Configuration Manual

MSc Research Project
Masters in Data Analytics

Minakshi Tikone
Student ID: x19235283

School of Computing
National College of Ireland

Supervisor: Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Minakshi Tikone		
Student ID:	X19235283		
Programme:	Msc Data Analytics	**Year:**	2021-2022
Module:	Msc Research Project		
Lecturer:	Hicham Rifai		
Submission Due Date:	31-01-2022		
Project Title:	Ovarian cancer classification using histopathological image dataset.		
Word Count:	790	**Page Count:**	12

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: Minakshi Tikone

Date: 31-01-2022

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

Minakshi Tikone
X19235283

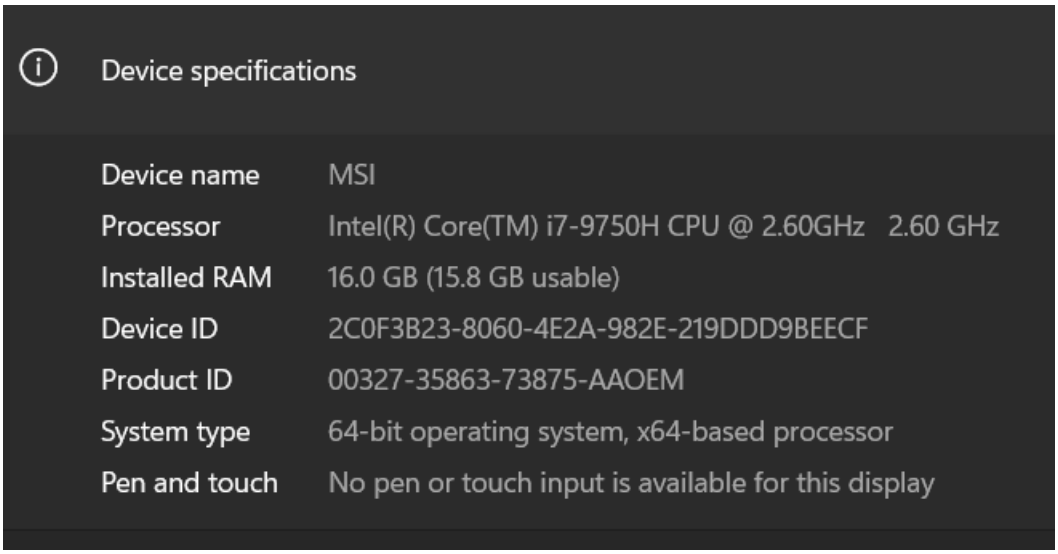
1. Introduction

The purpose of this document is to lay out the steps involved in coding the project. The hardware and software combinations that will be required to duplicate future studies are detailed. This section goes over the programming and implementation techniques that are required for efficient operations.

2. System Specification

I. Hardware Requirements

The hardware requirements which are required to execute the code is shown in fig 1



Device specifications

Device name	MSI
Processor	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.60 GHz
Installed RAM	16.0 GB (15.8 GB usable)
Device ID	2C0F3B23-8060-4E2A-982E-219DDD9BEECF
Product ID	00327-35863-73875-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.Specification of device

II. Software Requirements

Software requirements which were required for executing the code and model is explained below.

Anaconda -Jupyter Notebook

Anaconda is a freely available, open-source, and easy-to-use Python coding platform. The following figure 2 shows the anaconda prompt screen in the base root environment. Base root is the chosen environment for executing the DCNN model.

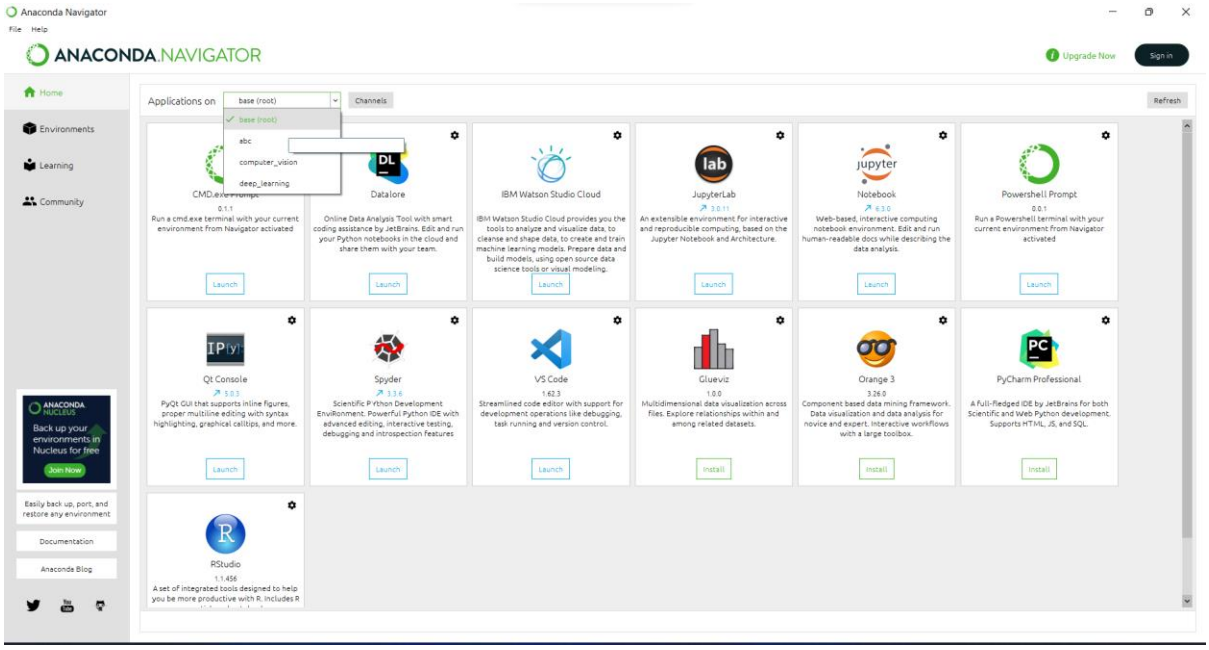


Figure 2.Anaconda Navigator

3. Data Gathering

The dataset is collected from GDC Data portal.

- Ovarian tumor wsi image dataset is gathered from the https://portal.gdc.cancer.gov/repository where the data category is Tissue slide.

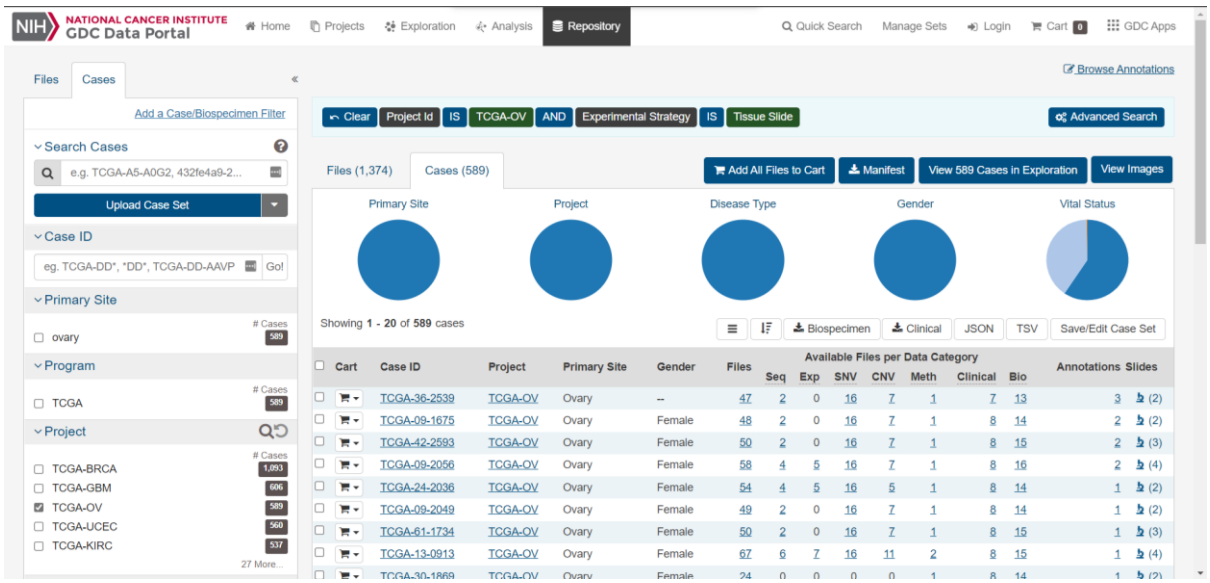


Figure 3.GDC portal for Tissue slide

Biospecimen csv dataset is gathered from https://portal.gdc.cancer.gov/repository where the data category is biospecimen.

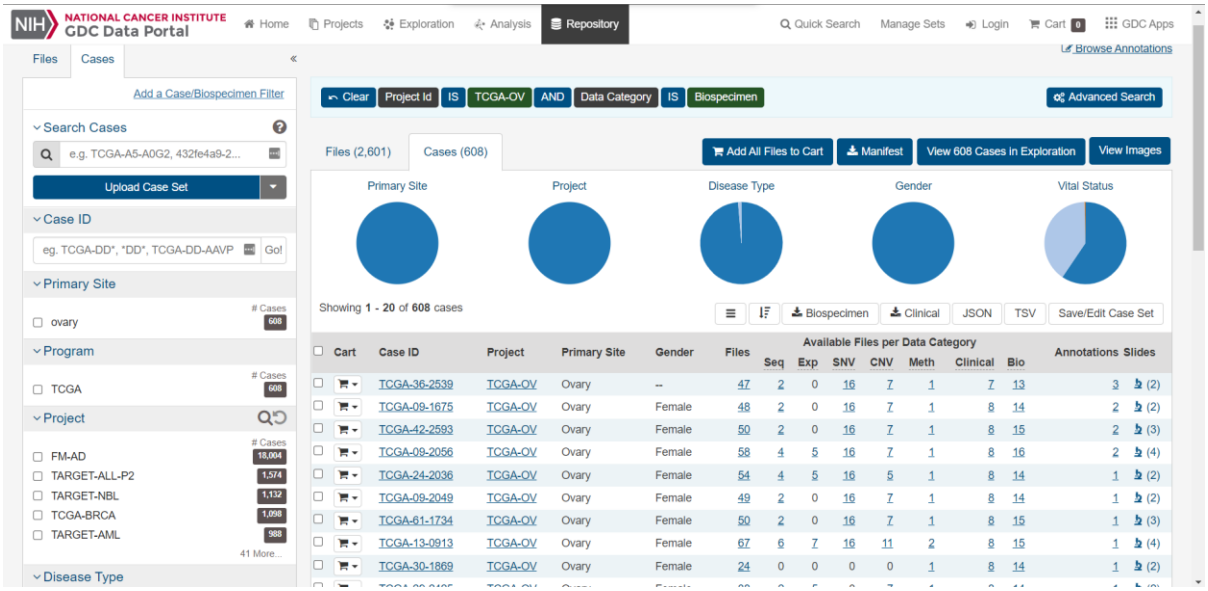


Figure 4.GDC portal for Biospecime data

Ovarian WSI slides and Biospecimen csv are publicly available on the GDC data portal.

Both the datasets are downloaded in the form of Manifest. These manifest files then loaded into the GDC data transfer tool which is a windows application shown in figure 5. GDC transfer tool helps to convert the manifest data into a readable format. The TCGA slide images are transfer into '.svs' format using the tool and the biospecimen file is transfer into csv format.

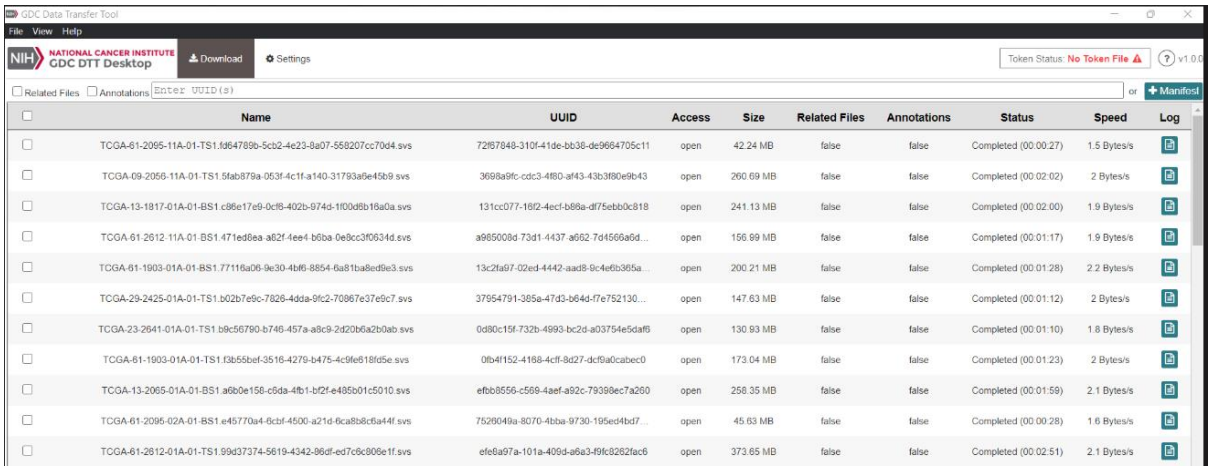


Figure 5 .GDC desktop application

4. Installing Python Libraries

- All the important python libraries are installed using Anaconda prompt or using Anaconda Navigator.

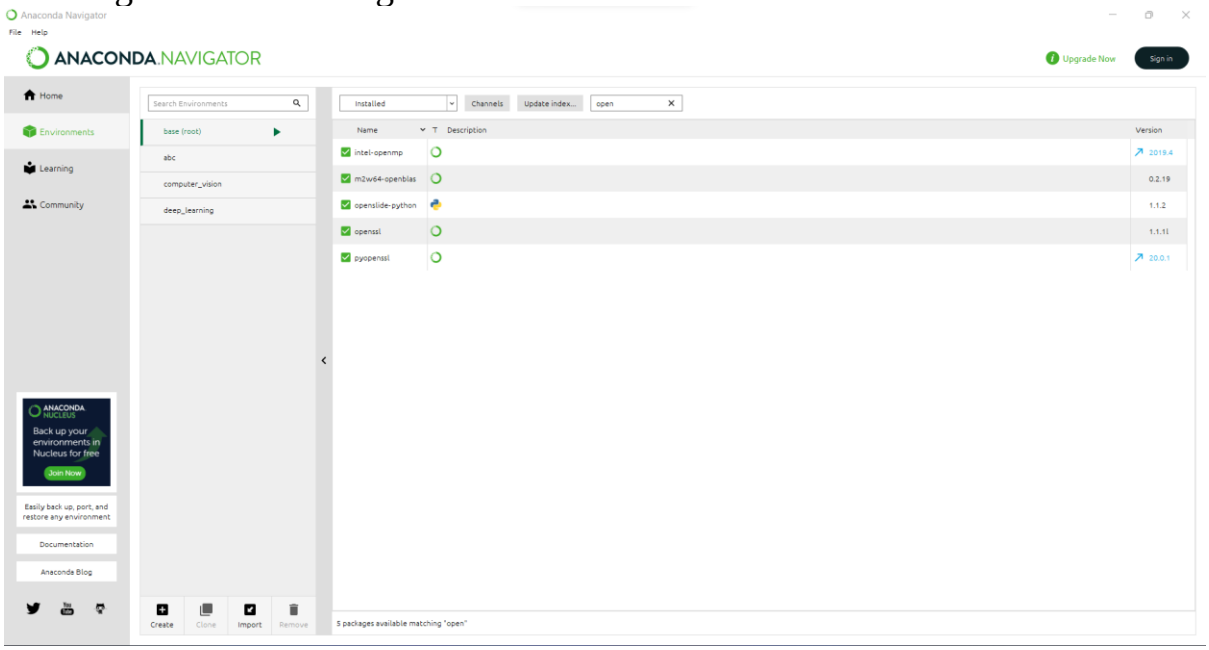


Figure 6 .Anaconda Navigator

The figure shows all of the necessary libraries that must be imported in order to accomplish the project. The following is a list of packages that must be installed before code may be executed:

Numpy Version: 1.19.5
Pandas Version: 1.1.5
Tensorflow Version: 2.5.0
Matplotlib Version: 3.2.2
Sklearn Version: 0.22.2.post1
Keras Version: 2.5.0
OpenCV Version: 4.1.2
Keras Tuner Version: 1.0.3

- Importing the Libraries
  For importing the openslide library, the following code shown in the below figure needs to be run by providing the openslide library file path, system platform.

```
In [1]: #Importing the libraries
        import os
        import platform;platform.system()
        from ctypes.util import find_library
        from ctypes import *

        #Setting up platform for Openslide
        if platform.system() == 'Windows':
            os.environ['PATH'] = r"C:\Users\minak\anaconda3\openslide\openslide-win64-20171122\bin" + ";" + os.environ['PATH']
            _lib = cdll.LoadLibrary(find_library("libopenslide-0.dll"))
```

```
In [2]: import openslide
        import matplotlib.pyplot as plt
        from pathlib import Path
        import os
        import glob
        import time
        import numpy as np
        import gc
        import progressbar
        from PIL import Image
        from skimage import filters
        import cv2
        import pandas as pd
        from sklearn.model_selection import train_test_split
        import keras
        import tensorflow as tf
        from keras.applications.vgg16 import VGG16
        from keras import models
        from keras import layers
        from keras import optimizers
        import shutil
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten
        from tensorflow.keras.optimizers import Adam
        from tensorflow.keras.metrics import categorical_crossentropy
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.models import Model
        from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
```

```
In [ ]: from tensorflow.python.client import device_lib
        print(device_lib.list_local_devices())
```

5. Data Preprocessing

Step1: Setting up the directories to save the slide images

Setting up directories

```
: path='C:\\Users\\minak\\Downloads\\research\\ovarian\\Ovarian_cancer_prediction-master\\SVS_FILES\\'
  mask_dir = 'C:\\Users\\minak\\Downloads\\research\\ovarian\\Ovarian_cancer_prediction-master\\tcga_slides\\'
  image_mask = 'C:\\Users\\minak\\Downloads\\research\\ovarian\\Ovarian_cancer_prediction-master\\train_label_masks\\'
```

Step 2:Reading the csv file using pandas and finding the missing values by using the isnull() function.

Handling the biospecimen dataset

```
bio_data = pd.read_csv('bio_specimen1.csv')
```

```
#Summary of null values
bio_data.isnull().sum()
```

```
bcr_patient_uuid                    0
bcr_sample_barcode                  0
bcr_slide_barcode                   0
bcr_slide_uuid                      0
image_file_name                     0
is_derived_from_ffpe                0
percent_lymphocyte_infiltration     0
percent_monocyte_infiltration       0
percent_necrosis                    0
percent_neutrophil_infiltration     0
percent_normal_cells                0
percent_stromal_cells               0
percent_tumor_cells                 0
percent_tumor_nuclei                0
section_location                    0
clinical_stage                      0
tumor_grade                         0
dtype: int64
```

Step 3: Dropping the columns which are not significant for the model by using the drop function

```
bio_data=bio_data.drop(['is_derived_from_ffpe','percent_lymphocyte_infiltration','percent_monocyte_infiltration','percent_neutrophil_infiltration'],axis=1)
bio_data.head()
```

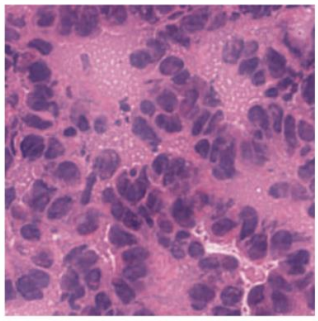
Step 4: Handling and opening WSI slides using openslide

Handling the WSI images

```
#Reading image using openslide library
slides1 = openslide.OpenSlide(os.path.join(path+ 'TCGA-13-0913-01A-01-TS1.4ad93ffb-b838-4555-a664-fb3e41923260.svs'))
```

```
patch = slides1.read_region((17800,19500), 0, (256, 256))
```

```
#displaying the extracted patch
display(patch)
slides1.close()
```



Step 5:Image conversion(SVS images to JPG and TIFF)

Image Conversion ¶

```
# Converting the SVS images to JPEG images and TIFF slides and storing in different d
def mask_images(images):
    for i, image in enumerate(images):
        slide = openslide.OpenSlide(os.path.join(path+f'/{image}.svs')) # Making Opens
        patch = slide.get_thumbnail(size=(600,400))
        imgs = cv2.cvtColor(np.array(patch)[:,:,0:3],cv2.COLOR_BGR2RGB)
        gray = cv2.cvtColor(np.array(patch)[:,:,0:3],cv2.COLOR_BGR2GRAY)
        dil = cv2.dilate(gray, kernel = np.ones((7,7),np.uint8))
        activeMap = cv2.erode(dil, kernel = np.ones((7,7),np.uint8))
        cv2.imwrite(os.path.join(mask_dir,f'{image}.jpg'),imgs)
        cv2.imwrite(os.path.join(image_mask,f'{image}.tiff'),activeMap)
        #ax[i//2, i%2].imshow(Image.fromarray(activeMap)) #Displaying Image
        slide.close()
```

Step 6: Adding metadata to tiff images using tifftools library

```
(base) PS C:\Users\minak\Downloads\research\ovarian\Ovarian_cancer_prediction-master\train_label_masks> convert TCGA-13-0913-01A-01-TS1.4ad93ffb-b838-4555-a664-fb3e41923260.tiff -define tiff:tile-geometry=256x25
6 'ptif:TCGA-13-0913-01A-01-TS1.4ad93ffb-b838-4555-a664-fb3e41923260.tiff'
>> convert TCGA-13-0913-02A-01-BS1.1ac7ab05-f6af-42db-acd0-0d303a1dd585.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-13-0913-02A-01-BS1.1ac7ab05-f6af-42db-acd0-0d303a1dd585.tiff'
>> convert TCGA-13-0913-02A-01-TS1.6b3e1267-f670-4836-9e04-012d1b63f14a.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-13-0913-02A-01-TS1.6b3e1267-f670-4836-9e04-012d1b63f14a.tiff'
>> convert TCGA-13-1817-01A-01-BS1.c86e17e9-0cf6-402b-974d-1f00d6b16a0a.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-13-1817-01A-01-BS1.c86e17e9-0cf6-402b-974d-1f00d6b16a0a.tiff'
>> convert TCGA-13-1817-02A-01-BS1.2f9967b6-cb04-4d01-ad90-52c30ee97836.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-13-1817-02A-01-BS1.2f9967b6-cb04-4d01-ad90-52c30ee97836.tiff'
>> convert TCGA-13-2059-01A-01-BS1.f4a0b6c4-94ad-4412-ba8d-369e518af8be.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-13-2059-01A-01-BS1.f4a0b6c4-94ad-4412-ba8d-369e518af8be.tiff'
>> convert TCGA-13-2065-01A-01-BS1.a6b0e158-c6da-4fb1-bf2f-e485b01c5010.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-13-2065-01A-01-BS1.a6b0e158-c6da-4fb1-bf2f-e485b01c5010.tiff'
>> convert TCGA-23-1023-01A-01-BS1.f2260f08-0126-4fe9-bbb0-09e4fc2fb30e.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-23-1023-01A-01-BS1.f2260f08-0126-4fe9-bbb0-09e4fc2fb30e.tiff'
>> convert TCGA-23-1023-01A-02-BS2.1a4fa055-18ac-4c34-aa07-3e6fbba79d49.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-23-1023-01A-02-BS2.1a4fa055-18ac-4c34-aa07-3e6fbba79d49.tiff'
>> convert TCGA-23-2641-01A-01-BS1.dd2cb54e-ffc1-4ba7-a776-9aa12827fae2.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-23-2641-01A-01-BS1.dd2cb54e-ffc1-4ba7-a776-9aa12827fae2.tiff'
>> convert TCGA-23-2641-01A-01-TS1.b9c56790-b746-457a-a8c9-2d20b6a2b0ab.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-23-2641-01A-01-TS1.b9c56790-b746-457a-a8c9-2d20b6a2b0ab.tiff'
>> convert TCGA-24-2036-01A-01-TS1.a09a994b-2da2-4b47-a479-2ac663d0fa9f.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-24-2036-01A-01-TS1.a09a994b-2da2-4b47-a479-2ac663d0fa9f.tiff'
>> convert TCGA-24-2280-01A-01-TS1.2dd90f1f-fe80-438f-bb4e-5d80716e6962.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-24-2280-01A-01-TS1.2dd90f1f-fe80-438f-bb4e-5d80716e6962.tiff'
>> convert TCGA-24-2298-11A-01-TS1.d4b2f444-eecd-41d6-97c6-0a356b6a4cb3.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-24-2298-11A-01-TS1.d4b2f444-eecd-41d6-97c6-0a356b6a4cb3.tiff'
>> convert TCGA-25-2390-01A-01-BS1.755bb36f-5911-414c-bef6-6a1c330300be.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-25-2390-01A-01-BS1.755bb36f-5911-414c-bef6-6a1c330300be.tiff'
>> convert TCGA-29-1696-01A-01-TS1.9f136ba5-516c-439d-8e2d-a9061cce1191.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-29-1696-01A-01-TS1.9f136ba5-516c-439d-8e2d-a9061cce1191.tiff'
>> convert TCGA-29-1705-01A-01-BS1.febc5de5-d46e-4d34-a597-501782af363a.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-29-1705-01A-01-BS1.febc5de5-d46e-4d34-a597-501782af363a.tiff'
>> convert TCGA-29-1705-02A-01-TS1.4d5e4d51-e4ba-4d62-925e-3a3c302c9195.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-29-1705-02A-01-TS1.4d5e4d51-e4ba-4d62-925e-3a3c302c9195.tiff'
>> convert TCGA-29-2414-02A-01-BS1.d98dd76c-5162-4b4d-b0fa-0604d8a40117.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-29-2414-02A-01-BS1.d98dd76c-5162-4b4d-b0fa-0604d8a40117.tiff'
>> convert TCGA-29-2414-02A-01-TS1.9e8b6cda-a655-40f9-a448-d45218dc54c8.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-29-2414-02A-01-TS1.9e8b6cda-a655-40f9-a448-d45218dc54c8.tiff'
>> convert TCGA-29-2425-01A-01-BS1.219c877f-a1fc-4322-bc83-e1ab4e10f906.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-29-2425-01A-01-BS1.219c877f-a1fc-4322-bc83-e1ab4e10f906.tiff'
>> convert TCGA-29-2425-01A-01-TS1.b02b7e9c-7826-4dda-9fc2-70867e37e9c7.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-29-2425-01A-01-TS1.b02b7e9c-7826-4dda-9fc2-70867e37e9c7.tiff'
>> convert TCGA-30-1853-01A-01-BS1.97d7d1f1-3b08-47ed-84d2-9c7439d1720c.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-30-1853-01A-01-BS1.97d7d1f1-3b08-47ed-84d2-9c7439d1720c.tiff'
>> convert TCGA-30-1853-01A-02-BS2.59402746-32fc-4ce7-ac6f-235da20da619.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-30-1853-01A-02-BS2.59402746-32fc-4ce7-ac6f-235da20da619.tiff'
>> convert TCGA-30-1869-01A-01-BS1.973cce11-963d-418d-8ec0-d4fdfbc4f524.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-30-1869-01A-01-BS1.973cce11-963d-418d-8ec0-d4fdfbc4f524.tiff'
>> convert TCGA-36-2538-01A-01-BS1.7f6cfc89-51ce-4364-8ed0-8d6d310dba8c.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-36-2538-01A-01-BS1.7f6cfc89-51ce-4364-8ed0-8d6d310dba8c.tiff'
>> convert TCGA-36-2539-01A-01-BS1.5af1cf67-427a-46ec-86fc-3063eaf41e5f.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-36-2539-01A-01-BS1.5af1cf67-427a-46ec-86fc-3063eaf41e5f.tiff'
>> convert TCGA-36-2539-01A-01-TS1.2e02674c-32b0-45c5-b3e9-c3c28c5bb1b9.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-36-2539-01A-01-TS1.2e02674c-32b0-45c5-b3e9-c3c28c5bb1b9.tiff'
>> convert TCGA-57-1992-11A-01-BS1.5bde7102-0e67-4a0f-8804-21be2a137f1e.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-57-1992-11A-01-BS1.5bde7102-0e67-4a0f-8804-21be2a137f1e.tiff'
>> convert TCGA-57-1992-11A-01-TS1.4126a0b7-7adb-455f-a8e2-5a945440e80a.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-57-1992-11A-01-TS1.4126a0b7-7adb-455f-a8e2-5a945440e80a.tiff'
>> convert TCGA-61-1734-11A-01-TS1.c55c7e4a-e73f-4dc1-88e4-c4318da07063.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-61-1734-11A-01-TS1.c55c7e4a-e73f-4dc1-88e4-c4318da07063.tiff'
>> convert TCGA-61-1903-01A-01-BS1.77116a06-9e30-4bf6-8854-6a81ba8ed9e3.tiff -define tiff:tile-geometry=256x256 'ptif:TCGA-61-1903-01A-01-BS1.77116a06-9e30-4bf6-8854-6a81ba8ed9e3.tiff'
```

6. Models

Step 1: Splitting the dataset into train set and validation set

Implemnting Deep Learning

```
#Splitting the dataset in train and validation sets
c_train, c_val = train_test_split(bio_data, test_size=0.20, random_state=101)

print(c_train.shape)
print(c_val.shape)
```

```
(36, 14)
(9, 14)
```

Step2: Setting index in biospecimen dataframe and setting values for hyperparameters. Using default image size

```
In [39]: # Setting image_name as the index in bio specimen dataframe
         bio_data=bio_data.set_index('image_name')
         train_list = list(c_train['image_name'])
         val_list = list(c_val['image_name'])
```

```
In [41]: #Setting up Hyperparameters
         num_train_samples = len(c_train)
         num_val_samples = len(c_val)
         train_batch_size = 32
         val_batch_size = 32

         train_steps = np.ceil(num_train_samples / train_batch_size)
         val_steps = np.ceil(num_val_samples / val_batch_size)
```

```
In [43]: IMAGE_SIZE = 256
```

Step 3: Creating training and validation directories for storing masked images

```
[45]: #Creating training and validation directories
      train_dir = os.path.join(image_mask, 'train_dir')
      os.mkdir(train_dir)
      val_dir = os.path.join(image_mask, 'val_dir')
      os.mkdir(val_dir)

      # create new folders inside train_dir
      Grade_3 = os.path.join(train_dir, 'Grade_3')
      os.mkdir(Grade_3)
      Grade_2 = os.path.join(train_dir, 'Grade_2')
      os.mkdir(Grade_2)


      # create new folders inside val_dir
      Grade_3 = os.path.join(val_dir, 'Grade_3')
      os.mkdir(Grade_3)
      Grade_2 = os.path.join(val_dir, 'Grade_2')
      os.mkdir(Grade_2)
```

```
for image in train_list:
    try:
        fname = image
        target = bio_data.loc[image, 'tumor_grade']

        if target == 'G3':
            label = 'Grade_3'
        elif target == 'G2':
            label = 'Grade_2'

         # source path to image
        src = os.path.join(image_mask,f'{fname}.tiff')
            # destination path to image
        dst = os.path.join(train_dir,label,f'{fname}.tiff')
            # move the image from the source to the destination
        shutil.move(src, dst)
        #print(src,dst)
    except:
        continue

for image in val_list:
    try:
        fname = image
        target = bio_data.loc[image, 'tumor_grade']

        if target == 'G3':
            label = 'Grade_3'
        elif target == 'G2':
            label = 'Grade_2'



        src = os.path.join(image_mask,f'{fname}.tiff')
            # destination path to image
        dst = os.path.join(val_dir,label,f'{fname}.tiff')
            # move the image from the source to the destination
        shutil.move(src, dst)
    except:
        continue
```

Step4: Data Augmentation

Data Augmentation

```
#Data Augmentation using ImageDataGenerator
datagen = ImageDataGenerator(rescale = 1.0 / 255,
                             rotation_range = 90,
                             zoom_range = 0.2,
                             horizontal_flip=True,
                             vertical_flip=True)

train_gen = datagen.flow_from_directory(train_dir,
                                        target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                        batch_size=train_batch_size,
                                        class_mode='categorical')

val_gen = datagen.flow_from_directory(val_dir,
                                      target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                      batch_size=val_batch_size,
                                      class_mode='categorical')

# Note: shuffle=False causes the test dataset to not be shuffled
test_gen = datagen.flow_from_directory(val_dir,
                                       target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                       batch_size=1,
                                       class_mode='categorical',
                                       shuffle=False)
```

```
Found 36 images belonging to 2 classes.
Found 9 images belonging to 2 classes.
Found 9 images belonging to 2 classes.
```

Step5:Executing the VGG16 and VGG19 models

```
# Create the model VGG16
num_classes = 2
model = models.Sequential()

# Add the vgg convolutional base model
model.add(vgg_conv)


# Add Dense new layers
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(num_classes, activation='sigmoid'))

# Show a summary of the model. Check the number of trainable parameters
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 8, 8, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              33555456
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 2050
=================================================================
Total params: 48,272,194
Trainable params: 48,272,194
Non-trainable params: 0
```

```
#VGG19 implementation
model1 = models.Sequential()

# Add the vgg convolutional base model
model1.add(VGG_19_pre_trained)


# Add Dense new layers
model1.add(layers.Flatten())
model1.add(layers.Dense(1024, activation='relu'))
model1.add(layers.Dropout(0.2))
model1.add(layers.Dense(num_classes, activation='sigmoid'))

# Show a summary of the model. Check the number of trainable parameters
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 512)               20024384
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              525312
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 2050
=================================================================
Total params: 20,551,746
Trainable params: 20,551,746
Non-trainable params: 0
```

Step 6: Evaluating the VGG16 and VGG19 models

```
#Fitting VGG16 model
history = model.fit(train_gen, steps_per_epoch=train_steps,
                    validation_data=val_gen,
                    validation_steps=val_steps,
                    epochs=10, verbose=1)
```

```
Epoch 1/10
2/2 [==============================] - 4s 2s/step - loss: 0.7843 - accuracy: 0.6111 - val_loss: 1.3696 - val_accuracy: 0.44
44
Epoch 2/10
2/2 [==============================] - 19s 9s/step - loss: 0.6753 - accuracy: 0.7222 - val_loss: 0.7458 - val_accuracy: 0.4
444
Epoch 3/10
2/2 [==============================] - 20s 10s/step - loss: 0.6376 - accuracy: 0.7222 - val_loss: 0.7860 - val_accuracy: 0.
4444
Epoch 4/10
2/2 [==============================] - 4s 2s/step - loss: 0.5985 - accuracy: 0.7222 - val_loss: 0.9436 - val_accuracy: 0.44
44
Epoch 5/10
2/2 [==============================] - 4s 2s/step - loss: 0.5834 - accuracy: 0.7222 - val_loss: 0.9497 - val_accuracy: 0.44
44
Epoch 6/10
2/2 [==============================] - 21s 10s/step - loss: 0.5945 - accuracy: 0.7222 - val_loss: 0.8650 - val_accuracy: 0.
4444
Epoch 7/10
2/2 [==============================] - 4s 2s/step - loss: 0.6150 - accuracy: 0.7222 - val_loss: 0.7676 - val_accuracy: 0.44
44
Epoch 8/10
2/2 [==============================] - 4s 2s/step - loss: 0.6282 - accuracy: 0.7222 - val_loss: 0.7241 - val_accuracy: 0.44
44
Epoch 9/10
2/2 [==============================] - 4s 2s/step - loss: 0.6275 - accuracy: 0.7222 - val_loss: 0.7589 - val_accuracy: 0.44
44
Epoch 10/10
2/2 [==============================] - 4s 2s/step - loss: 0.6204 - accuracy: 0.7222 - val_loss: 0.8250 - val_accuracy: 0.44
44
```

```
#Fitting VGG19 model
history1 = model1.fit(train_gen, steps_per_epoch=train_steps,
                    validation_data=val_gen,
                    validation_steps=val_steps,
                    epochs=10, verbose=1)
```

```
C:\Users\minak\anaconda3\lib\site-packages\tensorflow\python\data\ops\dataset_ops.py:3349: UserWarning: Even though the tf.
config.experimental_run_functions_eagerly option is set, this option does not apply to tf.data functions. tf.data functions
are still traced and executed as graphs.
  warnings.warn(
```

```
Epoch 1/10
2/2 [==============================] - 24s 12s/step - loss: 0.7243 - accuracy: 0.3889 - val_loss: 1.0290 - val_accuracy: 0.
4444
Epoch 2/10
2/2 [==============================] - 4s 2s/step - loss: 0.6706 - accuracy: 0.7222 - val_loss: 1.0191 - val_accuracy: 0.44
44
Epoch 3/10
2/2 [==============================] - 4s 2s/step - loss: 0.6082 - accuracy: 0.7222 - val_loss: 1.0868 - val_accuracy: 0.44
44
Epoch 4/10
2/2 [==============================] - 22s 11s/step - loss: 0.6577 - accuracy: 0.7222 - val_loss: 0.9232 - val_accuracy: 0.
4444
Epoch 5/10
2/2 [==============================] - 4s 2s/step - loss: 0.5636 - accuracy: 0.7222 - val_loss: 0.7127 - val_accuracy: 0.44
44
Epoch 6/10
2/2 [==============================] - 22s 11s/step - loss: 0.6355 - accuracy: 0.6944 - val_loss: 0.7832 - val_accuracy: 0.
4444
Epoch 7/10
2/2 [==============================] - 4s 2s/step - loss: 0.5917 - accuracy: 0.7222 - val_loss: 0.9417 - val_accuracy: 0.44
44
Epoch 8/10
2/2 [==============================] - 23s 11s/step - loss: 0.6420 - accuracy: 0.7222 - val_loss: 0.9252 - val_accuracy: 0.
4444
Epoch 9/10
2/2 [==============================] - 4s 2s/step - loss: 0.6025 - accuracy: 0.7222 - val_loss: 0.7580 - val_accuracy: 0.44
44
Epoch 10/10
2/2 [==============================] - 22s 11s/step - loss: 0.5348 - accuracy: 0.7222 - val_loss: 0.9296 - val_accuracy: 0.
4444
```

Step 7: Visualising the result

Visualising the training and validation accuracy and loss of the VGG16 model

```
# Visualising for VGG16
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss for VGG16')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy for VGG16')
plt.legend()
plt.figure()
```

Visualising the training and validation accuracy and loss of the VGG19 model

```
# Visualising for VGG19

acc = history1.history['accuracy']
val_acc = history1.history['val_accuracy']
loss = history1.history['loss']
val_loss = history1.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss for VGG19')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy for VGG19')
plt.legend()
plt.figure()
```