

Configuration Manual

MSc Research Project
Data Analytics

Harshal Milind Tayade
Student ID: x18182763

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Harshal Milind Tayade
Student ID:	x18182763
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	28/09/2020
Project Title:	Configuration Manual
Word Count:	1319
Page Count:	14

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:	Harshal Milind Tayade
Date:	27th September 2020

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

Harshal Milind Tayade
x18182763

1 Introduction

This configuration manual provides a high level overview of the hardware and software requirements to replicate the research. This manual will prove helpful in understanding the coding steps needed to reproduce this research right from setting up the execution environment to visualizing the model results. A step-by-step guide below is divided into different sections for simplicity.

2 Hardware Requirement

The project was implemented on a Lenovo Legion Y740 laptop with the configuration details mentioned in figure 2

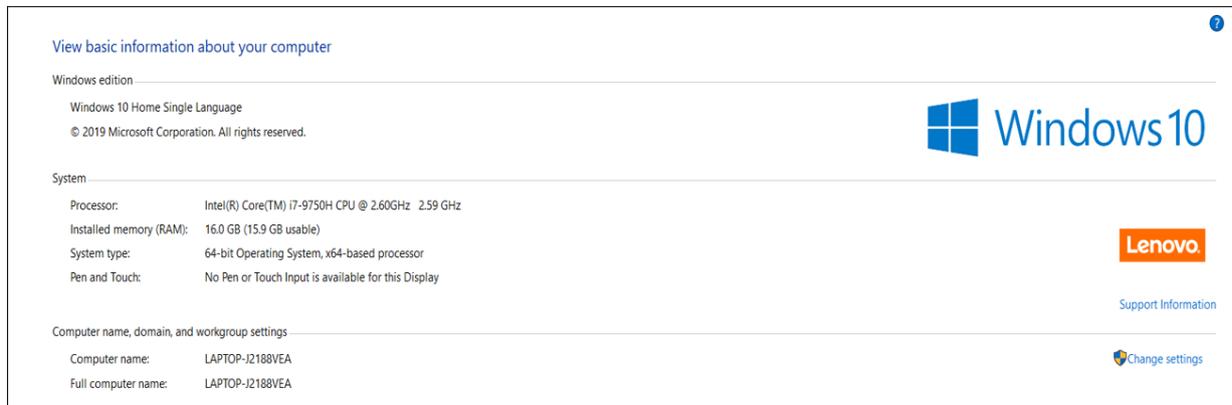


Figure 1: System Configuration

3 GPU Configuration

The project was implemented using Nvidia GeForce RTX 2060 with the configuration as shown in figure 3

4 Software Requirement

The software packages mentioned in table 1 were used in project implementation

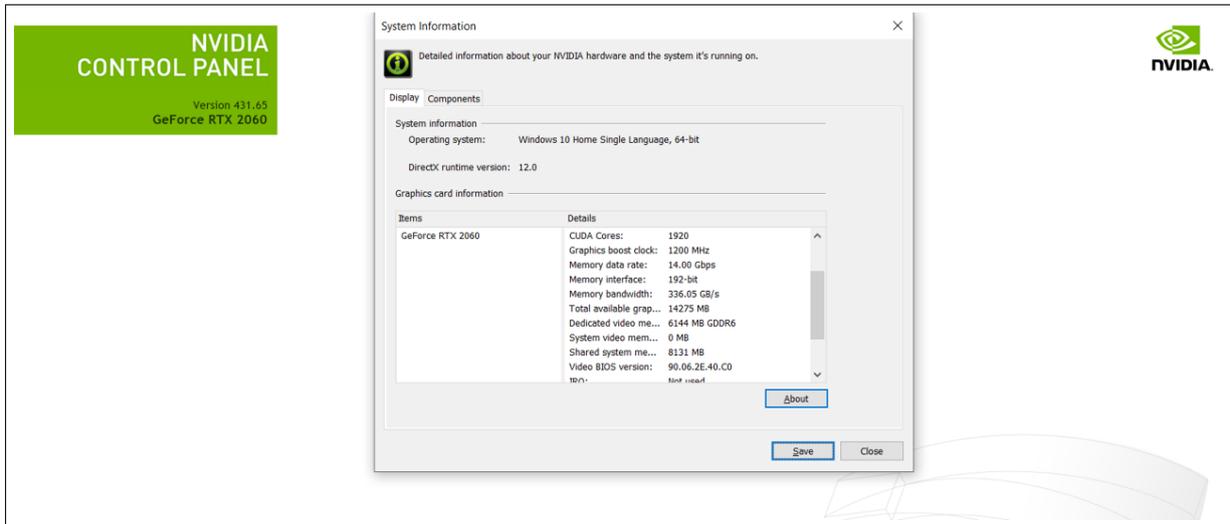


Figure 2: System Configuration

Name	Version
Anaconda Navigator	1.9.12
Jupyter Notebook	6.0.3
CMD.exe prompt	0.1.1
Python	3.7
Spyder	4.1.3
Google Chrome	84.0
Tableau	Professional Edition
Overleaf	N/A
Microsoft PowerPoint	2020 Edition

Table 1: Required Software Packages

5 Programming Environment Setup

Python programming language was used for project implementation. To achieve this we adopted the Anaconda development environment whose dashboard is shown in figure 3. Anaconda hosts bundle of applications which are suitable programming, debugging, visualization and data-mining. For our project we considered Jupyter Notebook for code development and testing. For advanced debugging spyder was used.

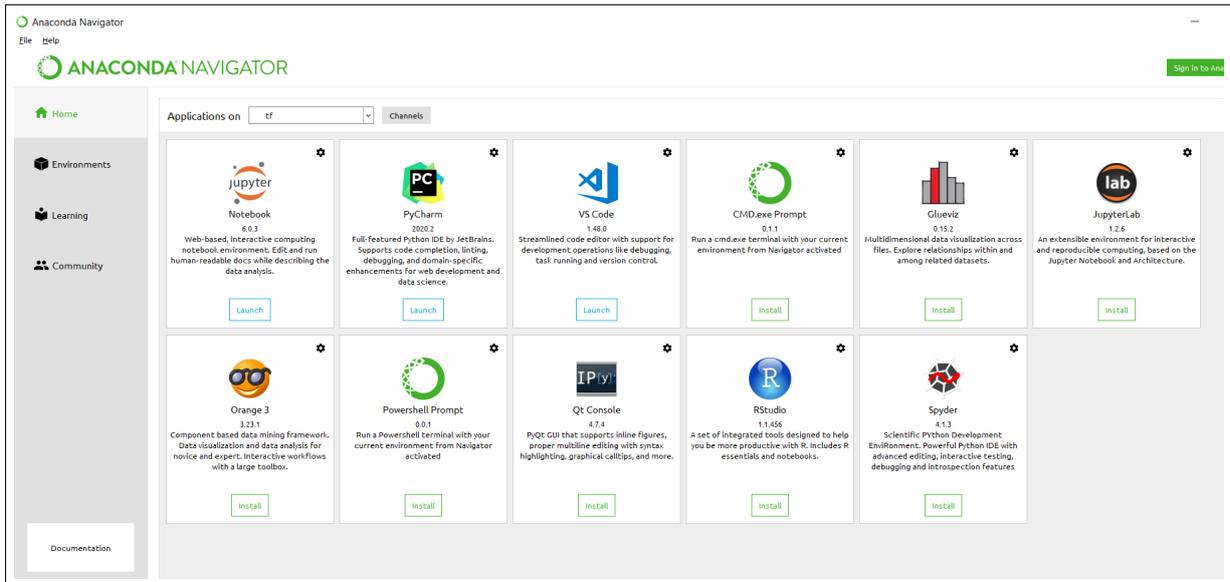


Figure 3: Anaconda Programming Environment

5.1 Steps to setup the development, testing and debugging environment

1. Download the Anaconda Navigator Individual edition from the Official site ¹

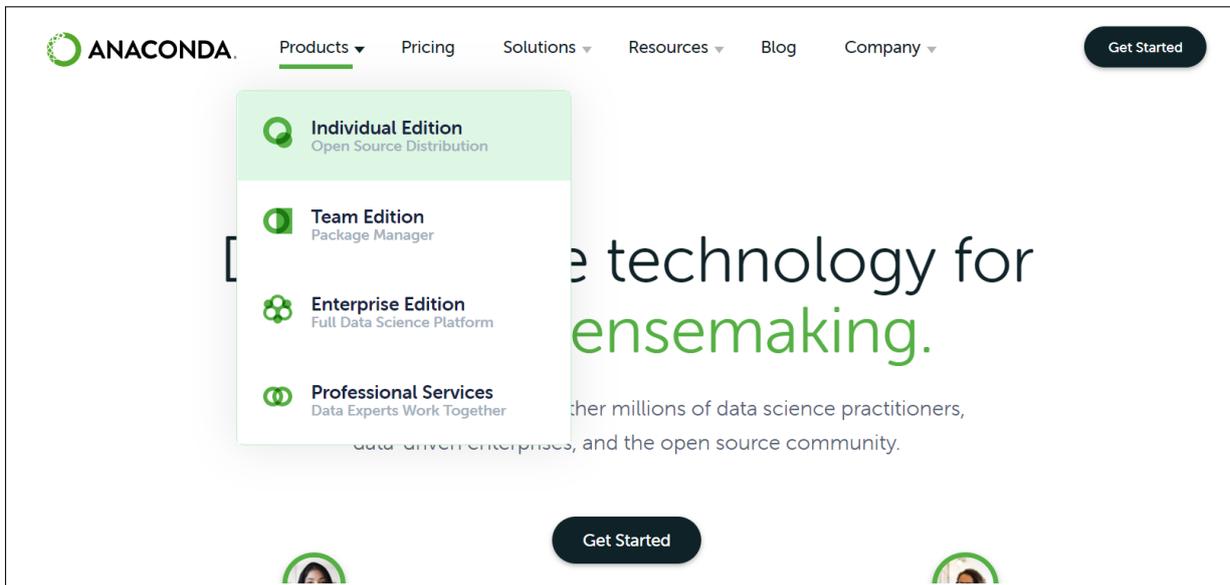


Figure 4: Anaconda Programming Environment

2. Install Jupyter Notebook and Spyder applications from the Home tab of Anaconda Navigator highlighted in figure 6

¹<https://www.anaconda.com/products/individual>

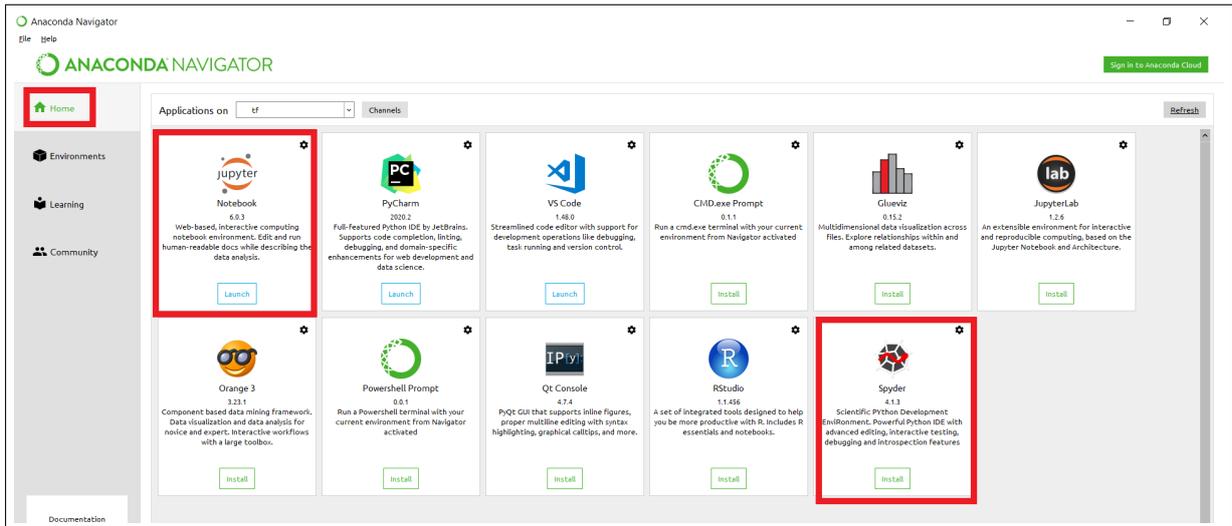


Figure 5: Installing Jupyter and Spyder

3. Install CUDA version 10.1 from the official NVIDIA developers website ² and cuDNN version 7.6.5 compatible with CUDA 10.1 from cuDNN archive ³

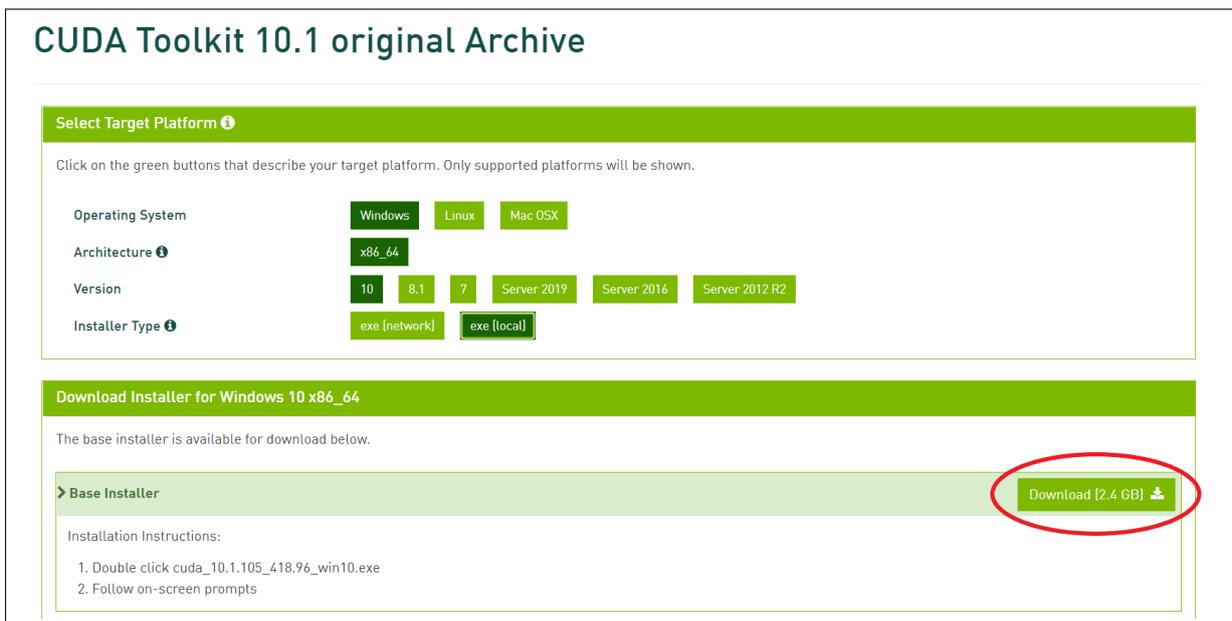


Figure 6: Nvidia CUDA 10.1

4. Install the tensorflow environment in Anaconda using CMD.EXE prompt in the Navigator window and following executing commands from the Anaconda tensorflow website. ⁴

²<https://developer.nvidia.com/cuda-10.1-download-archive>

³<https://developer.nvidia.com/rdp/cudnn-archive#a-collapse765-101>

⁴<https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/>

6 Dataset Aquisition

The research was carried out by sourcing a Zenodo website open to research dataset. This dataset consisted of different types of laryngeal tissue ranging from healthy to pre-cancerous lesions. The dataset can be downloaded from the Zenodo website ⁵ by clicking the highlighted button.



Figure 7: Dataset Download

7 Dataset Transformation

The sourced data is contained in zip file. The extracted files are structured in three folders. Each folder containing four other folders that represent different types of tissues. Figure 8 shows the original structuring.



Figure 8: Original folders of the dataset

```
import os
import shutil
import tarfile
```

```
basePath = "C:\\Users\\Harshal\\Desktop\\Project Code"
tar = tarfile.open(basePath + "\\laryngeal dataset.tar"
) tar.extractall()
```

```
tissue_classes = ['He', 'Le', 'IPCL', 'Hbv']
for class_name in tissue_classes:
    try:
        os.makedirs(basePath + "\\dataset\\" + class_name)
    except FileExistsError as exc:
        print(exc)
```

⁵<https://zenodo.org/record/1003200#.XzXb7ChKhPa>

```

folders = ["FOLD 1", "FOLD 2", "FOLD 3"]
classes = ["He", "Hbv", "Le", "IPCL"]
for folder in folders:
    for class_name in classes:
        if class_name == "He":
            new_class = classes[0]
        elif class_name == "Hbv":
            new_class = classes[1]
        elif class_name == "Le":
            new_class = classes[2]
        elif class_name == "IPCL":
            new_class = classes[3]
        try:
            for image in os.listdir(basePath + "\\laryngeal dataset\\" + folder + "\\"):
                shutil.copy2(basePath + "\\laryngeal dataset\\" + folder + "\\\" + class_name + "\\\" +
image, basePath + "\\dataset\\" + new_class)
        except FileExistsError as exc:
            print(exc)

```

These images are then restructured for binary classification into two folders. "Healthy" representing "he" folder images and "Cancerous" representing "Hbv", "IPCL" and "Le" folder images. This was automated using python script shown in the code.

8 Data Pre-Processing

The images required to be denoised using appropriate image processing technique. After analysis of literature review, we have implemented Gaussian Filtering in our code. We evaluated our proposed method by using BRISQUE image quality mertric. The coding snippets below describe the required libraries and pre-processing steps.

```

import shutil
import cv2
import os
import glob
import Augmentor
import numpy as np
import time
import matplotlib.pyplot as plt
import imquality.brisque as brisque
import matplotlib.pyplot as plt
import matplotlib.image as img
from PIL import Image, ImageFilter
from matplotlib import pyplot as plt
from pylab import array, plot, show, axis, arange, figure, uint8
from skimage import io, img_as_float
from skimage.restoration import denoise_nl_means, estimate_sigma
from skimage import data, img_as_float
from skimage import io, img_as_float
from sklearn.datasets import load_files
from glob import glob
from os import listdir
from os.path import isfile, join

```

```

import glob
os.chdir(basePath + "\\dataset\\")
for file in glob.glob('*.*.png', recursive=True):
    img = cv2.imread(file)
    dst = cv2.GaussianBlur(img, (5,5), cv2.BORDER_DEFAULT)
    filename = file
    cv2.imwrite(filename, dst)

```

```

print("BRISQUE score of Gaussian Smoothing : ",brisque.score(dst))

```

9 Data Augmentation

As the data is from biomedical domain the challenge of smaller dataset size had to be overcome. This was solved using appropriate data augmentation techniques which included different geometric transformations shown in the below coding snippet.

```
import shutil
import cv2
import glob
import Augmentor
import numpy as np
```

```
he_images = Augmentor.Pipeline(basePath + "\\dataset\\He\\")
ipcl_images = Augmentor.Pipeline(basePath + "\\dataset\\IPCL\\")
le_images = Augmentor.Pipeline(basePath + "\\dataset\\Le\\")
hbv_images = Augmentor.Pipeline(basePath + "\\dataset\\Hbv\\")

he_images.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
ipcl_images.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
le_images.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)

he_images.zoom(probability=0.5, min_factor=1.1, max_factor=1.5)
ipcl_images.zoom(probability=0.5, min_factor=1.1, max_factor=1.5)
le_images.zoom(probability=0.5, min_factor=1.1, max_factor=1.5)

he_images.flip_left_right(probability=0.5)
ipcl_images.flip_left_right(probability=0.5)
le_images.flip_left_right(probability=0.5)

he_images.flip_top_bottom(probability=0.5)
ipcl_images.flip_top_bottom(probability=0.5)
le_images.flip_top_bottom(probability=0.5)

hbv_images.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
hbv_images.zoom(probability=0.5, min_factor=1.1, max_factor=1.5)
hbv_images.flip_left_right(probability=0.5)
hbv_images.flip_top_bottom(probability=0.5)
```

```

Executing Pipeline: 0% | 0/3000 [00:00<?, ? Samples/s]

Initialised with 330 image(s) found.
Output directory set to C:\Users\Harshal\Desktop\Project Code\dataset\He\output.Initialised with 330 image(s) found.
Output directory set to C:\Users\Harshal\Desktop\Project Code\dataset\IPCL\output.Initialised with 330 image(s) found.
Output directory set to C:\Users\Harshal\Desktop\Project Code\dataset\Le\output.Initialised with 330 image(s) found.
Output directory set to C:\Users\Harshal\Desktop\Project Code\dataset\Hbv\output.

Processing <PIL.Image.Image image mode=RGB size=101x101 at 0x26AC2E9C948>: 100% ██████████ 3000/3000 [00:05<00:00, 579.92 Samp
les/s]
Processing <PIL.Image.Image image mode=RGB size=101x101 at 0x26AC3546188>: 100% ██████████ 3000/3000 [00:05<00:00, 598.19 Samp
les/s]
Processing <PIL.Image.Image image mode=RGB size=101x101 at 0x26AC27E1EC8>: 100% ██████████ 3000/3000 [00:05<00:00, 569.70 Samp
les/s]
Processing <PIL.Image.Image image mode=RGB size=101x101 at 0x26AC1C9AFC8>: 100% ██████████ 3000/3000 [00:05<00:00, 555.50 Samp
les/s]

```

Figure 9: Data Augmentation Pipeline Output

10 Data Modelling

All the models are designed and implemented using keras library for deep learning and Anaconda Jupyter Notebook. For implementing the two baseline models i.e. Convolutional Neural Network and DenseNet121 based transfer learning models we have created three folders named train, validate and test as shown in the below code.

```
dataset_dir = "\\content\\data\\"
train_dir = "\\content\\train\\"
val_dir = "\\content\\val\\"
test_dir = "\\content\\test\\"
train_ratio = 0.8
val_ratio = 0.1
test_ratio = 0.1
```

10.1 Baseline 1 - CNN model

The CNN model was designed from scratch using the configuration shown in below snippet. The libraries required for the model implementation is also shown in the following code snippet.

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard
import tensorflow

```

```

classifier = Sequential()
classifier.add(Conv2D(32, (3,3), input_shape=(100,100,3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2), strides=2)) # if stride not given it equal to pool filter size
classifier.add(Conv2D(32, (3,3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2), strides=2))
classifier.add(Flatten())
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dense(units=2, activation='softmax'))

```

The model was trained using different hyper-parameters which are included in the code below.

The CNN model was evaluated on different metrics. The evaluation code snippet along with visualized results are shown in figure 10

```

from keras.optimizers import SGD
opt = SGD(lr=0.001)
classifier.compile(loss = "categorical_crossentropy", optimizer = opt, metrics=['accuracy'])

```

```

from sklearn.metrics import classification_report, confusion_matrix
#Confusion Matrix and Classification Report
Y_pred = classifier.predict_generator(val_set, steps = 4950)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(val_set.classes, y_pred))
print('Classification Report')
target_names = ['Healthy', 'Cancerous']
print(classification_report(val_set.classes, y_pred, target_names=target_names))

cm = confusion_matrix(val_set.classes, y_pred) total=sum(sum(cm))

#from confusion matrix calculate accuracy accuracy1=(cm[0,0]+cm[1,1])/total
print ('Accuracy : ', accuracy1)

sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
print('Sensitivity : ', sensitivity1)

specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])
print('Specificity : ', specificity1)

```

```

WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 4950 batches). You may need to use the repeat() function when building your dataset.
Confusion Matrix
[[939  51]
 [154 836]]
Classification Report

```

	precision	recall	f1-score	support
Healthy	0.86	0.95	0.90	990
Unhealthy	0.94	0.84	0.89	990
accuracy			0.90	1980
macro avg	0.90	0.90	0.90	1980
weighted avg	0.90	0.90	0.90	1980

```

Accuracy : 0.8964646464646465
Sensitivity : 0.9484848484848485
Specificity : 0.8444444444444444

```

Figure 10: CNN Model Evaluation

10.2 Baseline 2 - Dense-Net 121 model

The transfer learning Dense-Net 121 model was designed using the keras api library ⁶. The model is trained on "ImageNet" dataset. The library and configuration parameters is shown in figure 11

```
from keras.application import DenseNet121
```

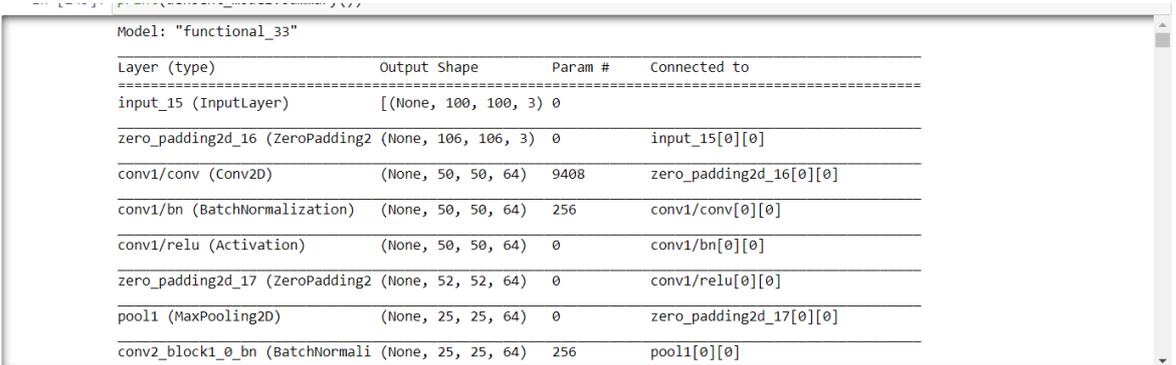
```
base_model = DenseNet121(weights='imagenet',include_top=False, input_shape=(100,100,3)) #imports the INception model and discards the last 1000 neuron layer.
```

```
x = base_model.output  
x = Flatten(name="Flatten")(x)  
x = Dropout(0.5)(x)  
preds = Dense(2,activation = 'softmax')(x) #final layer with softmax activation
```

```
densent_model = Model(inputs = base_model.input,outputs = preds)  
#specify the inputs  
#specify the outputs  
#now a model has been created based on our architecture
```

```
for layer in densent_model.layers[:-10]:  
    layer.trainable = True
```

```
print(densent_model.summary())
```



Model: "functional_33"

Layer (type)	Output Shape	Param #	Connected to
input_15 (InputLayer)	[(None, 100, 100, 3)]	0	
zero_padding2d_16 (ZeroPadding2D)	(None, 106, 106, 3)	0	input_15[0][0]
conv1/conv (Conv2D)	(None, 50, 50, 64)	9408	zero_padding2d_16[0][0]
conv1/bn (BatchNormalization)	(None, 50, 50, 64)	256	conv1/conv[0][0]
conv1/relu (Activation)	(None, 50, 50, 64)	0	conv1/bn[0][0]
zero_padding2d_17 (ZeroPadding2D)	(None, 52, 52, 64)	0	conv1/relu[0][0]
pool1 (MaxPooling2D)	(None, 25, 25, 64)	0	zero_padding2d_17[0][0]
conv2_block1_0_bn (BatchNormali	(None, 25, 25, 64)	256	pool1[0][0]

Figure 11: Dense-Net 121 model Configuration

```
from keras.optimizers import SGD  
opt = SGD(lr=0.001)  
densent_model.compile(loss = "categorical_crossentropy", optimizer = opt, metrics=['accuracy'])
```

```
fit_history = densent_model.fit_generator(train_set,  
    steps_per_epoch=step_size_train,  
    epochs = 10,  
    validation_data = val_set,  
    validation_steps = step_size_val)
```

⁶<https://keras.io/api/applications/densenet/>

```

Epoch 1/10
990/990 [=====] - 123s 124ms/step - loss: 0.4284 - accuracy: 0.8860 - val_loss: 0.8234 - val_accuracy:
0.6667
Epoch 2/10
990/990 [=====] - 121s 122ms/step - loss: 0.0367 - accuracy: 0.9905 - val_loss: 0.4999 - val_accuracy:
0.7825
Epoch 3/10
990/990 [=====] - 121s 123ms/step - loss: 0.0163 - accuracy: 0.9964 - val_loss: 0.6517 - val_accuracy:
0.6870
Epoch 4/10
990/990 [=====] - 121s 122ms/step - loss: 0.0046 - accuracy: 0.9984 - val_loss: 0.5767 - val_accuracy:
0.7302
Epoch 5/10
990/990 [=====] - 121s 122ms/step - loss: 0.0019 - accuracy: 0.9994 - val_loss: 0.5330 - val_accuracy:
0.7576
Epoch 6/10
990/990 [=====] - 122s 123ms/step - loss: 7.7041e-04 - accuracy: 0.9998 - val_loss: 0.5961 - val_accu
acy: 0.7373

```

Figure 12: Dense-Net 121 model Training

The Dense-Net 121 model was trained based on the hyper parameters shown in the code snippet 12. Every epoch monitors the accuracy and loss for training and validation set.

```

from sklearn.metrics import classification_report, confusion_matrix

#Confution Matrix and Classification Report
Y_pred = densent_model.predict_generator(val_set, steps = 4950)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(val_set.classes, y_pred))
print('Classification Report')
target_names = ['Healthy', 'Unhealthy']
print(classification_report(val_set.classes, y_pred, target_names=target_names))

cm = confusion_matrix(val_set.classes, y_pred)
total = sum(sum(cm))

#from confusion matrix calculate accuracy
accuracy1 = (cm[0,0] + cm[1,1]) / total
print('Accuracy : ', accuracy1)

sensitivity1 = cm[0,0] / (cm[0,0] + cm[0,1])
print('Sensitivity : ', sensitivity1)

specificity1 = cm[1,1] / (cm[1,0] + cm[1,1])
print('Specificity : ', specificity1)

```

```

WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at
least `steps_per_epoch * epochs` batches (in this case, 4950 batches). You may need to use the repeat() function when building
your dataset.
Confusion Matrix
[[828 162]
 [327 663]]
Classification Report

```

	precision	recall	f1-score	support
Healthy	0.72	0.84	0.77	990
Unhealthy	0.80	0.67	0.73	990
accuracy			0.75	1980
macro avg	0.76	0.75	0.75	1980
weighted avg	0.76	0.75	0.75	1980

```

Accuracy : 0.753030303030303
Sensitivity : 0.8363636363636363
Specificity : 0.6696969696969697

```

Figure 13: Dense-Net 121 model Evaluation

The Dense-Net 121 is evaluated using the code snippet shown in figure 13

10.3 Attention-based Multiple Instance Learning

This section explains the implementation for our novel method in detection of laryngeal cancer. We have designed this code by referring to (Ilse et al.; 2018; Wang et al.; 2018). The code is arranged using modular design using functions and classes wherever appropriate. This will help the user in reproducing the results in easy manner. The main class contains the basepath of the dataset and K-fold cross validation parameters. The main function calls two methods. First method loads the data and second method trains, tests and returns the model evaluation. The snippet below summarizes the main method.

```
import numpy as np

if __name__ == "__main__":
    input_dimensions = (64,64,3)
    run = 1
    n_folds = 3
    acc = np.zeros((run, n_folds), dtype=float)
    data_path = "E:\\Laryngeal Dataset\\MIL-dataset"
    for irun in range(run):
        dataset=load_dataset(dataset_path=data_path, n_folds=n_folds, rand_state=irun)
        for ifold in range(n_folds):
            print ('run=', irun, ' fold=', ifold)
            accuracy[irun][ifold] = model_training(input_dimensions, dataset[ifold], irun, ifold)
    print ('minet mean accuracy = ', np.mean(accuracy))
    print ('std = ', np.std(accuracy))
```

The coding snippet denote the libraries that were used for implementation of our novel MIL technique. They comprise of standard python libraries, keras libraries, file handling libraries and visualization libraries.

```
import os
import cv2
import sys
import time
import glob
import random
import imageio
import argparse
import threading
import numpy as np
import scipy.misc as sci
import tensorflow as tf
import matplotlib.pyplot as plt
from random import shuffle
from keras import backend as K
from keras.models import Model
from keras.layers import Layer
from keras.optimizers import SGD,Adam
from keras.regularizers import l2
from keras.utils import multi_gpu_model
from keras import activations, initializers, regularizers
from keras.layers import Input, Dense, Layer, Dropout, Conv2D, MaxPooling2D, Flatten, multiply
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping
import keras
from sklearn.model_selection import KFold
```

Based on the K-fold cross validation the data from Healthy and Cancerous image folders are shuffles into train and test set. Different images in each set during every fold execution shown in the following code. Here we have used 3-fold cross validation.

```
def load_data(dataset_path, n_folds, random_state):

    p_path = glob.glob(dataset_path+"\\Healthy\\*.png")
    n_path = glob.glob(dataset_path+"\\Cancerous\\*.png")

    p_num = len(p_path)
    n_num = len(n_path)

    all_path = p_path + n_path
    kf = KFold(n_splits=n_folds, shuffle=True, random_state=random_state)
    datasets = []
    for train_idx, test_idx in kf.split(all_path):
        dataset =
        dataset['train'] = [all_path[ibag] for ibag in train_idx]
        dataset['test'] = [all_path[ibag] for ibag in test_idx]
        datasets.append(dataset)
    return datasets
```

The model_training() function in the depicted code snippet handles the code execution

from creating train and test bags, batch generation for each phase, model training and testing to result generation.

```
def model_training(input_dim, dataset, irun, ifold):

    train_bags = dataset['train']
    test_bags = dataset['test']

    # convert bag to batch
    train_set = generate_batch(train_bags)
    test_set = generate_batch(test_bags)
    model = cell_network(input_dim, useMulGpu=False)
    # train model
    t1 = time.time()
    num_batch = len(train_set)
    #for epoch in range(10):
    model_name = train_eval(model, train_set, irun, ifold)
    print("load saved model weights")
    model.load_weights(model_name)
    test_loss, test_acc = test_eval(model, test_set)
    t2 = time.time()
    print ('run time:', (t2 - t1) / 60.0, 'min')
    print ('test_acc:'.3f'.format(test_acc))

    return test_acc
```

We create image bags for training based on the class label of the image. The function in the below code returns bag of image containing image data and labels.

```
def generate_batch(path):
    bags = []
    for each_path in path:
        name_img = []
        img = []
        img_path = glob.glob(each_path)
        num_ins = len(img_path)
        label = each_path.split('\\')[-2]
        if label == 'Healthy':
            curr_label = np.ones(num_ins, dtype=np.uint8)
        else:
            curr_label = np.zeros(num_ins, dtype=np.uint8)
        for each_img in img_path:
            img_data = cv2.imread(each_img, cv2.IMREAD_UNCHANGED).astype(float)
            scale_percent = 50
            width = 64
            height = 64
            dsize = (width, height)
            #resize image
            img_data = cv2.resize(img_data, dsize)
            img_data[:, :, 0] -= 123.68
            img_data[:, :, 1] -= 116.779
            img_data[:, :, 2] -= 103.939
            img_data /= 255
            img.append(np.expand_dims(img_data, 0))
            name_img.append(each_img.split('\\')[-1])
        stack_img = np.concatenate(img, axis=0)
        bags.append((stack_img, curr_label, name_img))
    return bags
```

```

def cell_net(input_dim, useMulGpu=False):

    lr = 1e-2
    weight_decay = 0.005
    momentum = 0.9

    data_input = Input(shape=input_dim, dtype='float32', name='input')
    conv1 = Conv2D(36, kernel_size=(4,4), kernel_regularizer=l2(weight_decay), activation='relu')(data_input)
    conv1 = MaxPooling2D((2,2))(conv1)

    conv2 = Conv2D(48, kernel_size=(3,3), kernel_regularizer=l2(weight_decay), activation='relu')(conv1)
    conv2 = MaxPooling2D((2,2))(conv2)
    x = Flatten()(conv2)

    fc1 = Dense(512, activation='relu', kernel_regularizer=l2(weight_decay), name='fc1')(x)
    fc1 = Dropout(0.5)(fc1)
    fc2 = Dense(512, activation='relu', kernel_regularizer=l2(weight_decay), name='fc2')(fc1)
    fc2 = Dropout(0.5)(fc2)

    alpha = MilAttention(L_dim=128, output_dim=1, kernel_regularizer=l2(weight_decay), name='alpha',
                        use_gated=False)(fc2)
    x_mul = multiply([alpha, fc2])

    out = LastSigmoid(output_dim=1, name='FC1.sigmoid')(x_mul)

    model = Model(inputs=[data_input], outputs=[out])

    if useMulGpu == True:
        parallel_model = multi_gpu_model(model, gpus=2)
        parallel_model.compile(optimizer=Adam(lr=lr, beta_1=0.9, beta_2=0.999), loss=bag_loss,
                              metrics=[bag_accuracy,
                                      tf.keras.metrics.TruePositives(), tf.keras.metrics.FalsePositives(),
                                      tf.keras.metrics.FalseNegatives()])
    else:
        model.compile(optimizer=Adam(lr=lr, beta_1=0.9, beta_2=0.999), loss=bag_loss, metrics=[bag_accuracy,
                                                bag_loss],
                    tf.keras.metrics.TruePositives(), tf.keras.metrics.FalsePositives(),
                    tf.keras.metrics.FalseNegatives())
        parallel_model = model

    return parallel_model

```

The function `cell_network()` in the code defines the MIL model architecture consisting of the attention-based MIL pooling. All the hyper-parameters are set in this function.

The MIL attention class consists of multiple functions that perform the transformations defined in the pooling mechanism. Based on the hyper-parameter values and kernel configuration the constructor function initializes the data variables. The build function averages the weights generated by the convolutional network. The call function introduces the “tanh” product based on the attention mechanism.

```

def bag_accuracy(y_true, y_pred):
    y_true = K.mean(y_true, axis=0, keepdims=False)
    y_pred = K.mean(y_pred, axis=0, keepdims=False)
    acc = K.mean(K.equal(y_true, K.round(y_pred)))
    return acc

def bag_loss(y_true, y_pred):
    y_true = K.mean(y_true, axis=0, keepdims=False)
    y_pred = K.mean(y_pred, axis=0, keepdims=False)
    loss = K.mean(K.binary_crossentropy(y_true, y_pred), axis=-1)
    return loss

```

The code shows the custom function used to evaluate the train/validation accuracy and loss for each fold.

	A	B	C	D	E	F	G	H	I	J	K	L
1	true_negatives			true_positives	262.0132		false_negatives	774.0994		false_positives	186.662	
2	true_negatives	2707.1958		true_positives	1814.9043		false_negatives	1495.185		false_positives	1233.974	
3	true_negatives	8118.8174		true_positives	3749.113		false_negatives	1798.539		false_positives	2204.299	
4	true_negatives	13643.8838		true_positives	5842.5439		false_negatives	1949.222		false_positives	2896.236	
5	true_negatives	19440.7109		true_positives	8014.6948		false_negatives	2046.483		false_positives	3287.649	
6	true_negatives	25512.9141		true_positives	10198.4805		false_negatives	2126.21		false_positives	3586.497	
7	true_negatives	31683.6523		true_positives	12391.875		false_negatives	2206.338		false_positives	3837.316	
8	true_negatives	37892.3203		true_positives	14565.125		false_negatives	2274.712		false_positives	4024.957	
9	true_negatives	44195.9375		true_positives	16756.1055		false_negatives	2340.566		false_positives	4164.717	
10	true_negatives	50532.5234		true_positives	18947.4805		false_negatives	2405.13		false_positives	4277.784	
11	true_negatives	56896.3438		true_positives	21140.4844		false_negatives	2481.118		false_positives	4379.075	
12	true_negatives	63259.0703		true_positives	23334.9609		false_negatives	2554.434		false_positives	4468.94	
13	true_negatives	69634.4844		true_positives	25524.627		false_negatives	2614.884		false_positives	4552.261	
14	true_negatives	76033.9766		true_positives	27748.2754		false_negatives	2669.829		false_positives	4643.327	
15	true_negatives	82397.4062		true_positives	29935.1914		false_negatives	2721.786		false_positives	4739.244	
16	true_negatives	88795.7422		true_positives	32155.627		false_negatives	2772.089		false_positives	4821.866	
17	true_negatives	95175.2734		true_positives	34370.8516		false_negatives	2818.838		false_positives	4901.844	
18	true_negatives	101566.3281		true_positives	36587.707		false_negatives	2869.318		false_positives	5000.329	
19	true_negatives	107933.2031		true_positives	38792.1836		false_negatives	2918.209		false_positives	5074.118	
20	true_negatives	114339.3594		true_positives	40997.0039		false_negatives	2957.985		false_positives	5145.558	
21		120756.0938										
22	Mean	60525.76184			20156.4624			2339.749			3871.333	
23												
24												
25												
26												
27												
28												
29												

Figure 14: Excel for Metric Calculation

The figure 14 shows the collection of True Positives, True Negatives, False Positives and False Negatives from each epoch in a tabular format in excel. They were used to calculate sensitivity, specificity and F1 score.

The code for our proposed MIL architecture is referred from the GitHub ⁷.

References

Ilse, M., Tomczak, J. M. and Welling, M. (2018). Attention-based deep multiple instance learning, *Conference Proceeding* .

Wang, X., Yan, Y., Tang, P., Bai, X. and Liu, W. (2018). Revisiting multiple instance neural networks, *Pattern Recognition* **74**: 15–24.

⁷https://github.com/utayao/Atten_Deep_MIL