National College of Ireland

Lung Cancer Classification from Histologic Images using Capsule Networks

MSc Research Project
Data Analytics

Bedanga Bikash Roy Medhi
Student ID: x18182127

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Bedanga Bikash Roy Medhi
Student ID:	x18182127
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	28/09/2020
Project Title:	Lung Cancer Classification from Histologic Images using Capsule Networks
Word Count:	1651
Page Count:	24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	27th September 2020

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):	

Lung Cancer Classification from Histologic Images using Capsule Networks

Bedanga Bikash Roy Medhi
x18182127

1 Software and Hardware requirements

The project implementations required the use of some dedicated hardware which are listed below:

Processor and Generation: Intel Core i7-3612QM

Operating System : Ubuntu 18.04.4 LTS

Ram : 16 GB DDR3

Hard Disk Drive : 1 TB

The software required for the implementation are:

Anaconda version : 4.8

Python version 3.7

Google colab

LibreOffice Calc

Google Chrome 84 and above

Visual Studio Code

2 Environment setup

For the implementations has been done in two phases.

Phase 1 which it the pre-processing is done on the local system.

Phase 2 where the model training and testing are done in the Google colaboratory environment.

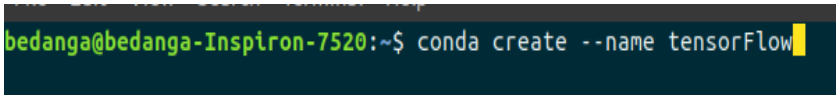

Figure 1: Creating environment

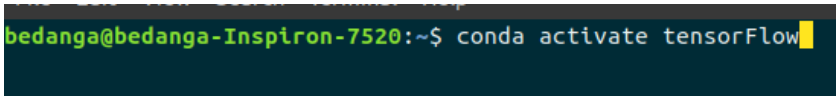

Figure 2: Activate environment

2.1 Phase 1 : Local environment setup

Download Anaconda 4.8[1].

After installing Anaconda a new environment was set up for image processing.

The new environment has to be activated in the terminal in order to be used.

To launch the jupyter notebook the "jupyter notebook" has to be executed and it will launch.

2.2 Phase 2: Colab environment setup

For the colab environment setup we need to first setup login credentials for colab by signing up in Google.

Then we need to login in colab home[2]

After successful successful login the colaboratory provides notebooks to execute python code.

To utilize the GPU dedicated for the session can be accessed by changing the runtime. In order to do that we need to visit the runtime tab in the top left side of the screen and click on it as shown in Figure 3.

[1]https://www.anaconda.com/products/individual
[2]https://colab.research.google.com/

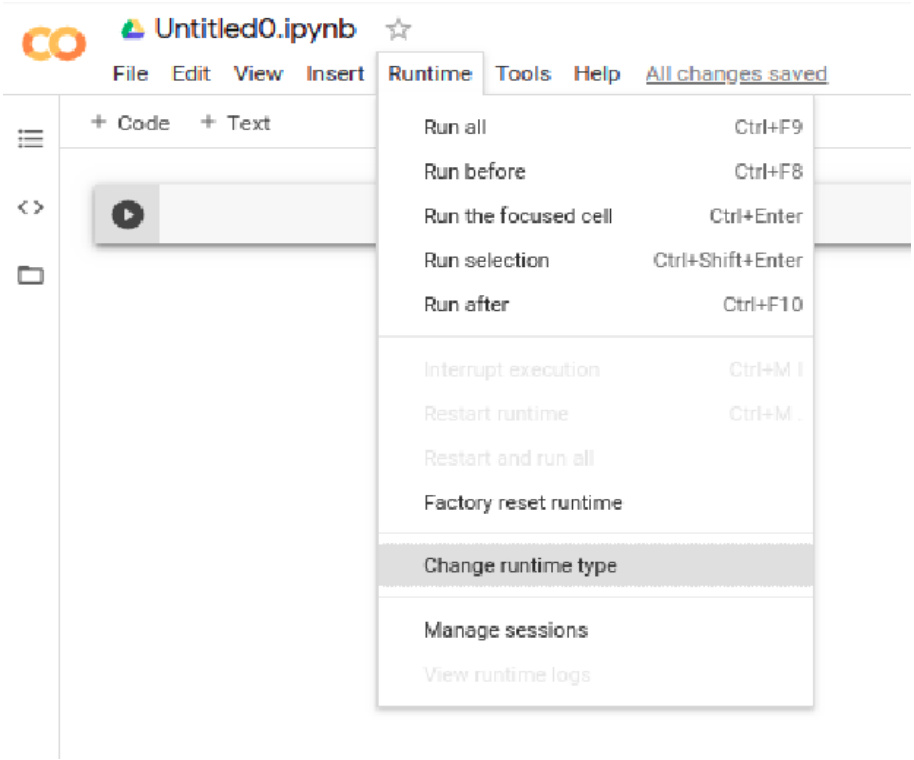

Figure 3: Change runtime

After that click on the change runtime option and a popup will appear where we can select the GPU option and save it. The runtime restarts with the GPU allocated it.

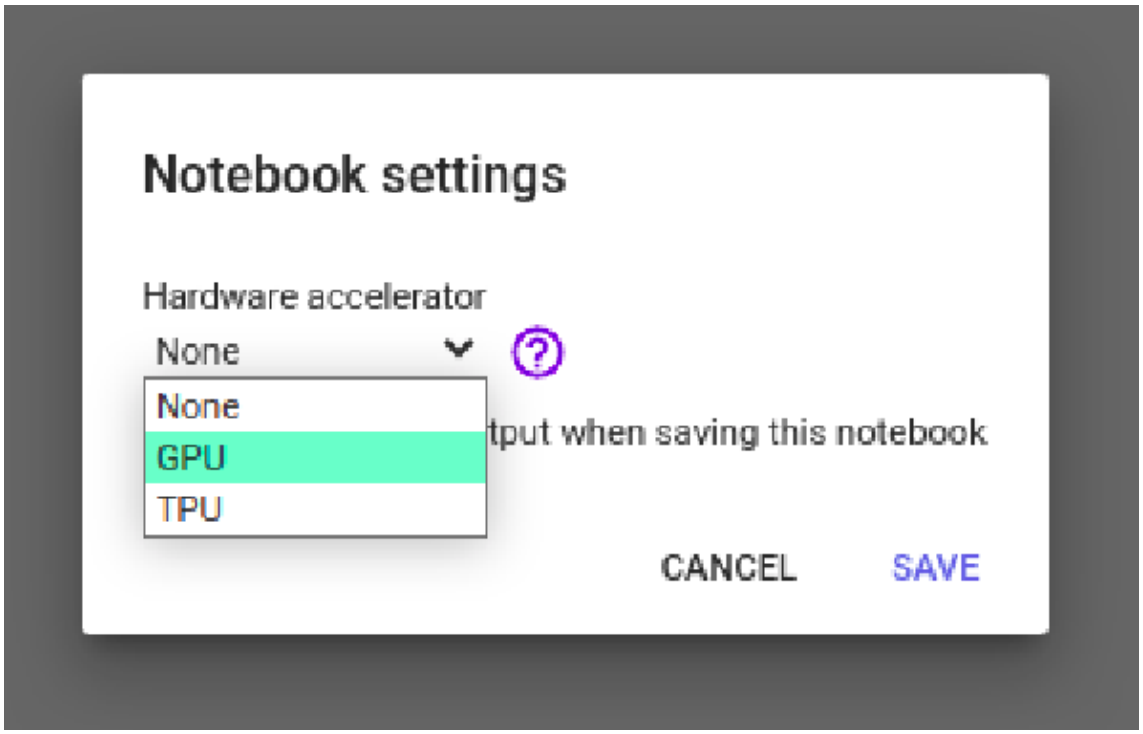

Figure 4: Setting GPU

It provides around 13 GB of RAM and 68 GB of disk space for a session.

3 Data Source

The data source is from the archives of Cornell University[3]. The dataset can be downloaded from the link[4] provided in the documentation of the dataset(Borkowski et al.; 2019).
Folder structure of the dataset: Once the download is completed the images are arranged in the following way:

Main directory inside the compressed file is lung_colon_image_set. There are two sub directories here; lung_image_sets and colon_image_sets.

colon_image_sets: Colon cancer image set having images of colon cancers. Inside this folder there are two more sub directories. These sub directories colon_n and colon_aca which are the two colon cancer types mentioned in the documentation i.e. benign colonic tissues and colon adenocarcinomas.

lung_image_sets : Lung cancer image set having images of lung cancers. Inside this folder there are three more sub directories. These sub directories lung_n, lung_scc and lung_aca which are the three lung cancer types mentioned in the documentation i.e. benign lung tissues, lung squamous cell carcinomas and lung adenocarcinomas.

Each sub type has 5,000 images that sums up to 25,000 images for all the five types of cancer.

Figure 5: Dataset Source

4 Code Structure

The below section explains the code structure of the project.

BrisqueScore directory: contains the .ipynb files for calculation of BRISQUE score of the images.

capsulelayers.py : The file contains the layer definitions of the capsule network.

[3]https://arxiv.org/abs/1912.12142
[4]https://github.com/tampapath/lung_colon_image_set/blob/master/README.md

LoadCapsuleNetworkModel.py : This file contains the function where the Capsule Network is define. It is used for instantiating the Capsule Network.

LoadCNNModel.py : The module contains the definition of the 3 pre-trained CNN architectures. The respective CNN models can be instantiated using this module.

Pre_Processing_Normalization.ipynb : The .ipynb notebook is used for the stain normalization of the images.

pre_processing_splitting_df.ipynb : This notebook contains the code for data transformation and splitting.

CapsuleNetwork.ipynb : The notebook is used for the training and evaluation of the Capsule Network.

VGG19.ipynb : The notebook is used for the training and evaluation of the VGG19 model.

RestNet50.ipynb : The notebook has the code for training and evaluation of the ResNet50 model.

DenseNet.ipynb : Notebook used for training and evaluation of the DenseNet-121 model.

5 Python Libraries and versions

Table 1: Libraries and versions

Library	**Version**	**Reference**	**Installation Command**
numpy	1.18	https://numpy.org/	pip install numpy
pandas	1.1.0	https://pandas.pydata.org/	pip install pandas
Matplotlib	3.1.1	https://matplotlib.org/	python -m pip install -U matplotlib
staintools	2.1.2	https://pypi.org/project/staintools/	pip install staintools
tqdm	4.46.0	https://tqdm.github.io/releases/	pip install tqdm
skimage	0.17.2	https://scikit-image.org/	python -m pip install -U scikit-image
image-quality	1.2.5	https://pypi.org/project/image-quality/	pip install image-quality
cv2 (opencv-python)	4.2.0	https://opencv-python-tutroals.readthedocs.io/en/latest/index.html	pip install opencv-python
sklearn	0.22.1	https://scikit-learn.org/stable/	pip install -U scikit-learn
keras	2.2.4	https://keras.io/	pip install keras==2.2.4
tensorflow-gpu	1.15	https://www.tensorflow.org/	pip install tensorflow-gpu==1.15
seaborn	0.10.0	https://seaborn.pydata.org/	pip install seaborn

6 Data loading, splitting and pre-processing

The following section will explain the pre-processing, splitting and loading.

6.1 Data pre-processing

In the pre-processing the images include stain normalization and splitting of data.

Libraries of BRISQUE score calculation:

```
#Importing Libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import zipfile
import staintools
from tqdm import tqdm, tqdm_notebook
from skimage import img_as_ubyte, img_as_float
from skimage import io
#Image Quality:
import imquality.brisque as brisque
```

Function that calculates the BRISQUE score.

```
#Function for calculating the BRISQUE score of the
images
path='./lung_colon_image_setlung_image_sets/'
def calc_brisque_score_n(dirs):
  for i in tqdm(range(len(os.listdir(path+dirs)))):
    imgPath = path+dirs+'/'+os.listdir(path+dirs)[i]
    img_f = img_as_float(io.imread(imgPath))
    row = ["Image Class":dirs, "Image
File ":os.listdir(path+dirs)[i],"Brisque
Score":brisque.score(img_f)]
    brisque_score_raw.loc[len(brisque_score_raw.index)]
= list(row[0].values())
```

For the stain normalization of the images the following libraries are used; shown.

```
#Importing Libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import staintools
import random
from tqdm import tqdm, tqdm_notebook
#Thread import concurrent.futures
#Time
import time
#Warnings import warnings
%load_ext autoreload
%autoreload 2
%matplotlib inline
random.seed(123)
```

Code for setting the target image for stain normalization.

```
for dirs in dir_lst:
 if dirs == 'lung_aca':
  target_aca = cv2.imread(path+dirs+'/'+
  'lungaca3024.jpeg')
  target_aca = cv2.cvtColor(target_aca,cv2.COLOR_BGR2RGB)
  normalizer_aca = staintools.StainNormalizer(method=
  'vahadane')
  normalizer_aca.fit(target_aca)
 elif dirs == 'lung_scc':
  target_scc = cv2.imread(path+dirs+'/'+'
  lungscc3597.jpeg')
  target_scc = cv2.cvtColor(target_scc,cv2.COLOR_BGR2RGB)
  normalizer_scc = staintools.StainNormalizer(method=
  'vahadane')
 else:
  target_n = cv2.imread(path+dirs+'/'+'
  lungn3468.jpeg')
  target_n = cv2.cvtColor(target_n,cv2.COLOR_BGR2RGB)
  normalizer_n = staintools.StainNormalizer(method=
  'vahadane')
  normalizer_n.fit(target_n)
```

Applying normalization the other images and saving them in new directory.

```
'''
Creates new directory structure for the normalized
image
normalize----
            |
            ----lung_aca
            |
            ----lung_n
            |
            ----lung_scc
'''
os.mkdir('normalize/')
for dirs in dir_lst:
 os.mkdir('normalize/'+dirs)
'''
The below list comprehensions are used contain the
path of the images.
'''
lung_n_lst = [path+'lung_n'+'/'+image for image in
os.listdir(path+'lung_n')]
lung_aca_lst = [path+'lung_aca'+'/'+image for image in
os.listdir(path+'lung_aca')]
lung_scc_lst = [path+'lung_scc'+'/'+image for image in
os.listdir(path+'lung_scc')]

def stain_Normalize(imgPath):
 img = cv2.cvtColor(cv2.imread(imgPath),
 cv2.COLOR_BGR2RGB)
 a = imgPath.split('/')
 if a[len(a)-2] == 'lung_aca':
  n_img = normalizer_aca.transform(img)
 elif a[len(a)-2] == 'lung_scc':
  n_img = normalizer_scc.transform(img)
 else:
  n_img = normalizer_n.transform(img)
 resize = cv2.resize(n_img,(500,500),interpolation =
 cv2.INTER_CUBIC)
 a = imgPath.split('/')
 plt.imsave('./normalize/'+a[len(a)-2]+'/'+a[len(a)-1],
 resize)
```

Calling the executor that normalizes the images.

```
'''
Thread pool executor for normalizing the images of
lung_aca lung adenocarcinoma
'''
with concurrent.futures.ThreadPoolExecutor() as
executor:
 executor.map(stain_Normalize, lung_aca_lst)
'''
Thread pool executor for normalizing the images of
lung_n benign lung tissue
'''
with concurrent.futures.ThreadPoolExecutor() as
executor:
 executor.map(stain_Normalize, lung_n_lst)
'''
Thread pool executor for normalizing the images of
lung_scc lung squamous cell carcinoma
'''
with concurrent.futures.ThreadPoolExecutor() as
executor:
 executor.map(stain_Normalize, lung_scc_lst)
```

6.2 Data splitting

In the next phase, the images are splitted in to test, train and validation and the labels are stored in csv.

The libraries used in the splitting and label encoding process are shown below.

```
#Libraries for splitting and label encoding
import numpy as np
import os
import pandas as pd
import cv2
import random
import shutil
from sklearn.model_selection import train_test_split
import zipfile
from tqdm import tqdm, tqdm_notebook
from sklearn.preprocessing import LabelBinarizer
```

Extracting the image names and class labels and storing in dataframe.

```
 #Initializing the dataframe for storing image name
and labels
#'Image':  Image file names
#'CancerType':  Cancer type
dataset = pd.DataFrame(columns=['Image','CancerType'])
#Source director for the images
path ='./normalize/'
#Inserting the image names and labels in the "dataset"
dataframe
for dirs in dir_class:
  path_ac = path+dirs+'//'
  for img in tqdm(os.listdir(path_ac)):
    dataset = dataset.append('Image':
img,'CancerType':dirs, ignore_index=True)
#Shuffling the dataframe
Data = dataset.sample(frac=1).reset_index(drop=True)
Data.head()
Output:
        Image                CancerType
0   lungscc2534.jpeg          lung_scc
1   lungaca4544.jpeg          lung_aca
2   lungaca947.jpeg           lung_aca
3   lungn21.jpeg              lung_n
4   lungn3111.jpeg            lung_n
```

Encoding the class labels using LabelBinarizer and stored them in a new dataframe df_encoded.

```
 #Initializing the label encode using LabelBinarizer()
lb = LabelBinarizer()
#Encoding the labels specified in the 'CancerType'
column
#encoded_ :  contains the label encodings
encoded_ = lb.fit_transform(Data['CancerType'])
#Labels in the binarizer object
lb.classes_
Output:
array(['lung_aca', 'lung_n', 'lung_scc'], dtype='<U8')
#Appending the encoded values to a dataframe
df_encoded = pd.DataFrame(encoded_,
columns=lb.classes_)
#dataframe containing the encodings
df_encoded.head()
Output:
    lung_aca     lung_n    lung_scc
0       0          0          1
1       1          0          0
2       1          0          0
3       0          1          0
4       0          1          0
```

Appended the encoding and the image names into one single dataframe.

```
 #Concatenating the Data and df_encoded dataframes to
a new dataframe
DataFinal = pd.concat([Data,df_encoded], axis=1)
```

Splitting the dataframe into train, test and validation sets. The ratio of split is 80:20 for train and test. For the training set 20% is split off for the validation set.

```
 #Creating Train - Test split
train, test = train_test_split(DataFinal, test_size =
0.20,random_state=42,shuffle=True)
train=train.reset_index(drop=True)
test = test.reset_index(drop=True)
```

```
 #Creating Train - Validation split
train, val = train_test_split(train, test_size =
0.20,random_state=42,shuffle=True)
train = train.reset_index(drop=True)
val = val.reset_index(drop=True)
```

The images from the actual source are transferred to newly created directories for Test, Train and Validation.

```
#Creating directories for Train, Test and Validation
for storing the images.
os.mkdir('Train')
os.mkdir('Test')
os.mkdir('Val')
'''
The function fetches the image names from the
dataframe and store it in a directory
'''
def transfer(dir,df):
  path = './normalize/'
  for index, row in tqdm(df.iterrows()):
    shutil.copy(path+row['CancerType']+'/'+
    row['Image'],dir) '''
The train, test and validation images names are
fetched from the dataframe and copied to the
respective directory
'''
transfer('Train',train)
transfer('Test',test)
transfer('Val',val)
```

Exporting the dataframes of train, test and validation to .csv files.

```
#Exporing the train, test and val dataframes to csv
files.
train.to_csv('train.csv', index=False)
test.to_csv('test.csv', index=False)
val.to_csv('val.csv', index=False)
```

In the last step the .csv files and the directories containing the images are zipped together. The zipped file is uploaded in google drive and will be used for training and testing.

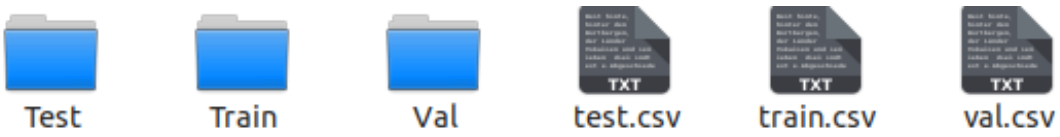

Figure 6: Folder structure

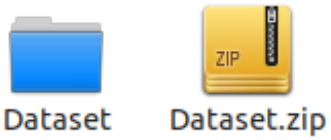

Figure 7: Files zipped together

6.3 Data Loading

The dataset is loaded in to Google drive and then using the code below the drive can be mounted to the colab environment. We need to click on the blue link shown in the figure to generate the authentication key required for mounting.

```
#Mounting Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

Key for connecting the drive with google colab is generated by the link shown below.

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

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?cl
```

Figure 8: Link to generate authentication key

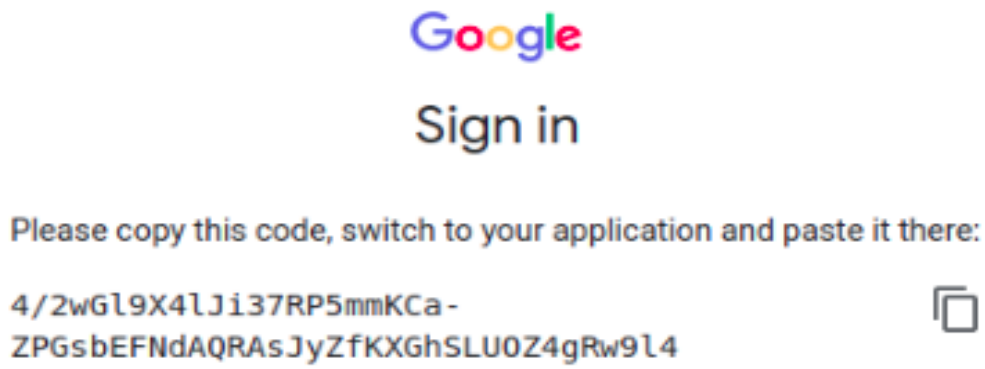

Figure 9: Authentication key

The successful mounting can be seen on the left panel of the notebook shown in the figure below.

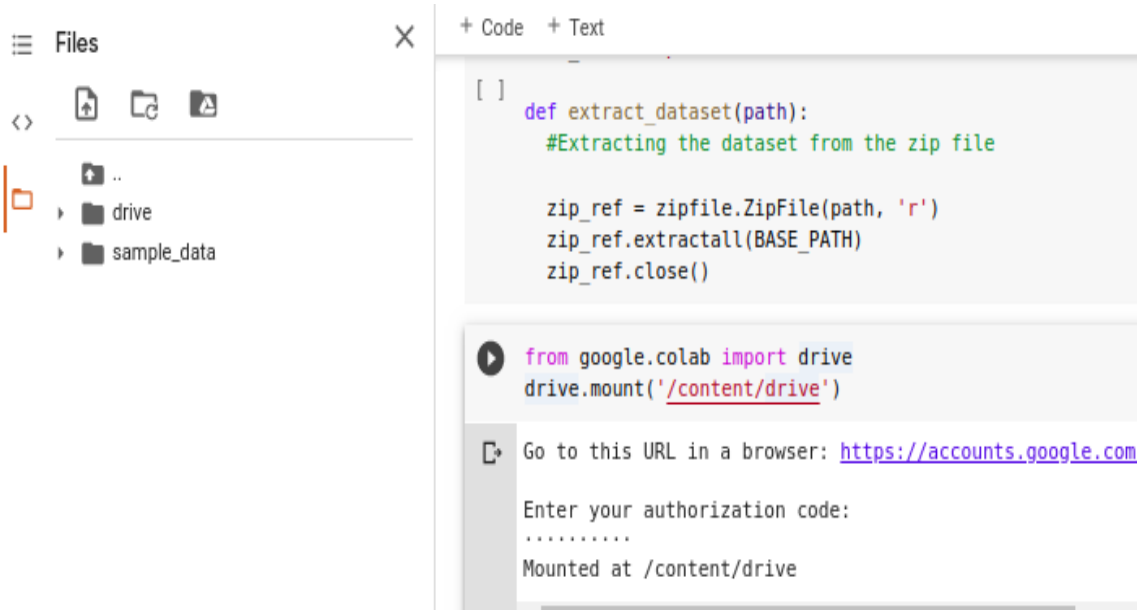

Figure 10: Drive mounted

7 Modelling and Training of Capsule Network

The following description is regarding the initialization of the Capsule Network and training. The tensorflow and keras version used are 1.15 and 2.2.4 respectively. The capsulelayers.py, LoadCapsuleNetworkModel.py and CapsuleNetwork.ipynb are used for initialization and training. The three modules mentioned above are used for the training of the Capsule Network.

Libraries used in capsulelayers.py

```
from keras import backend as K
import tensorflow as tf
import numpy as np
from keras import layers, initializers, regularizers,
constraints
from keras.utils import conv_utils
from keras.layers import InputSpec
from keras.utils.conv_utils import conv_output_length
```

Libraries used in LoadCapsuleNetworkModel.py

```
from keras import backend as K
from keras import optimizers, layers, models
from keras.layers import Input,Layer
from keras.layers import Reshape, BatchNormalization,
Dense, Deconvolution2D,Activation, Conv2D
from keras.callbacks import ModelCheckpoint, Callback
from keras.utils import plot_model
import tensorflow as tf
from capslayers import CapsuleLayer, CapsToScalars,
Conv2DCaps, FlattenCaps, ConvCapsuleLayer3D, Mask_CID,
ConvertToCaps, Mask
```

Both the modules are required to be loaded into Google drive which can be loaded into the colab environment with the below command shown below.

```
!cp /content/drive/My Drive/DeepCaps/capslayers.py
/content/
!cp /content/drive/My Drive/DeepCaps
/LoadCapsuleNetworkModel.py /content/
```

Libraries used in CapsuleNetwork.ipynb.

```
 import zipfile
import os
import cv2
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
import numpy as np
from keras import backend as K
from sklearn.model_selection import train_test_split
from keras import callbacks
from keras import optimizers
from keras.preprocessing.image import
ImageDataGenerator
from LoadCapsuleNetworkModel import Capsnet
from sklearn.metrics import matthews_corrcoef,
confusion_matrix, classification_report
import itertools
```

Loading the data into numpy arrays.

```
def transform_data(df,dir):
  '''
  Function that transforms the images for test train
  and validation into numpy arrays along with their
  label encodings
  '''
  temp_X = []
  temp_y = []
  path = '/content/Dataset/'+dir
  for index, row in df.iterrows():
    #image
    img = cv2.cvtColor(cv2.imread(path+'/'+
    row['Image']),cv2.COLOR_BGR2RGB)
    resize = cv2.resize(img,(128,128),interpolation =
    cv2.INTER_CUBIC)
    temp_X.append(resize)
    #labels
    temp_y.append(np.array([row['lung_aca'],
    row['lung_n'],row['lung_scc']]).astype('float32'))
  X = np.stack(temp_X)
  y = np.stack(temp_y)
  X = X.astype('float32')/255.
  return X,y
```

Initializing the Capsule Network model.

```
#Loading the model for training and evaluation
model, eval_model = Capsnet(input_shape=[128,128,3],
n_class=3, routings=3)
#Model summary
model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 128, 128, 3)	0	
conv2d_1 (Conv2D)	(None, 128, 128, 128	3584	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 128, 128, 128	512	conv2d_1[0][0]
convert_to_caps_1 (ConvertToCap	(None, 128, 128, 128	0	batch_normalization_1[0][0]

Figure 11: Capsule Network summary

Initialize the optimizer and callbacks for the Capsule Network.

```
#callbacks
log = callbacks.CSVLogger('CapsuleNetwork.csv')
checkpoint = callbacks.ModelCheckpoint(
'save_weights_ep_no_epoch:02d.h5', save_best_only=True,
save_weights_only=True, verbose=1)
lr_decay = callbacks.LearningRateScheduler(schedule
=lambda epoch:  0.0001 * np.exp(-epoch / 10.))
# compile the model
model.compile(optimizer=optimizers.Adam(lr=0.0001),
loss=[margin_loss, 'mse'], metrics='capsnet':
"accuracy")
```

Train the Capsule Network by calling fit.generator(). The augmentations are specified in the data generator for training.

```
#Training with data augmentation
model.fit_generator(generator =train_generator(x_train,
y_train, 32), steps_per_epoch=int(y_train.shape[0] /
32), epochs=30, validation_data=[[x_test, y_test],
[y_test, x_test]], callbacks=[log, checkpoint,
lr_decay])
#Saving the trained model
model.save_weights( '/content/drive/My
Drive/Research_Dataset/DeepCaps/
trained_model_CapsuleNetwork.h5')
print('Trained model saved to
ŕained_model_CapsuleNetwork.h5')
return model
```

8 Modelling and Training of CNN models

Tensorflow and keras libraries are used for defining the CNN models. The initialization of the pre-trained CNN architectures are done in the following steps:

Loading the LoadCNNModel.py this module contains all the definitions of the CNN models. The model definitions are specified in different functions, that can be invoked by passing the input size and number of output classes with the "imagenet" weights.

```
#Loading the LoadCNNModel.py in colab environment that
contains the model definitions
!cp /content/drive/My Drive/LoadCNNModel.py /content
```

Libraries used in the LoadCNNModel.py module that loads the model for training.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.applications.resnet50 import
ResNet50
from tensorflow.keras.applications.densenet import
DenseNet121
from tensorflow.keras.applications.vgg19 import VGG19
```

The RestNet-50,VGG19 and DenseNet121 model are defined in the loadRestNetModel(),loadVGG19Model() and loadDenseNetModel() function that takes the input size of the image and the number of output classes.

```
def loadRestNetModel(input_shape,out_shape):
  pre_trained_model = ResNet50(input_shape= input_shape,
                      # Shape of our images
                     include_top = False,
# Leave out the last fully connected layer
                     weights = 'imagenet')
  for layer in pre_trained_model.layers:
   layer.trainable = False
  x = layers.Flatten()(pre_trained_model.output)
  # Add a fully connected layer with 1,024 hidden
  units and ReLU activation
  x = layers.Dense(1024, activation='relu')(x)
  # Add a dropout rate of 0.2
  x = layers.Dropout(0.2)(x)
  # Add a final softmax layer for classification
  x = layers.Dense(out_shape, activation='softmax')(x)
  model = Model( pre_trained_model.input, x)
  return model
```

```
def loadVGG19Model(input_shape,out_shape):
  #new_input = layers.Input(shape=input_shape)
  modelvgg19 = VGG19(input_shape = input_shape,
        include_top=False,weights="imagenet")
  for layer in modelvgg19.layers:
      layer.trainable = False
  x = layers.Flatten()(modelvgg19.output)
  x = layers.Dense(1024,activation='relu')(x)
  x = layers.Dense(512,activation='relu')(x)
  x = layers.Dense(out_shape, activation='softmax')(x)
  model = Model(modelvgg19.input,x)
  return model
```

```
def loadDenseNetModel(input_shape,out_shape):
  pre_trained_model = DenseNet121(input_shape=input_shape,
                              # Shape of our images
                              include_top = False,
            # Leave out the last fully connected layer
                              weights = 'imagenet')
  for layer in pre_trained_model.layers:
      layer.trainable = False
  x = layers.Flatten()(pre_trained_model.output)
  # Add a fully connected layer with 1,024 hidden
  units and ReLU activation
  x = layers.Dense(1024, activation='relu')(x)
  # Add a dropout rate of 0.2
  x = layers.Dropout(0.2)(x)
  # Add a final softmax layer for classification
  x = layers.Dense (out_shape, activation='softmax')(x)
  model = Model( pre_trained_model.input, x)
  return model
```

There are three .ipynb files used for training of the CNN models. In each file the libraries used for training of each model are shown below.

```
import os
import numpy as np
import pandas as pd
import cv2
import tensorflow as tf
from tensorflow import keras
import zipfile
import seaborn as sns
from sklearn import metrics
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import
LearningRateScheduler,ModelCheckpoint, CSVLogger
```

Before initializing the models the module LoadCNNModel has to be imported for the CNN models to be initialized.

```
from LoadCNNModel import loadRestNetModel
from LoadCNNModel import loadDenseNetModel
from LoadCNNModel import loadVGG19Model
```

Initialization of the three CNN models.

```
#Initializing the VGG19 model
modelvgg19 = loadVGG19Model((224,224,3),3)
#Initializing the RestNet50 model
model = loadRestNetModel((224,224,3),3)
#Initializing the DenseNet model
model = loadDenseNetModel((224,224,3),3)
```

Initialization of data generator for training of the CNN models using augmentation. The augmentations to be done on the images are specified in the data generator function itself.

```
image_size = 224
dataGenerator = ImageDataGenerator(horizontal_flip=True,
            vertical_flip=True,rotation_range=90,
            zoom_range=[0.3,0.6],rescale=1/255.)
test_Data_Generator = ImageDataGenerator(rescale=1/255.)
trainGenerator = dataGenerator.flow_from_dataframe(train,
            directory='/content/Dataset/Train',
            target_size=(image_size, image_size),
            x_col="Image",
            y_col=['lung_aca','lung_n','lung_scc'],
            class_mode='raw',
            shuffle=False,
            subset='training',
            batch_size=32)
validGenerator = test_Data_Generator.flow_from_dataframe(
            val,
            directory='/content/Dataset/Val',
            target_size=(image_size,image_size),
            x_col="Image",
            y_col=['lung_aca','lung_n','lung_scc'],
            class_mode='raw',
            shuffle=False,
            batch_size=32) test_gen =
test_Data_Generator.flow_from_dataframe(test,
            directory='/content/Dataset/Test',
            target_size=(image_size,image_size),
            x_col="Image",
            y_col=['lung_aca','lung_n','lung_scc'],
            class_mode='raw',
            shuffle=False,
            batch_size=32)
```

In the last step we initialize the optimizer, the callbacks and execute the training for the models.

```
#Initilizing the optimizer
adam = Adam(lr=0.0001)
model.compile(loss='categorical_crossentropy',
optimizer=adam, metrics=['accuracy'])
```

```
#Initializing the callbacks
log = CSVLogger('/content/drive/My
Drive/Research_Dataset/DenseNet121/DenseNet121.csv')
lr_decay = LearningRateScheduler(schedule=lambda
epoch:  0.0001 * np.exp(-epoch / 10.))
mc = ModelCheckpoint('/content/drive/My
Drive/Research_Dataset/DenseNet121/densenet121_model.h5',
monitor='val_accuracy',
mode='max', verbose=1, save_best_only=True)
```

9 Evaluations

The evaluations for the models considered in the sutdy are For the evaluations the following libraries and are used:

To evaluate the model with MCC and the confusion matrix the module can be imported using:

```
from sklearn.metrics import matthews_corrcoef,
confusion_matrix,classification_report
```

To estimate the Matthew's Correlation Coefficient of a model.

```
#Estimating the Matthew's correlation matrix
mcc = matthews_corrcoef(y_actual, y_pred)
print("Matthews Correlation Coefficient:  ", mcc)
Output:
Matthews Correlation Coefficient:  0.9980009334422903
```

Plotting the train and validation loss and accuracy curve.

```
plt.figure(figsize=(20,10))
plt.plot(history['capsnet_acc'],color='black',
linewidth=3.0)
plt.plot(history['val_capsnet_acc'],color='blue',
linewidth=3.0)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

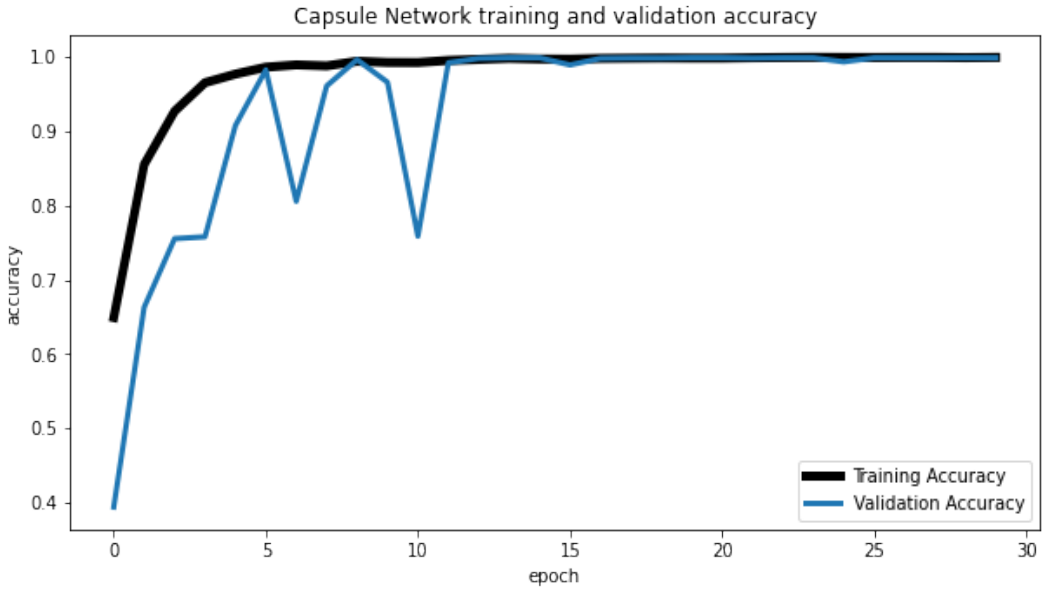

Figure 12: Train and validation accuracy curve

```
plt.figure(figsize=(20,10))
plt.plot(history['loss'], color='black',linewidth=3.0)
plt.plot(history['val_loss'],color='blue',linewidth=2.0)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

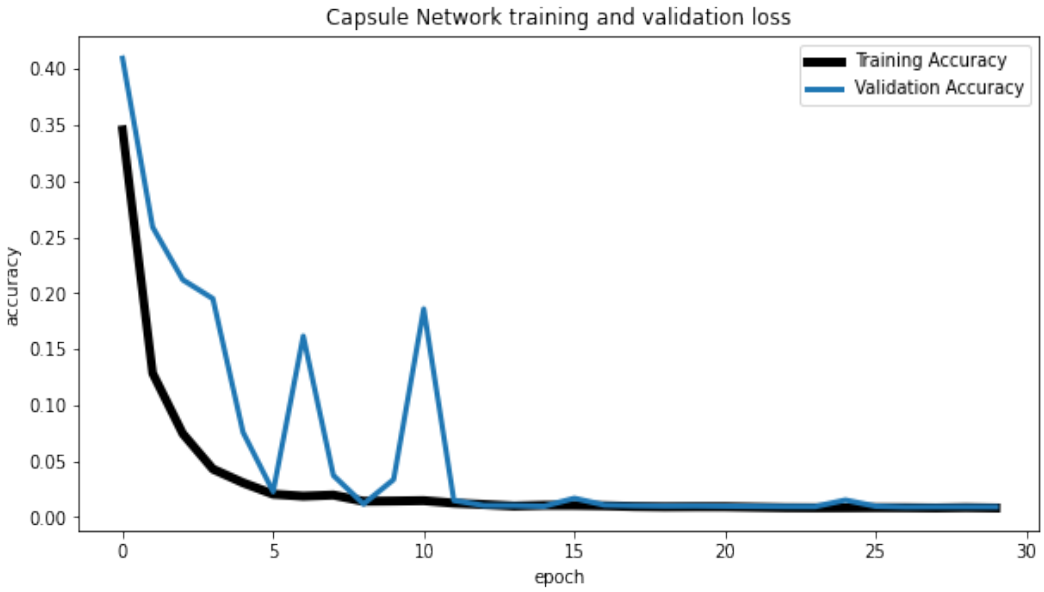

Figure 13: Train and validation loss curve

The execution of the below code plots the confusion matrix.

```
#Plotting the confusion matrix
plt.figure(figsize=(10,8))
ax= plt.subplot()
sns.heatmap(confusion_mtx.T, annot=True, ax =
ax,fmt='g',annot_kws="size":  17)
ax.xaxis.set_ticklabels(list(label_encodings['CancerType'
]))
ax.yaxis.set_ticklabels(list(label_encodings['CancerType'
]))
ax.xaxis.label.set_fontsize(15)
ax.yaxis.label.set_fontsize(15)
ax.set_xlabel('Actual Class')
ax.set_ylabel('Predicited Class')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

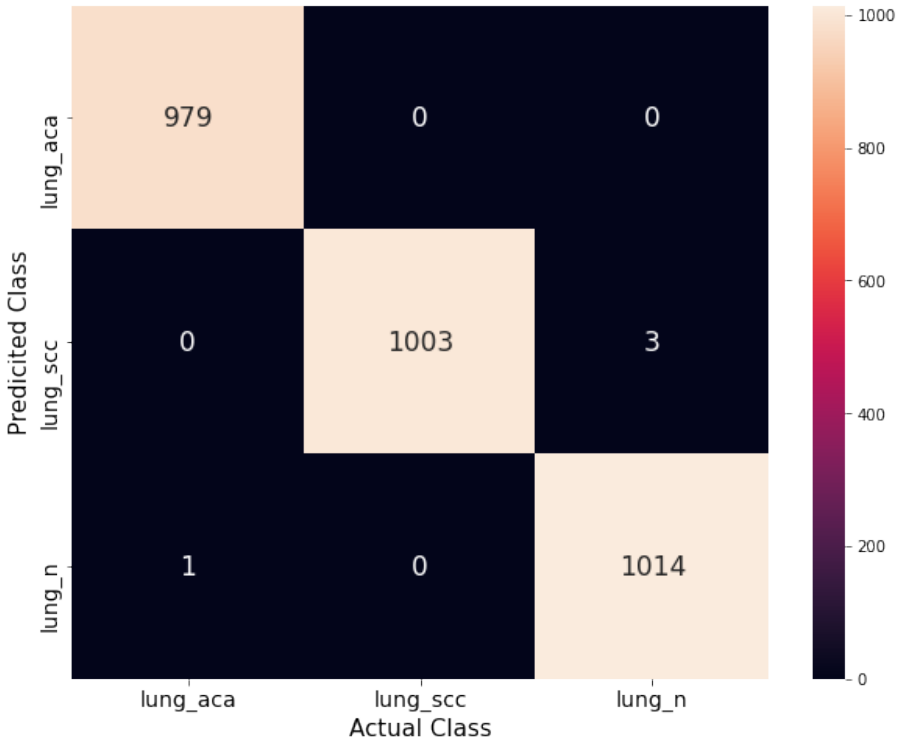

Figure 14: Confusion matrix

References

Borkowski, A. A., Bui, M. M., Thomas, L. B., Wilson, C. P., DeLand, L. A. and Mastorides, S. M. (2019). Lung and colon cancer histopathological image dataset (lc25000).