Configuration Manual

MSc Research Project
Data Analytics

Olawaunmi Sunday Anota
Student ID: x19239149

School of Computing
National College of Ireland

Supervisor: Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Olawaunmi Sunday Anota
Student ID:	x19239149
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Noel Cosgrave
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	758
Page Count:	10

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:	18th September 2021

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

Olawaunmi Sunday Anota
x19239149

1 Introduction

This configuration manual outlines the different software and hardware specifications and versions used in the research work "Comparative examination of deep learning and machine learning approaches in predicting radiation pneumonitis," as well as the processes for applying them. Future medical researchers would be able to easily reproduce this research for additional analysis and extension thanks to this work effort.
The following sections comprise this manual: The specifics of the environment setup are detailed in Section 2. The libraries needed to complete this project are discussed in Section 3. Section 4 contains all of the dataset's information. Section 5 describes how the models are implemented and provides details on the code repository and Section 6 evaluation of the performance of the models.

1.1 Project Overview

The goal of this research is to compare deep learning and machine learning techniques for predicting radiation pneumonitis in computed tomography images of Non-Small Cell Lung Cancer (NSCLC) patients.

2 System Specifications

The following are the requirements: The software and hardware setups and to train the model for such a big image data, a large GPU (Graphics Processing Unit) size is required.

2.1 Hardware Requirements

- **Operating System:** Windows 10 Home
- **Processor:** AMD Ryzen 7 4700U with Radeon Graphics 2.00 GHz
- **Installed RAM:** 8GB
- **System Type:** 64-bit operating system, x64-based processor
- **Graphical Processing Unit (GPU) :** Nvidia Tesla V100 -SXM2 16GB (Colab Pro)

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.42.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P0    23W / 300W |      0MiB / 16160MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
```

Figure 1: CUDA Version

2.2 Software Requirements

- **Programming Language :** Python version 3.6.9
- **Integrated Development Environment (IDE):** Google Colaboratory

2.3 Google Colaboratory Environment Setup

The project was developed entirely in Python and deployed on Google Colab. Google has already built up the environment in which the codes can be executed.

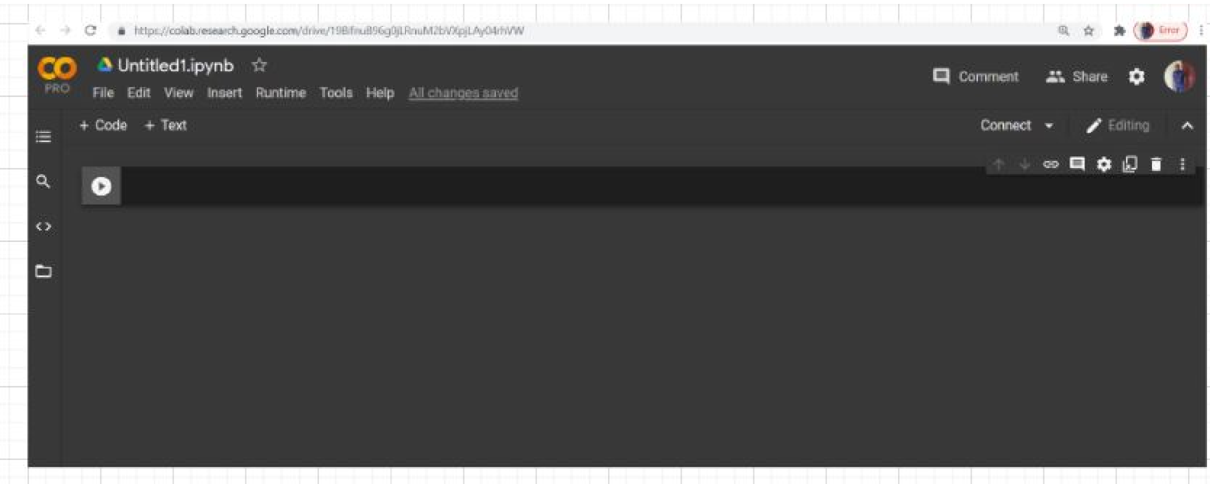


Figure 2: Colab Environment

3 Python Libraries Required

In Figure 3, all of the python libraries that were necessary to execute this research project are listed, along with the codes that is used to import. Installation of python libraries not required in Google Colab.

```python
import warnings
warnings.filterwarnings("ignore")

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt
import seaborn as sns

import os
from glob import glob
from tqdm import tqdm

from time import time

import cv2

import scipy.stats as stats

import keras
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras import initializers, layers, utils
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Layer, Input

from sklearn.utils import shuffle
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

from sklearn import svm
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

DEBUG = False

%matplotlib inline
```

Figure 3: Python Libraries

4 Data Selection

The 4D-Lung Image dataset used for this research project was published by Hugo et al. (2016) on The Cancer Imaging Archive (TCIA) website. The dataset consists of 347,330 images from 20 NSCLC cohort subjects CT scans. The data consist of Computed Tomography (CT) scans images of chemoradiotherapy.

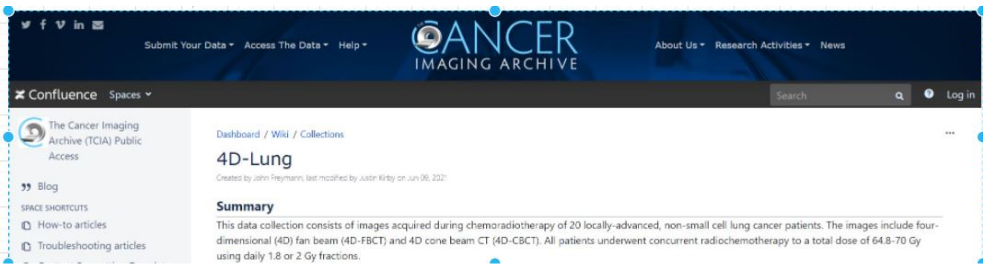


Figure 4: The Cancer Imaging Archive

4.1 Loading of Dataset

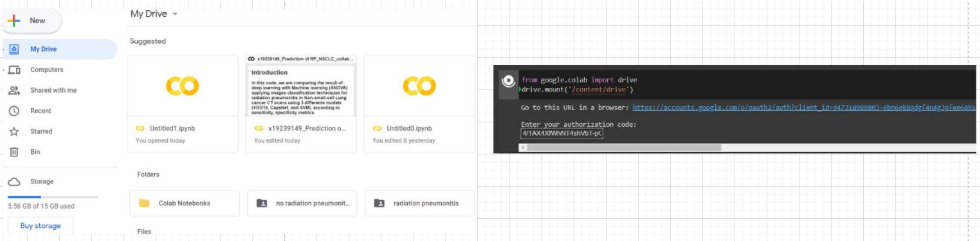


Figure 5: Mount Drive

The files are accessible since Google Drive is mounted[1] as shown in Figure 5.

```
[ ] rad_path = '/content/drive/MyDrive/radiation pneumonitis/'
    no_rad_path = '/content/drive/MyDrive/no radiation pneumonitis/'
```

Figure 6: Drive Folder

The folder directory is set to the location where the dataset is stored so that files may be retrieved from there.

4.2 Exploratory Data Analysis

Data Wrangling and Visualizes

```
[ ] # read and resize images function
    def __read_img(path, IMG_SIZE=(244, 244)):
        img = cv2.imread(path)
        img = cv2.resize(img, IMG_SIZE )
        #img = img /255.0

        return img
```

```
    # visualize samples function
    def plot_imgs(title, paths):
        fig = plt.figure(figsize=(15, 10), dpi=72)
        fig.suptitle(title, fontsize=20, y=1.05)

        for i, row in enumerate(paths):
            try:
                img = __read_img(row)
            except:
                img = row
            plt.subplot(3, 5, i+1)
            plt.xticks([])
            plt.yticks([])
            plt.grid(False)
            #plt.title(title)
            plt.imshow(img)
        plt.tight_layout()
        plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None)
        plt.show()
```

```
[ ] # show some Radiation Pneumonitis samples
    plot_imgs('Radiation Pneumonitis', rad_imgs_path[:15])
```

Figure 7: Data Wrangling and Visuals

This is done to get more information about the data.

[1]https://drive.google.com/drive/my-drive

5 Data Pre-processing: Deep Learning

```
# set some hyperparameters
RANDOM_SEED = 42
IMG_SIZE = (224, 224) # size of vgg16 input
BATCH_SIZE = 32
TRAIN_STEPS = 16
VALID_STEPS = 8
EPOCHS = 25

N_FOLDS = 5 # YOU CAN CHANGE N_FOLDS HERE

input_shape = IMG_SIZE + (3, ) #(244, 244, 3)
n_class = 2 # no. of class [0.0, 1.0]
```

```
# combine all data
imgs = []
labels = []

if DEBUG:
    rad_imgs_path = rad_imgs_path[:100]
    no_rad_imgs_path = no_rad_imgs_path[:100]

for im_path in tqdm(rad_imgs_path, total=len(rad_imgs_path)):
    img = __read_img(im_path, IMG_SIZE)
    label = 1

    imgs.append(img)
    labels.append(label)

for im_path in tqdm(no_rad_imgs_path, total=len(no_rad_imgs_path)):
    img = __read_img(im_path, IMG_SIZE)
    label = 0

    imgs.append(img)
    labels.append(label)
```

```
100%|██████████| 1062/1062 [04:46<00:00,  3.71it/s]
100%|██████████| 637/637 [02:47<00:00,  3.80it/s]
```

```
imgs, labels = np.array(imgs), utils.to_categorical(np.array(labels), n_class)
imgs.shape, labels.shape
```

```
((1699, 224, 224, 3), (1699, 2))
```

Figure 8: Data Pre-processing Deep Learning

In this section the hyperparamters are set to run deep learning models.

Data Augmentation

```
# Data Augmentation and generator
datagen_train = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    fill_mode = "nearest",
    brightness_range=[0.5, 1.25],
    horizontal_flip=True,
    vertical_flip=True,
)

datagen_valid = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
)


# fit imgs to data generator
datagen_train.fit(imgs)
```

```
def train_valid_generators(trn_idx, val_idx):
    train_imgs, valid_imgs = imgs[trn_idx], imgs[val_idx]
    train_labels, valid_labels = labels[trn_idx], labels[val_idx]

    # split data to train and valid generators
    train_generator = datagen_train.flow(
        train_imgs, train_labels,
        batch_size=BATCH_SIZE,
        shuffle=True,
        seed=RANDOM_SEED
    )

    validation_generator = datagen_valid.flow(
        valid_imgs, valid_labels,
        batch_size=int(BATCH_SIZE/2),
        shuffle=False,
        seed=RANDOM_SEED
    )

    return train_generator, validation_generator
```

Figure 9: Image Augmentation

Horizontal flip, vertical flip, height shift range, width shift range, rescale, shear range, and data was separated into train generator and validation generator are all utilized in the code above.

5.1 Data Preprocessing: Machine Learning

As illustrated from Figure 10 below, the data is transformed from a 4D image to 2D image, thereafter pricncipal component analysis is applied as feature extraction.

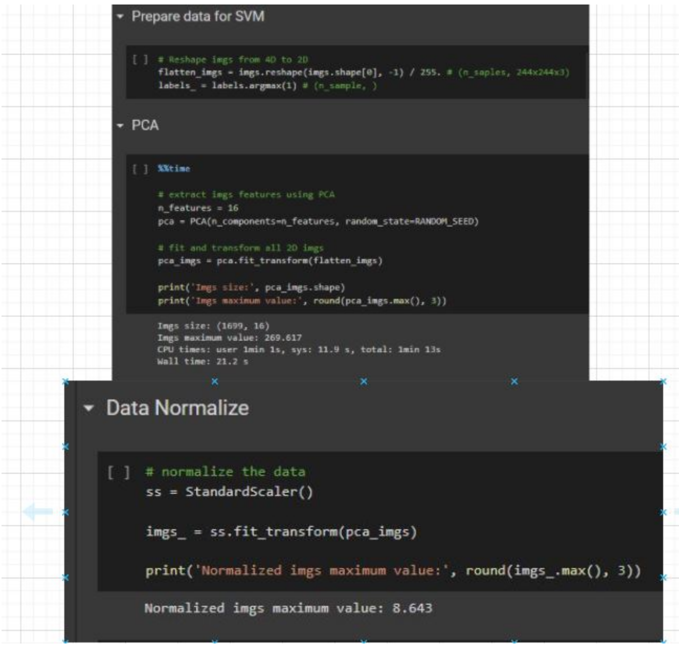


Figure 10: Data Preprocessing(SVM)

6 Implementation of Models

VGG16, CapsuleNet, and SVM were the three models utilized.

6.1 VGG16

The code for VGG16 pre-trained on the ImageNet dataset is shown in Figure 11. Adam is the optimizer that was utilized. reLU and softmax are the activation functions utilized. Because there are only two classes in the dataset, the loss function is binary crossentropy. VGG16.ipynb is the file to execute; it was written in Google Colab and performed using GPU.

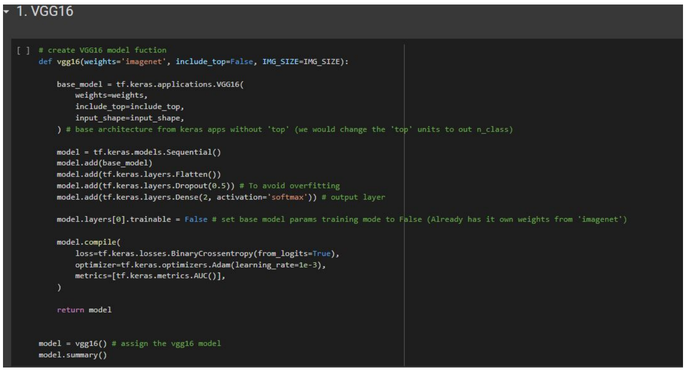
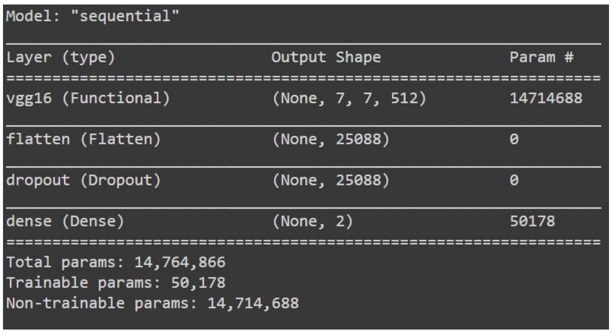
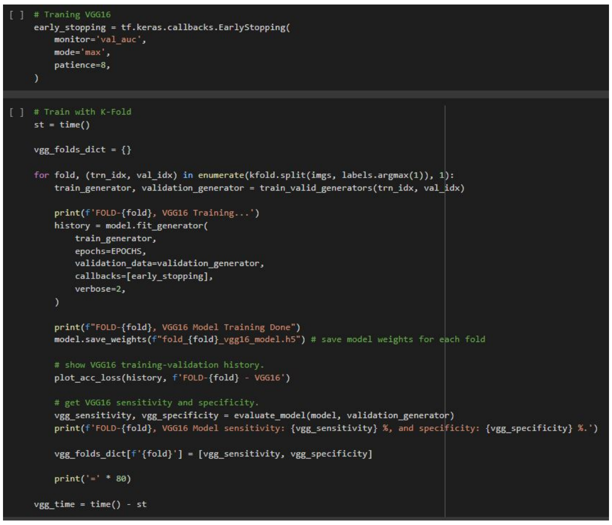


Figure 11: Building VGG16 model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
vgg16 (Functional)          (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)           (None, 25088)             0
_________________________________________________________________
dropout (Dropout)           (None, 25088)             0
_________________________________________________________________
dense (Dense)               (None, 2)                 50178
=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
```

Figure 12: Summary of VGG16 model



```
[ ] # Traning VGG16
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor='val_auc',
        mode='max',
        patience=8,
    )

[ ] # Train with K-Fold
    st = time()

    vgg_folds_dict = {}

    for fold, (trn_idx, val_idx) in enumerate(kfold.split(imgs, labels.argmax(1)), 1):
        train_generator, validation_generator = train_valid_generators(trn_idx, val_idx)

        print(f'FOLD-{fold}, VGG16 Training...')
        history = model.fit_generator(
            train_generator,
            epochs=EPOCHS,
            validation_data=validation_generator,
            callbacks=[early_stopping],
            verbose=2,
        )

        print(f"FOLD-{fold}, VGG16 Model Training Done")
        model.save_weights(f"fold_{fold}_vgg16_model.h5") # save model weights for each fold

        # show VGG16 training-validation history.
        plot_acc_loss(history, f'FOLD-{fold} - VGG16')

        # get VGG16 sensitivity and specificity.
        vgg_sensitivity, vgg_specificity = evaluate_model(model, validation_generator)
        print(f'FOLD-{fold}, VGG16 Model sensitivity: {vgg_sensitivity} %, and specificity: {vgg_specificity} %.')

        vgg_folds_dict[f'{fold}'] = [vgg_sensitivity, vgg_specificity]

        print('=' * 80)

    vgg_time = time() - st
```

Figure 13: Training of VGG16 model

6.2 CapsuleNet

The code for CapsuleNet built from scratch is shown in Figure 13. Adam is the optimizer that was utilized. reLU and softmax are the activation functions utilized. Because there are only two classes in the dataset, the loss function is binary crossentropy. CapsNet.ipynb is the file to execute; it was written in Google Colab and performed using GPU.

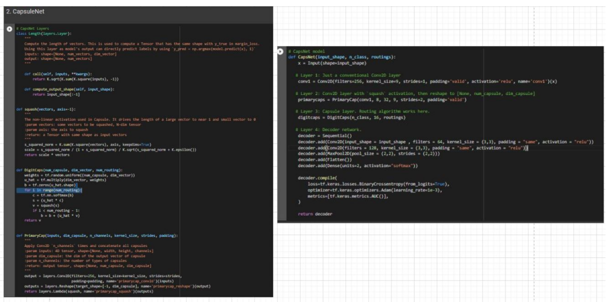


Figure 14: Building of CapsNet Model

```
[ ] # define CapsNet model
    capsNmodel = CapsNet(input_shape=input_shape, n_class=n_class, routings=1)
    capsNmodel.summary()

    Model: "sequential_1"
    _________________________________________________________________
    Layer (type)                 Output Shape              Param #
    =================================================================
    conv2d (Conv2D)              (None, 224, 224, 64)      1792
    _________________________________________________________________
    conv2d_1 (Conv2D)            (None, 224, 224, 128)     73856
    _________________________________________________________________
    max_pooling2d (MaxPooling2D) (None, 112, 112, 128)     0
    _________________________________________________________________
    flatten_1 (Flatten)          (None, 1605632)           0
    _________________________________________________________________
    dense_1 (Dense)              (None, 2)                 3211266
    =================================================================
    Total params: 3,286,914
    Trainable params: 3,286,914
    Non-trainable params: 0
    _________________________________________________________________
```

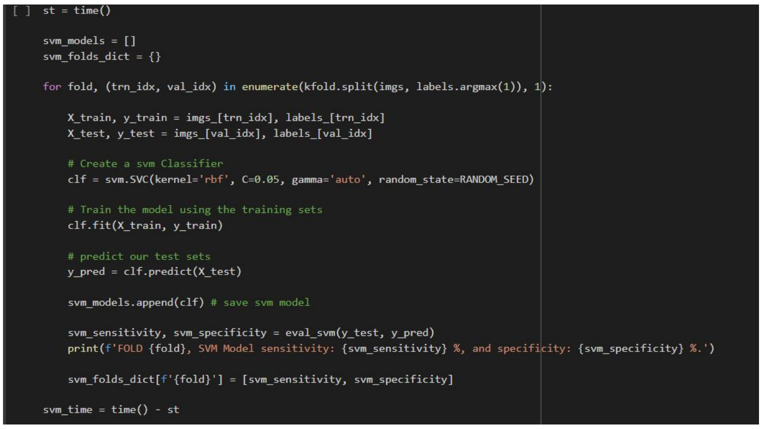
Figure 15: Summary of CapsNet Model

```
# Train CapsNet with K-Fold
st = time()

cap_folds_dict = {}

for fold, (trn_idx, val_idx) in enumerate(kfold.split(imgs, labels.argmax(1)), 1):
    train_generator, validation_generator = train_valid_generators(trn_idx, val_idx)

    print(f'FOLD-{fold}, CapsNet Training...')
    history = capsNmodel.fit_generator(
        train_generator,
        epochs=EPOCHS,
        validation_data=validation_generator,
        callbacks=[early_stopping],
        verbose=2,
    )

    print(f"FOLD-{fold}, CapsNet Model Training Done")
    capsNmodel.save_weights(f"fold_{fold}_CapsNet_model.h5") # save model weights for each fold

    # show CapsNet training-validation history.
    plot_acc_loss(history, f'FOLD-{fold} - CapsNet')

    # get CapsNet sensitivity and specificity.
    cap_sensitivity, cap_specificity = evaluate_model(capsNmodel, validation_generator)
    print(f'FOLD-{fold}, CapsNet Model sensitivity: {cap_sensitivity} %, and specificity: {cap_specificity} %.')

    cap_folds_dict[f'{fold}'] = [cap_sensitivity, cap_specificity]

    print('=' * 80)

cap_time = time() - st
```

Figure 16: Training of CapsNet Model

6.3 SVM

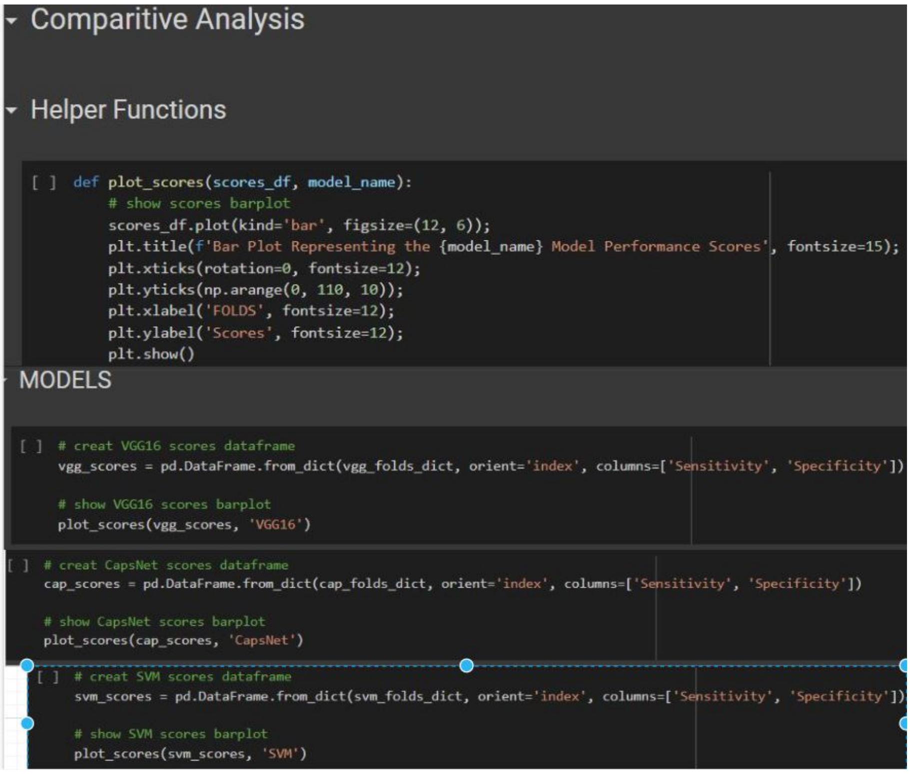
SVM.ipynb is the file to run. The model is built using the Figure17 below. The svm.SVC () function from the sklearn package was used to run the SVM classifcation model. The clf.predict() function was used in predicting RP in NSCLC from the classifier.



```
[ ] st = time()

    svm_models = []
    svm_folds_dict = {}

    for fold, (trn_idx, val_idx) in enumerate(kfold.split(imgs, labels.argmax(1)), 1):

        X_train, y_train = imgs_[trn_idx], labels_[trn_idx]
        X_test, y_test = imgs_[val_idx], labels_[val_idx]

        # Create a svm Classifier
        clf = svm.SVC(kernel='rbf', C=0.05, gamma='auto', random_state=RANDOM_SEED)

        # Train the model using the training sets
        clf.fit(X_train, y_train)

        # predict our test sets
        y_pred = clf.predict(X_test)

        svm_models.append(clf) # save svm model

        svm_sensitivity, svm_specificity = eval_svm(y_test, y_pred)
        print(f'FOLD {fold}, SVM Model sensitivity: {svm_sensitivity} %, and specificity: {svm_specificity} %.')

        svm_folds_dict[f'{fold}'] = [svm_sensitivity, svm_specificity]

    svm_time = time() - st
```

Figure 17: Svm Model

7 Comapartive Analysis of Model



```
Comparitive Analysis

Helper Functions

[ ] def plot_scores(scores_df, model_name):
        # show scores barplot
        scores_df.plot(kind='bar', figsize=(12, 6));
        plt.title(f'Bar Plot Representing the {model_name} Model Performance Scores', fontsize=15);
        plt.xticks(rotation=0, fontsize=12);
        plt.yticks(np.arange(0, 110, 10));
        plt.xlabel('FOLDS', fontsize=12);
        plt.ylabel('Scores', fontsize=12);
        plt.show()

MODELS

[ ] # creat VGG16 scores dataframe
    vgg_scores = pd.DataFrame.from_dict(vgg_folds_dict, orient='index', columns=['Sensitivity', 'Specificity'])

    # show VGG16 scores barplot
    plot_scores(vgg_scores, 'VGG16')

[ ] # creat CapsNet scores dataframe
    cap_scores = pd.DataFrame.from_dict(cap_folds_dict, orient='index', columns=['Sensitivity', 'Specificity'])

    # show CapsNet scores barplot
    plot_scores(cap_scores, 'CapsNet')

[ ] # creat SVM scores dataframe
    svm_scores = pd.DataFrame.from_dict(svm_folds_dict, orient='index', columns=['Sensitivity', 'Specificity'])

    # show SVM scores barplot
    plot_scores(svm_scores, 'SVM')
```

Figure 18: K-fold model comparison

The above Figure 18 shows the barplot of models 5 fold cross validation scores performance.

References

Hugo, G., Weiss, E., Sleeman, W., Balik, S., Keall, P., Lu, J. and Williamson, J. (2016). A longitudinal four-dimensional computed tomography and cone beam computed tomography dataset for image-guided radiation therapy research in lung cancer, *Medical Physics* **44**.