model_InceptionV3.add(tf.keras.layers.Dense(256, activation='relu'))
10 model_InceptionV3.add(tf.keras.layers.Dropout(0.5))
11 model_InceptionV3.add(tf.keras.layers.BatchNormalization())
12 model_InceptionV3.add(tf.keras.layers.Dense(128, activation='relu'))
13 model_InceptionV3.add(tf.keras.layers.Dropout(0.5))
14 model_InceptionV3.add(tf.keras.layers.BatchNormalization())
15 model_InceptionV3.add(tf.keras.layers.Dense(3, activation='softmax'))
16
17 model_InceptionV3.layers[0].trainable=False

```

Figure 22: Inception v3 Mode Defining

```

1 # training a model
2 model_InceptionV3.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy",
3                                     tf.keras.metrics.Precision(top_k=1),
4                                     tf.keras.metrics.Recall(top_k=1)])
5
6 history_model_InceptionV3 = model_InceptionV3.fit(train_data, batch_size=train_batch_size, validation_data=val_data, epochs=50)

```

Figure 23: Inception v3 Model Training

After training of Inception V3 model, result can be evaluated in the figure 24.

```

Epoch 48/50
214/214 [=====] - 59s 273ms/step - loss: 0.5763 - accuracy: 0.7574 - precision: 0.7574 - recall: 0.7
574 - val_loss: 0.3535 - val_accuracy: 0.8625 - val_precision: 0.8625 - val_recall: 0.8625
Epoch 49/50
214/214 [=====] - 59s 274ms/step - loss: 0.6325 - accuracy: 0.7278 - precision: 0.7278 - recall: 0.7
278 - val_loss: 0.3508 - val_accuracy: 0.8656 - val_precision: 0.8656 - val_recall: 0.8656
Epoch 50/50
214/214 [=====] - 45s 210ms/step - loss: 0.6128 - accuracy: 0.7465 - precision: 0.7465 - recall: 0.7
465 - val_loss: 0.3639 - val_accuracy: 0.8875 - val_precision: 0.8875 - val_recall: 0.8875

```

Figure 24: Inception v3 Result

14.2 Custom Model

In this model, multiple customer layers are implemented to check and trained as shown in figure 25.

```

1 # Defining Custom Layers
2 custom_model = tf.keras.models.Sequential()
3 # Layer 1
4 custom_model.add(tf.keras.layers.Conv2D(64, (3, 3), padding='same', input_shape=target_shape, activation='relu'))
5 custom_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
6 custom_model.add(tf.keras.layers.BatchNormalization())
7 #Layer 2
8 custom_model.add(tf.keras.layers.Conv2D(128, (3, 3), padding='same',activation='relu'))
9 custom_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
10 custom_model.add(tf.keras.layers.BatchNormalization())
11 # Layer 3
12 custom_model.add(tf.keras.layers.Conv2D(256, (3, 3), padding='same',activation='relu'))
13 custom_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
14 custom_model.add(tf.keras.layers.BatchNormalization())
15
16 #Layer 4
17 custom_model.add(tf.keras.layers.Conv2D(256, (3, 3), padding='same',activation='relu'))
18 custom_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
19 # Nural network Layers
20 custom_model.add(tf.keras.layers.Flatten())
21
22 custom_model.add(tf.keras.layers.Dense(128, activation='relu',input_dim=128))
23 custom_model.add(tf.keras.layers.Dropout(0.3))
24 custom_model.add(tf.keras.layers.BatchNormalization())
25
26 custom_model.add(tf.keras.layers.Dense(128, activation='relu'))
27
28 custom_model.add(tf.keras.layers.Dense(3, activation='softmax'))

```

Figure 25: Custom Model

In figure 26 training of custom model can be observed.

```

1 # Training custom model
2 custom_model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",
3                                     tf.keras.metrics.Precision(top_k=1),
4                                     tf.keras.metrics.Recall(top_k=1)])
5
6 history_custom_model = custom_model.fit(train_data,batch_size=train_batch_size,validation_data=val_data,epochs=50)

```

Figure 26: Training custom model

After running for 50 epochs, In figure 27 output can be observed.

```

1: 0.6864 - val_loss: 1.0027 - val_accuracy: 0.5781 - val_precision_1: 0.5781 - val_recall_1: 0.5781
Epoch 45/50
214/214 [=====] - 113s 527ms/step - loss: 0.6399 - accuracy: 0.7106 - precision_1: 0.7106 - recall_
1: 0.7106 - val_loss: 0.4263 - val_accuracy: 0.8125 - val_precision_1: 0.8125 - val_recall_1: 0.8125
Epoch 46/50
214/214 [=====] - 112s 524ms/step - loss: 0.6426 - accuracy: 0.7083 - precision_1: 0.7083 - recall_
1: 0.7083 - val_loss: 1.2214 - val_accuracy: 0.7156 - val_precision_1: 0.7156 - val_recall_1: 0.7156
Epoch 47/50
214/214 [=====] - 122s 570ms/step - loss: 0.6071 - accuracy: 0.7262 - precision_1: 0.7262 - recall_
1: 0.7262 - val_loss: 0.6962 - val_accuracy: 0.6750 - val_precision_1: 0.6750 - val_recall_1: 0.6750
Epoch 48/50
214/214 [=====] - 125s 585ms/step - loss: 0.6028 - accuracy: 0.7332 - precision_1: 0.7332 - recall_
1: 0.7332 - val_loss: 0.4924 - val_accuracy: 0.7844 - val_precision_1: 0.7844 - val_recall_1: 0.7844
Epoch 49/50
214/214 [=====] - 123s 574ms/step - loss: 0.5975 - accuracy: 0.7457 - precision_1: 0.7457 - recall_
1: 0.7457 - val_loss: 0.3928 - val_accuracy: 0.8406 - val_precision_1: 0.8406 - val_recall_1: 0.8406
Epoch 50/50
214/214 [=====] - 115s 539ms/step - loss: 0.5998 - accuracy: 0.7379 - precision_1: 0.7379 - recall_
1: 0.7379 - val_loss: 0.3690 - val_accuracy: 0.8781 - val_precision_1: 0.8781 - val_recall_1: 0.8781

```

Figure 27: Outcome custom model

14.3 GAN Model

In this section of the project all the steps taken for the GAN model are explained.

14.3.1 select a real sample for the GAN model

As shown in figure 28 selecting the real sample from the data can be observed.

```
1 # select real samples
2 def generate_real_samples(dataset, n_samples):
3     # split into images and labels
4     images, labels = dataset
5     # choose random instances
6     ix = np.random.randint(0, images.shape[0], n_samples)
7     # select images and labels
8     X, labels = images[ix]/255., labels[ix]
9     # generate class labels
10    y = np.ones((n_samples, 1))
11    return [X, labels], y
12
```

Figure 28: Selecting Real Samples

14.3.2 Generate points in Latent Space as Input for the Generator

In figure 29 it is shown that the Points are generated in latent space which are work as input for the generator in GAN Model.

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

Figure 29: Generate Points in Latent Space

After generating inputs for the generator, a sequence of fake samples was also created with class labels and then a custom active function was created as shown in figure 30 and figure 31 respectively.

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

Figure 30: Generate Fake Samples


```

# custom activation function
def custom_activation(output):
    logexpsum = tf.keras.backend.sum(tf.keras.backend.exp(output), axis=-1, keepdims=True)
    result = logexpsum / (logexpsum + 1.0)
    return result

```

Figure 31: Custom Activation Function

Furthermore definition of the standalone supervised and unsupervised discriminator models are performed also define the standalone generator model as shown in figure 32 and figure 33 also in 34 output layer node and unsupervised output can be seen.

```

# define the standalone supervised and unsupervised discriminator models
def define_discriminator(in_shape=(128,128,3), n_classes=3):
    # image input
    in_image = tf.keras.layers.Input(shape=in_shape)
    # downsample
    fe = tf.keras.layers.Conv2D(64, (3,3), strides=(2,2), padding='same')(in_image)
    fe = tf.keras.layers.LeakyReLU(alpha=0.2)(fe)
    # downsample
    fe = tf.keras.layers.Conv2D(128, (3,3), strides=(2,2), padding='same')(fe)
    fe = tf.keras.layers.LeakyReLU(alpha=0.2)(fe)
    # downsample
    fe = tf.keras.layers.Conv2D(128, (3,3), strides=(2,2), padding='same')(fe)
    fe = tf.keras.layers.LeakyReLU(alpha=0.2)(fe)
    # downsample
    fe = tf.keras.layers.Conv2D(64, (3,3), strides=(2,2), padding='same')(fe)
    fe = tf.keras.layers.LeakyReLU(alpha=0.2)(fe)
    # flatten feature maps
    fe = tf.keras.layers.Flatten()(fe)
    # dropout
    #fe = tf.keras.layers.Dropout(0.4)(fe)

    #fe = tf.keras.layers.Dense(256)(fe)
    fe = tf.keras.layers.Dense(56)(fe)

```

Figure 32: Standalone Supervised and Unsupervised Discriminator Models

```

# output layer nodes
c_out_layer = tf.keras.layers.Dense(n_classes)(fe)
# supervised output
c_out_layer = tf.keras.layers.Activation('softmax')(c_out_layer)
# define and compile supervised discriminator model
c_model = tf.keras.models.Model(in_image, c_out_layer)
c_model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.0002, beta_1=0.5), metrics=['acc', 'tf.keras.metrics.confusion_matrix'])

# unsupervised output
d_out_layer = tf.keras.layers.Lambda(custom_activation)(fe)
# define and compile unsupervised discriminator model
d_model = tf.keras.models.Model(in_image, d_out_layer)
d_model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.0002, beta_1=0.5))
return d_model, c_model

```

Figure 33: Output Layer Node and Unsupervised Output

```

# define the standalone generator model
def define_generator(latent_dim):
    # image generator input
    in_lat = tf.keras.layers.Input(shape=(latent_dim,))
    # foundation for 4x4 image
    n_nodes = 1024 * 4 * 4
    gen = tf.keras.layers.Dense(n_nodes)(in_lat)
    gen = tf.keras.layers.LeakyReLU(alpha=0.2)(gen)
    gen = tf.keras.layers.Reshape((4, 4, 1024))(gen)
    # upsample to 8x8
    gen = tf.keras.layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')(gen)
    gen = tf.keras.layers.LeakyReLU(alpha=0.2)(gen)
    # upsample to 16x16
    gen = tf.keras.layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')(gen)
    gen = tf.keras.layers.LeakyReLU(alpha=0.2)(gen)
    # upsample to 32x32
    gen = tf.keras.layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')(gen)
    gen = tf.keras.layers.LeakyReLU(alpha=0.2)(gen)
    # upsample to 64x64
    gen = tf.keras.layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')(gen)
    gen = tf.keras.layers.LeakyReLU(alpha=0.2)(gen)
    # upsample to 128x128
    gen = tf.keras.layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same')(gen)
    gen = tf.keras.layers.LeakyReLU(alpha=0.2)(gen)
    # output
    out_layer = tf.keras.layers.Conv2D(3, (7,7), activation='tanh', padding='same')(gen)
    # define model
    model = tf.keras.models.Model(in_lat, out_layer)
    return model

```

Figure 34: Define the Standalone Generator

For updating the generator, the Discriminator and Generator models are combined as shown in figure 35.

```

# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect image output from generator as input to discriminator
    gan_output = d_model(g_model.output)
    # define gan model as taking noise and outputting a classification
    model = tf.keras.models.Model(g_model.input, gan_output)
    # compile model
    opt = tf.keras.optimizers.Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model

```

Figure 35: Combined Generator and Discriminator Model

Now generate samples and save them as a plot and save the model as shown in figure 36.

```

# generate samples and save as a plot and save the model
def summarize_performance(step, g_model, c_model, latent_dim, dataset, n_samples=16):
    # prepare fake examples
    X, _ = generate_fake_samples(g_model, latent_dim, n_samples)
    ran = np.random.randint(1,5)/100
    # scale from [-1,1] to [0,1]
    X = (X + 1) / 2.0
    # plot images
    for i in range(n_samples):
        # define subplot
        plt.subplot(4, 4, 1 + i)
        # turn off axis
        plt.axis('off')
        # plot raw pixel data
        plt.imshow(X[i, :, :, :],)

```

Figure 36: Generate Samples and Save as a Plot

After successfully combining both generator and discriminator, training of both has been performed, then calculate the number of batches per training epochs and manually enumerate epochs also perform updates on unsupervised and supervised discriminator and generator as shown in figure 37.

```

# train the generator and discriminator
def train(g_model, d_model, c_model, gan_model, train_dataset, test_dataset, latent_dim, n_epochs=50, n_batch=6):

    # calculate the number of batches per training epoch
    bat_per_epo = int(train_dataset[0].shape[0] / n_batch)

    gan_loss, gan_acc, gan_precision, gan_recall = [], [], [], []
    print('n_epochs=%d, bat_per_epo=%d' % (n_epochs, bat_per_epo))
    # manually enumerate epochs
    for epoch in tqdm(range(n_epochs), desc="EPOCHS =>"):
        for i in range(bat_per_epo):
            # update supervised discriminator (c)
            [Xsup_real, ysup_real], _ = generate_real_samples(train_dataset, n_batch)
            ran = np.random.randint(1,5)/100
            c_loss, c_acc, _, _ = c_model.train_on_batch(Xsup_real, ysup_real)
            # update unsupervised discriminator (d)
            [X_real, _], y_real = generate_real_samples(train_dataset, n_batch)
            d_loss1 = d_model.train_on_batch(X_real, y_real)
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_batch)
            d_loss2 = d_model.train_on_batch(X_fake, y_fake)
            # update generator (g)
            X_gan, y_gan = generate_latent_points(latent_dim, n_batch), np.ones((n_batch, 1))
            g_loss = gan_model.train_on_batch(X_gan, y_gan)

```

Figure 37: Training of Generator and Discriminator

In the next step create the discriminator models, create the generator, create the GAN, load image data and train the GAN model at 50 epochs and calculate the Accuracy, Precision and Recall values as shown in figure 38 and figure 39 respectively.

```

1 # create the discriminator models
2 d_model, c_model = define_discriminator(target_shape)
3 # create the generator
4 g_model = define_generator(latent_dim)
5 # create the gan
6 gan_model = define_gan(g_model, d_model)
7 # Load image data
8 train_dataset = [X_train, Y_train]
9 test_dataset = [X_test, Y_test]
10 # train model
11 gan_acc, gan_precision, gan_recall, gan_loss = train(g_model, d_model, c_model, gan_model, train_dataset, test_dataset, latent_dim)

```

Figure 38: Train GAN Model

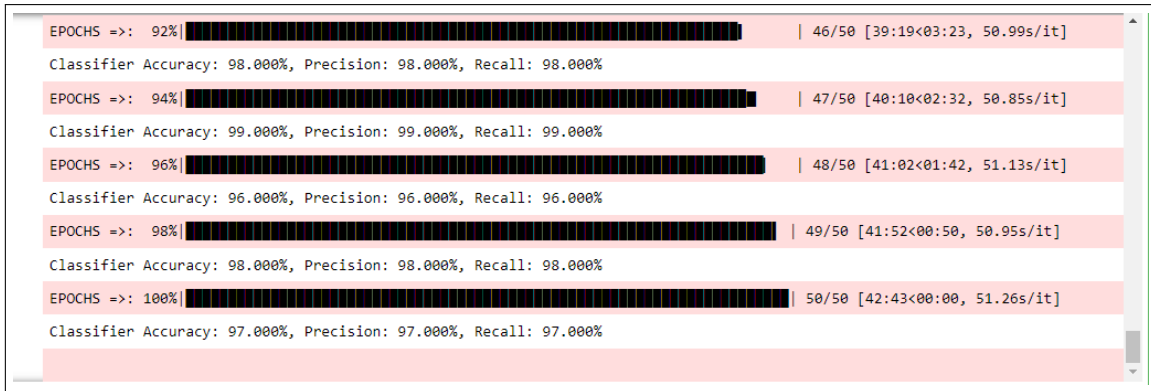


Figure 39: GAN Model Result

In figure 39 it can be observed that Accuracy, Precision and Recall is 97

15 Comparison of Models

In this part of the report, a comparison-based on accuracy, loss and Precision are done for a better understanding of the difference between models and which model generates better results.

15.1 Test Accuracy Comparison

Firstly, the comparison is done on the base of Accuracy as shown in figure 40.

```

1 fig = go.Figure(data=go.Scatter(
2     y=history_model_InceptionV3.history["val_accuracy"],
3     mode='lines+markers',
4     marker=dict(size=np.array(history_model_InceptionV3.history["val_accuracy"])*15,
5                       color=np.array(history_model_InceptionV3.history["val_accuracy"])*50),
6     name="InceptionV3"
7 ))
8
9
10 fig.add_trace(go.Scatter(
11     y=history_custom_model.history["val_accuracy"],
12     mode='lines+markers',
13     marker=dict(size=np.array(history_custom_model.history["val_accuracy"])*15,
14                       color=np.array(history_custom_model.history["val_accuracy"])*50),
15     name="Custom Model"
16 ))
17

```

Figure 40: Test Accuracy Comparison Inspection V3 and Custom Model

In above Figure 41 it can be observed the graph plot between Inspection V3 and the Custom model whereas in figure 41 graph plot for the GAN model is for accuracy comparison.

```

17
18 fig.add_trace(go.Scatter(
19     y=gan_acc,
20     mode='lines+markers',
21     marker=dict(size=np.array(gan_acc)*15,
22                       color=np.array(gan_acc)*50),
23     name="GAN Model"
24 ))
25
26 fig.update_layout(title=go.layout.Title(text="Test Accuracy Comparison",
27                                           font=go.layout.title.Font(size=25)))
28
29 fig.show()

```

Figure 41: Test Accuracy Comparison GAN Model

15.2 Test Loss Comparison

Now the comparison is done based on the Loss in the models as shown in figure 42.

```

1 fig = go.Figure(data=go.Scatter(
2     y=history_model_InceptionV3.history["val_loss"],
3     mode='lines+markers',
4     marker=dict(size=np.array(history_model_InceptionV3.history["val_loss"]),
5                       color=np.array(history_model_InceptionV3.history["val_loss"])*50),
6     name="InceptionV3"
7 ))
8
9
10 fig.add_trace(go.Scatter(
11     y=history_custom_model.history["val_loss"],
12     mode='lines+markers',
13     marker=dict(size=np.array(history_custom_model.history["val_loss"]),
14                       color=np.array(history_custom_model.history["val_loss"])*50),
15     name="Custom Model"
16 ))
17

```

Figure 42: Test Loss Comparison Inspection V3 and Custom Model

In above Figure 42, observed the graph plot between Inspection V3 and Custom model whereas in figure 43 graph plot is for the GAN model for loss comparison.

```

17
18 fig.add_trace(go.Scatter(
19     y=gan_loss,
20     mode='lines+markers',
21     marker=dict(size=np.array(gan_loss)*15,
22                       color=np.array(gan_loss)*50),
23     name="GAN Model"
24 ))
25
26
27 fig.update_layout(title=go.layout.Title(text="Test Loss Comparison",
28                                           font=go.layout.title.Font(size=25)))
29
30 fig.show()

```

Figure 43: Test Loss Comparison GAN Model

15.3 Test Precision Comparison

In figure 44 and figure 45 it is observed how comparison is done on behalf of precision.

```

1 fig = go.Figure(data=go.Scatter(
2     y=history_model_InceptionV3.history["precision"],
3     mode='lines+markers',
4     marker=dict(size=np.array(history_model_InceptionV3.history["precision"]),
5                     color=np.array(history_model_InceptionV3.history["precision"])*50),
6     name="InceptionV3"
7 ))
8
9
10 fig.add_trace(go.Scatter(
11     y=history_custom_model.history["precision_1"],
12     mode='lines+markers',
13     marker=dict(size=np.array(history_custom_model.history["precision_1"]),
14                     color=np.array(history_custom_model.history["precision_1"])*50),
15     name="Custom Model"
16 ))
17

```

Figure 44: Test Precision Comparison

```

17
18 fig.add_trace(go.Scatter(
19     y=gan_precision,
20     mode='lines+markers',
21     marker=dict(size=np.array(gan_precision)*15,
22                     color=np.array(gan_precision)*50),
23     name="GAN Model"
24 ))
25
26
27 fig.update_layout(title=go.layout.Title(text="Test Precision Comparison",
28                                         font=go.layout.title.Font(size=25)))
29
30 fig.show()

```

Figure 45: Test Precision Comparison

16 Other Software Used in Project

We used the overleaf to provide documentation for the thesis. How Overleaf can be utilized for documentation is seen in Figure 46.

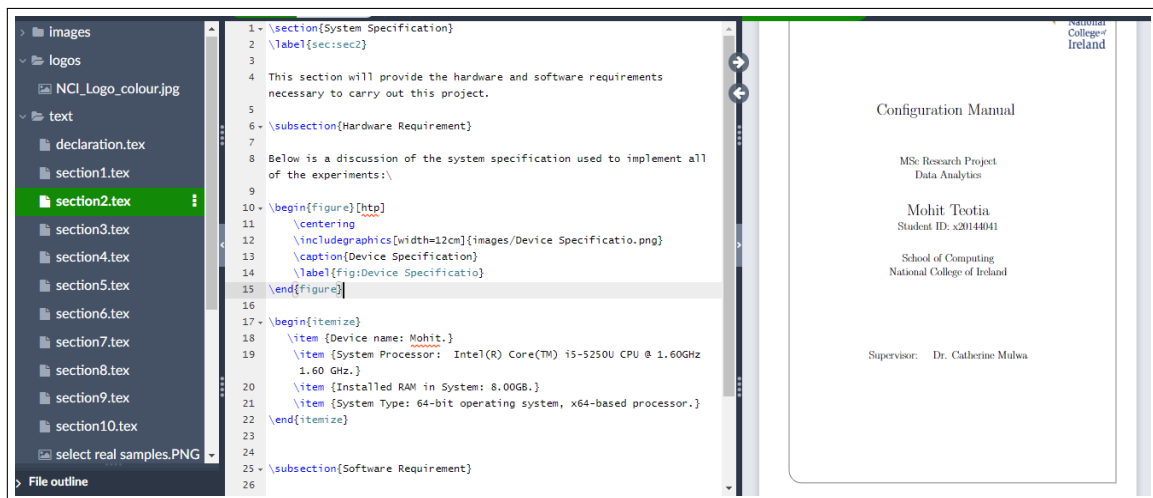


Figure 46: Overleaf

References

- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2015). Rethinking the inception architecture for computer vision.
URL: <https://arxiv.org/abs/1512.00567>
- William, W., Ware, J., Habinka, A. and Obungoloch, J. (2018). A review of image analysis and machine learning techniques for automated cervical cancer screening from pap-smear images, *Computer Methods and Programs in Biomedicine* **164**.
- Zhang, J. and Liu, Y. (n.d.). Cervical cancer detection using svm based feature screening, Vol. 3217, pp. 873–880.
- Zhang, S., , Xu, H., Zhang, L. and and, Y. Q. (2020). Cervical cancer: Epidemiology, risk factors and screening, *Chinese Journal of Cancer Research* **32**(6): 720–728.
URL: <https://doi.org/10.21147%2Fj.issn.1000-9604.2020.06.05>