DEEP LEARNING MODEL TO CLASSIFY ADINOMA AND ADINOCARCINOMA

TABLE OF CONTENTS

List of Figures

List of Tables

ABBREVIATIONS

CNN	-	Convolution Neural Network
RNN	-	Recursive Neural Network
WSI	-	Whole Slide Images
LSTM	-	Long-Short term Memory
RAM	-	Random Access Memory
GPU	-	Graphics Processing Unit
AUC	-	Area Under the Curve
ROC curve	-	Receiver Operating Characteristic curve

ABSTRACT

Computerized microscopic image analysis plays an important role in computer related diagnosis and prognosis. Machine learning techniques have powered many aspects of medical investigation and clinical practices. Recently, deep learning is emerging as a leading machine learning tool in computer vision and has attracted considerable attention in biomedical image analysis. Stomach and colon cancers are amongst the most common leading causes of cancer deaths in the world, with stomach cancer ranking fourth in men and seventh in women, and colon cancer ranking third in men and second in women. This problem can be solved by training a machine learning model to classify cancers. In this project two models will be created using different algorithms and deep learning networks. Utilizing aggregation techniques such as max-pooling (MP-aggr) and RNN (RNN-aggr), the models can be trained with recurrent neural networks (RNN) and convolution neural networks (CNN) on histopathological whole-slide images (WSI). WSIs are image samples that are obtained through specialized scanning devices that are comparable to microscopic images in primary diagnosis. The models were developed to categorize WSIs into different tumour categories. The outcomes show how our algorithms may be used in the medical sector on a regular basis to classify the various tumours in a histopathological WSI.

KEYWORDS: Recurrent Neural Network (RNN), Convolution Neural Network (CNN), Whole-Slide Images (WSI), Max Pooling, RNN Aggregation, colon ca

CHAPTER 1
SUMMARY OF THE BASE PAPER

1.1 PAPER DETAILS

Authors	:	Amu Iizuka, Fahdi Kanavati, Kei Kato, Michael Rambeau, KojiArihiro & Masayuki Tsuneki
Publisher	:	Nature – Scientific reports
Published on	:	30 January 2020
Impact factor	:	3.998
Indexing	:	SCIE
DOI	:	https://doi.org/10.1038/s41598-020-58467-9

1.2 INTRODUCTION

Colon and stomach cancers are among the most common tumours worldwide, according to a new analysis on global cancer statistics. In 2018, stomach cancer caused 8.2% of all cancer deaths worldwide, whereas colon cancer caused 9.2% of all cancer deaths. Among the other cancer types, stomach cancer is the third most common cause of death, followed by colon cancer. For the histopathological examination of specimens in a typical pathological diagnosis, light microscopy is used.

Whole slide images (WSIs) are image samples acquired using specialised scanning equipment that can serve as a substitute for microscopic pictures for the purposes of first diagnosis. The introduction of WSIs paved the way for image analysis methods in the medical sector. Pathologists can analyse WSIs and diagnose cancer with the aid of machine learning and deep learning techniques. Deep CNNs in particular have demonstrated excellent performance in a wide range of computer vision and medical image analysis applications.

Segmentation and cancer categorization are examples of promising computational pathology applications. Installing deep learning-based tools and workflow systems can enable pathologists to work more quickly, read images accurately, and with fewer errors by installing these programmes.

1.3 SYSTEM MODEL FOR BASE PAPER:

The WSIs are initially pre-processed and augmented into a number of 512x512 pixel pictures. The model is trained using a variety of augmentation approaches that were imported from the Python libraries. For classification into adenoma and adenocarcinoma, the pictures are loaded into the proposed architecture model.
Some of the augmentation techniques that are used are scaling, hue saturation offsets, rotating, shear, contrast and brightness variations. The images were also rescaled between the values of [-1.0, 1.0].

In order to classify the WSI images various parameters were taken into consideration namely depth multiplier of 0.35 with 625k iterations and a batch size of 128. The model also had a least learning rate of 0.001 and most learning rate of 0.05. The model also used the Adam optimisation technique, with values for momentum, epsilon, and decay rate of 0.9, 1.0, and 1.0 respectively. In addition to these methods, the weighted cross entropy loss function was used to obtain the lowest loss value attainable.

The pathologists collected millions of tiles from WSI pictures and classified them into a tile consisting of two labels, adenoma and adenocarcinoma, after applying the aforementioned procedures. This model was created utilising the Inception V3 architecture. Then, these classified images from the CNN model are aggregated utilising two distinct techniques. There are two types of aggregation: 1) Max-Pooling and 2) LSTM RNN.
Each WSI picture is given the label with the highest probability from all of its samples in the max pooling aggregation technique, whereas in the LSTM Recursive Neural Network (RNN) aggregation, the model was trained to incorporate the given information from all of the samples. Deep Convolution Neural Network (CNN) characteristics were used as input in this.

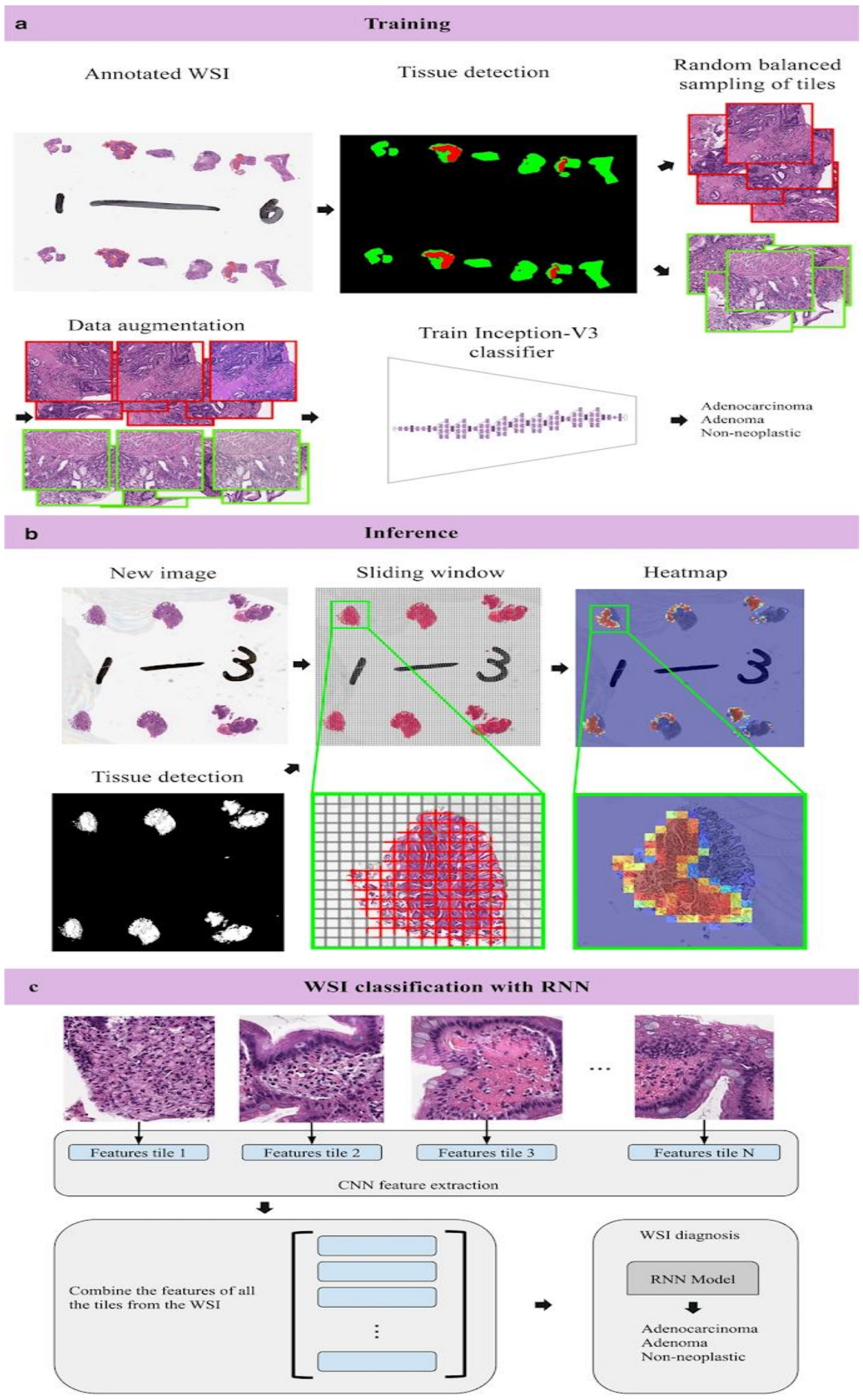


Fig 1.1: Summary of pipeline. (a) From WSI images of the training set, augmentation is done and inception v3 is trained. (b) A picture example of inference based on a new Whole Slide Image (WSI). (c) WSI using RNN.

CHAPTER 2
MERITS AND DEMERITS

2.1 MERITS

1. Time complexity for creating the model is lesser compared to the diagnosis done by the pathologists via observing through the microscope.

2. The accuracy of classifying the images by the pathologists only increases based on the experience gained by each pathologist.

3. The time taken for the model to be trained is directly proportional to the accuracy it yields.

4, The model could act as a second person to evaluate the results or warn the pathologist in case of different output on primary diagnosis and hence avoiding human errors.

2.2 DEMERITS

1. Huge size of WSI takes a lot of computational resources to process such as RAM and GPU space.

2. Every epoch takes a huge amount of time to train the model

3. Varying architecture and augmentation techniques show a big variation in the accuracy of the model.

CHAPTER 3
DATASET

3.1 About dataset used

Dataset given has data labelled whether it is adenocarcinoma and adenoma. The dataset [13] [14] which contains colon cancer WSI (adenoma and adenocarcinoma) was used to train the models. The images consist of 96x96 pixel images. A total of ~2.2 lakhs images were present from it. This Dataset is derived from 400 Whole Slide Images(WSI).

3.2 Stage 1: Image Pre-processing

The dataset that has been given consists of WSIs. So in order to get accurate images for the pathologist to diagnose the images given in the dataset are converted into 1s and 0s or true or false.

3.3 Stage 2: Image Classification

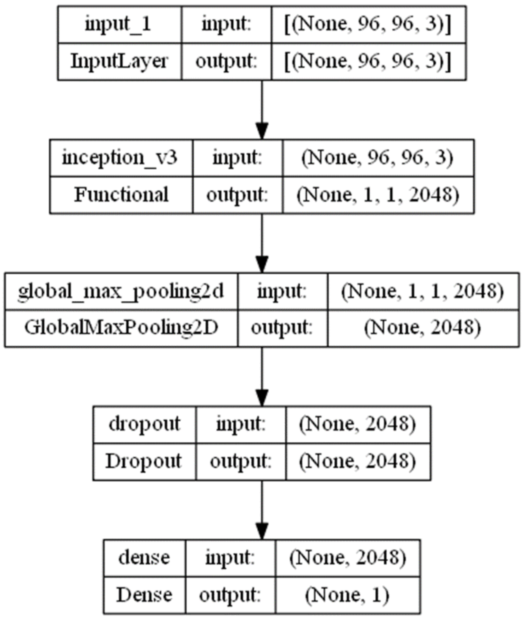
These pre-processed images are then classified into adenocarcinoma as 1 and adenoma as 0 and are stored in an excel sheet. Once the images have been classified the max-pooling aggregation and RNN aggregation is performed in each of the models.

CHAPTER 4
PROPOSED METHODOLOGY

4.1 Platform used

The model was trained with the help of Kaggle notebook kernel which had a maximum RAM size of 16 GB when GPU was not utilized and 13 GB when GPU was utilized. The GPU had a size of 16 GB. Python language was used to train the models.

4.1.1 Model 1 - Inception V3 architecture



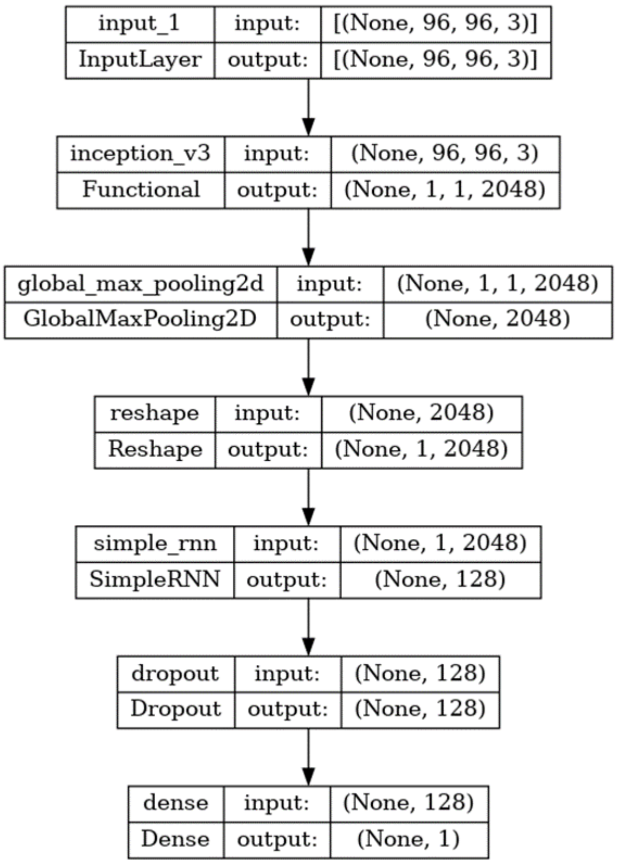
input_1 | input: | [(None, 96, 96, 3)]
InputLayer | output: | [(None, 96, 96, 3)]
inception_v3 | input: | (None, 96, 96, 3)
Functional | output: | (None, 1, 1, 2048)
global_max_pooling2d | input: | (None, 1, 1, 2048)
GlobalMaxPooling2D | output: | (None, 2048)
dropout | input: | (None, 2048)
Dropout | output: | (None, 2048)
dense | input: | (None, 2048)
Dense | output: | (None, 1)

Fig 4.1: Model 1-layer structure

The classified images from the excel sheet are retained into a variable that are later used for image augmentation to improve the model accuracy. The augmentations that we used in the following given model are random horizontal and vertical flips, maximum rotation range of 90 degrees, zoom, width and height shift and shearing of images.

Compared to that of the standard CNN model, Inception v3 architecture is best efficient in time complexity. Secondly, inception v3 architecture can utilize the imagenet package in a well efficient manner. Here the images are trained with a maximum of 16 epochs. GPU is utilized in kaggle and took around 15 mins to train the model for each epoch.

4.1.2 Model 2 - InceptionV3 with Simple RNN

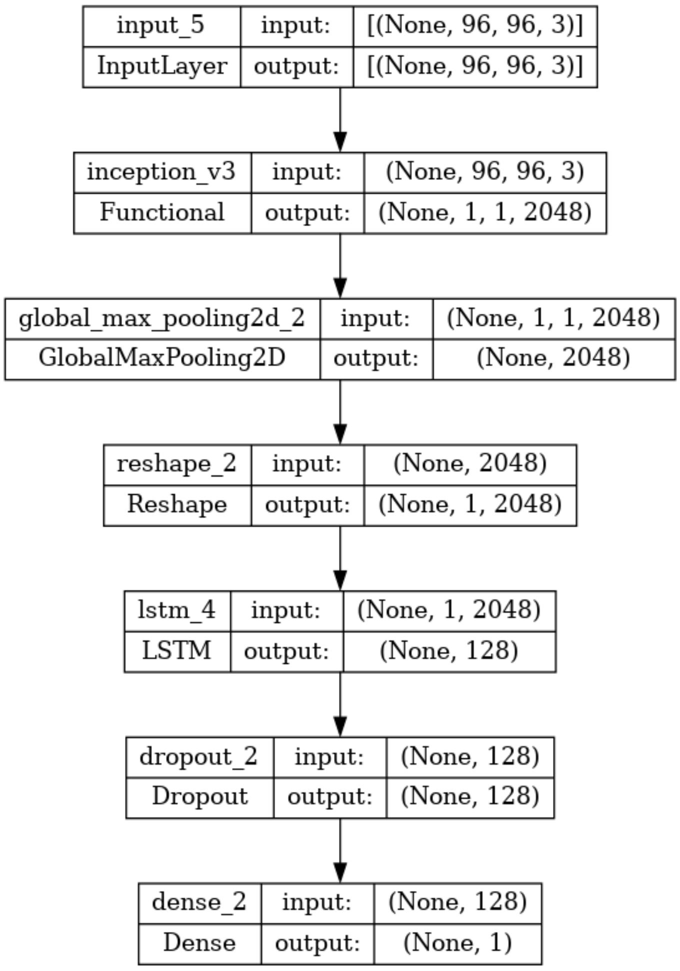


input_1 | input: | [(None, 96, 96, 3)]
InputLayer | output: | [(None, 96, 96, 3)]
inception_v3 | input: | (None, 96, 96, 3)
Functional | output: | (None, 1, 1, 2048)
global_max_pooling2d | input: | (None, 1, 1, 2048)
GlobalMaxPooling2D | output: | (None, 2048)
reshape | input: | (None, 2048)
Reshape | output: | (None, 1, 2048)
simple_rnn | input: | (None, 1, 2048)
SimpleRNN | output: | (None, 128)
dropout | input: | (None, 128)
Dropout | output: | (None, 128)
dense | input: | (None, 128)
Dense | output: | (None, 1)

Fig 4.2: Model 2-layer structure

Similar to that of model 1, the data augmentation process remains the same. Only the hyperparameters and the architecture used changes, we have added Reshape and Simple RNN layers to it. Reshape function is useful when we need to change the dimensions of an array, for example, when we want to convert a one-dimensional array into a two-dimensional array or vice versa. It can also be used to create arrays with a specific shape, such as matrices and tensors. In the case of using a simple RNN for image classification, the approach would be to treat each pixel in an image as a sequence of values, where the values represent the pixel intensity or color. This would result in a sequence of values for each pixel location in the image. The RNN would then be trained on these sequences to learn patterns in the data that can be used for classification.

4.1.3 Model 3 - InceptionV3 with LSTM



input_5 | input: | [(None, 96, 96, 3)]
InputLayer | output: | [(None, 96, 96, 3)]
inception_v3 | input: | (None, 96, 96, 3)
Functional | output: | (None, 1, 1, 2048)
global_max_pooling2d_2 | input: | (None, 1, 1, 2048)
GlobalMaxPooling2D | output: | (None, 2048)
reshape_2 | input: | (None, 2048)
Reshape | output: | (None, 1, 2048)
lstm_4 | input: | (None, 1, 2048)
LSTM | output: | (None, 128)
dropout_2 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_2 | input: | (None, 128)
Dense | output: | (None, 1)

Fig 4.3: Model 3-layer structure

This model is similar to model 2. We have replaced the SimpleRNN with the LSTM(Long Short Term Memory). A proper RNN structure had been generated by using the LSTM (Long Short-Term Memory) architecture. This is used to make sure that it doesn’t forget longer sequences and stores them in memory with the help of gates, whereas in SimpleRNN it can’t remember longer sequences. This helps the model retain important information. The LSTM model is then trained, similar to other models mentioned above, on the sequences to identify patterns to classify the images into Adenoma or Adenocarcinoma.

CHAPTER 5

SOURCE CODE

5.1 Model 1 - Inception V3 architecture

```
%matplotlib inline

!pip install livelossplot

# Imports
import numpy as np
import pandas as pd
from glob import glob
import os
import shutil
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix,
classification_report, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import seaborn as sn
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout, Flatten, Dense, GlobalMaxPooling2D, Input,
Concatenate
from keras.models import Model, Sequential
from keras.callbacks import CSVLogger, ModelCheckpoint
from keras.optimizers import Adam
from keras.applications.inception_v3 import InceptionV3
from livelossplot import PlotLossesKeras

# Hyperparams
SAMPLE_COUNT = 85000
TRAINING_RATIO = 0.9
IMAGE_SIZE = 96
EPOCHS = 16
BATCH_SIZE = 216
VERBOSITY = 1
TESTING_BATCH_SIZE = 5000

# Output files
TRAINING_LOGS_FILE = "training_logs.csv"
MODEL_FILE = "histopathologic_cancer_detector.h5"
TRAINING_PLOT_FILE = "training.png"
VALIDATION_PLOT_FILE = "validation.png"
ROC_PLOT_FILE = "roc.png"
MODEL_LAYERS_SIMPLIFIED = "model_plot_simplified.png"
MODEL_LAYERS = "model_plot.png"
INPUT_DIR = '../input/histopathologic-cancer-detection/'
training_path = '../training'
```

```
validation_path = '../validation'

# Read the train labels and import as DataFrame
training_dir = INPUT_DIR + 'train/'
data_frame = pd.DataFrame({'path': glob(os.path.join(training_dir,'*.tif'))})
data_frame['id'] = data_frame.path.map(lambda x: x.split('/')[4].split('.')[0])
labels = pd.read_csv(INPUT_DIR + 'train_labels.csv')
data_frame = data_frame.merge(labels, on = 'id')
negatives = data_frame[data_frame.label == 0].sample(SAMPLE_COUNT)
positives = data_frame[data_frame.label == 1].sample(SAMPLE_COUNT)
data_frame = pd.concat([negatives, positives]).reset_index()
data_frame = data_frame[['path', 'id', 'label']]

# Create folder for training and validation and sub folders for splitting the data
# as binary classification

for folder in [training_path, validation_path]:
    for subfolder in ['0', '1']:
        path = os.path.join(folder, subfolder)
        os.makedirs(path, exist_ok=True)

# Split the data according to TRAINING_RATIO  and copy the files to destination
folders

training, validation = train_test_split(data_frame, train_size=TRAINING_RATIO,
stratify=data_frame['label'])
data_frame.set_index('id', inplace=True)
for images_and_path in [(training, training_path), (validation, validation_path)]:
    images = images_and_path[0]
    path = images_and_path[1]
    for image in images['id'].values:
        file_name = image + '.tif'
        label = str(data_frame.loc[image,'label'])
        destination = os.path.join(path, label, file_name)
        if not os.path.exists(destination):
            source = os.path.join(INPUT_DIR + 'train', file_name)
            shutil.copyfile(source, destination)

# Data Augmentation and Reading the reading the images
# Data generation
training_generator = ImageDataGenerator(rescale=1./255,
                                        horizontal_flip=True,
                                        vertical_flip=True,
                                        rotation_range=90,
                                        zoom_range=0.2,
                                        width_shift_range=0.1,
                                        height_shift_range=0.1,
                                        shear_range=0.05,
                                        channel_shift_range=0.1).flow_from_directory(training_path,
                                    target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                batch_size=BATCH_SIZE,
                                class_mode='binary')
validation=ImageDataGenerator(rescale=1./255).flow_from_directory(validation_path,
                                                          target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                              batch_size=BATCH_SIZE,
                                                              class_mode='binary')

# Defining the layers of CNN
input_shape = (IMAGE_SIZE, IMAGE_SIZE, 3)
model = Sequential()
model.add(Input(input_shape))
model.add(InceptionV3(include_top=False, input_shape=input_shape))
model.add(GlobalMaxPooling2D())
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# compile the model
model.compile(optimizer=Adam(learning_rate=0.001,
decay=0.0001,beta_1=0.9,epsilon=1.0),
           loss='binary_crossentropy',
           metrics=['accuracy'])
model.summary()

# Plot the model layer and save it as a file
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
SVG(model_to_dot(model).create(prog='dot', format='svg'))

from keras.utils.vis_utils import plot_model
plot_model(model, to_file=MODEL_LAYERS_SIMPLIFIED, show_shapes=True,
show_layer_names=True, expand_nested=False)
plot_model(model, to_file=MODEL_LAYERS, show_shapes=True,
show_layer_names=True, expand_nested=True)

# Model Training
history = model.fit(training_generator,
                        steps_per_epoch=len(training_generator),
                        validation_data=validation_generator,
                        validation_steps=len(validation_generator),
                        epochs=EPOCHS,
                        verbose=VERBOSITY,
                        callbacks=[PlotLossesKeras(),
                                ModelCheckpoint(MODEL_FILE,
                                            monitor='val_accuracy',
                                            verbose=VERBOSITY,
                                            save_best_only=True,
                                            mode='max'),
                                CSVLogger(TRAINING_LOGS_FILE,
                                        append=False,
                            separator=',')])
```

```
# Three callbacks
# 1) Plot accuracy and loss graph live
# 2) Save the best model every epoch
# 3) Log the accuracy and loss for each epoch

model.save(MODEL_FILE)

# Load the weights from training
model.load_weights(MODEL_FILE)

# Plot training, validation accuracy and loss vs epoch
epochs = [i for i in range(1, len(history.history['loss'])+1)]
plt.style.use('dark_background')

plt.plot(epochs, history.history['loss'], color='blue', label="training_loss")
plt.plot(epochs, history.history['val_loss'], color='red', label="validation_loss")
plt.legend(loc='best')
plt.title('training')
plt.xlabel('epoch')
plt.savefig(TRAINING_PLOT_FILE, bbox_inches='tight')
plt.show()

plt.plot(epochs, history.history['accuracy'], color='blue', label="training_accuracy")
plt.plot(epochs, history.history['val_accuracy'], color='red',label="validation_accuracy")
plt.legend(loc='best')
plt.title('validation')
plt.xlabel('epoch')
plt.savefig(VALIDATION_PLOT_FILE, bbox_inches='tight')
plt.show()

# Plot ROC validation curse from validation data
roc_validation_generator =
ImageDataGenerator(rescale=1./255).flow_from_directory(validation_path,
target_size=(IMAGE_SIZE,IMAGE_SIZE), batch_size=BATCH_SIZE,
                                                        class_mode='binary',
                                                        shuffle=False)
predictions = model.predict_generator(roc_validation_generator,
steps=len(roc_validation_generator), verbose=VERBOSITY)
false_positive_rate, true_positive_rate, threshold =
roc_curve(roc_validation_generator.classes, predictions)
area_under_curve = auc(false_positive_rate, true_positive_rate)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(false_positive_rate, true_positive_rate, label='AUC =
{:.3f}'.format(area_under_curve))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.savefig(ROC_PLOT_FILE, bbox_inches='tight')
plt.show()

y_act = roc_validation_generator.classes
y_pred = [1 if i > 0.5 else 0 for i in predictions]
class_labels = list(roc_validation_generator.class_indices.keys())
cm=confusion_matrix(y_true=y_act, y_pred=y_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
print(classification_report(y_act, y_pred,target_names=class_labels))
```

5.2 Model 2 - InceptionV3 with Simple RNN

```
%matplotlib inline

!pip install livelossplot

# Imports

import numpy as np
import pandas as pd
from glob import glob
import os
import shutil
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix,
classification_report, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import seaborn as sn
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout, Flatten, Dense, GlobalMaxPooling2D, Input,
Concatenate
from keras.models import Model, Sequential
from keras.callbacks import CSVLogger, ModelCheckpoint
from keras.optimizers import Adam
from keras.applications.inception_v3 import InceptionV3
from livelossplot import PlotLossesKeras

# Output files
TRAINING_LOGS_FILE = "training_logs.csv"
MODEL_FILE = "histopathologic_cancer_detector.h5"
TRAINING_PLOT_FILE = "training.png"
VALIDATION_PLOT_FILE = "validation.png"
ROC_PLOT_FILE = "roc.png"
MODEL_LAYERS_SIMPLIFIED = "model_plot_simplified.png"
MODEL_LAYERS = "model_plot.png"
INPUT_DIR = '../input/histopathologic-cancer-detection/'
training_path = '../training'
validation_path = '../validation'

# Hyperparams
SAMPLE_COUNT = 85000
TRAINING_RATIO = 0.9
IMAGE_SIZE = 96
EPOCHS = 16
BATCH_SIZE = 216
VERBOSITY = 1
TESTING_BATCH_SIZE = 5000

# Data setup
training_dir = INPUT_DIR + 'train/'
```

```
data_frame = pd.DataFrame({'path': glob(os.path.join(training_dir,'*.tif'))})
data_frame['id'] = data_frame.path.map(lambda x: x.split('/')[4].split('.')[0])
labels = pd.read_csv(INPUT_DIR + 'train_labels.csv')
data_frame = data_frame.merge(labels, on = 'id')
negatives = data_frame[data_frame.label == 0].sample(SAMPLE_COUNT)
positives = data_frame[data_frame.label == 1].sample(SAMPLE_COUNT)
data_frame = pd.concat([negatives, positives]).reset_index()
data_frame = data_frame[['path', 'id', 'label']]

# Create folder for training and validation and sub folders for splitting the data
# as binary classification
for folder in [training_path, validation_path]:
    for subfolder in ['0', '1']:
        path = os.path.join(folder, subfolder)
        os.makedirs(path, exist_ok=True)

training, validation = train_test_split(data_frame, train_size=TRAINING_RATIO,
stratify=data_frame['label'])

data_frame.set_index('id', inplace=True)

for images_and_path in [(training, training_path), (validation, validation_path)]:
    images = images_and_path[0]
    path = images_and_path[1]
    for image in images['id'].values:
        file_name = image + '.tif'
        label = str(data_frame.loc[image,'label'])
        destination = os.path.join(path, label, file_name)
        if not os.path.exists(destination):
            source = os.path.join(INPUT_DIR + 'train', file_name)
            shutil.copyfile(source, destination)

# Data generation
training_generator = ImageDataGenerator(rescale=1./255,
                                        horizontal_flip=True,
                                        vertical_flip=True,
                                        rotation_range=90,
                                        zoom_range=0.2,
                                        width_shift_range=0.1,
                                        height_shift_range=0.1,
                                        shear_range=0.05,
                                        channel_shift_range=0.1).flow_from_directory(training_path,
                                                                target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                                               batch_size=BATCH_SIZE,
                                                                               class_mode='binary')
validation=ImageDataGenerator(rescale=1./255).flow_from_directory(validation_path,
                                                                target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                                    batch_size=BATCH_SIZE,
                                                                    class_mode='binary')
```

```
input_shape = (IMAGE_SIZE, IMAGE_SIZE, 3)
model = Sequential()
model.add(Input(input_shape))
model.add(InceptionV3(include_top=False, input_shape=input_shape))
model.add(GlobalMaxPooling2D())
model.add(Reshape((1, -1)))
model.add(SimpleRNN(128))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# compile the model
model.compile(optimizer=Adam(learning_rate=0.001,
decay=0.0001,beta_1=0.9,epsilon=1.0),
            loss='binary_crossentropy',
            metrics=['accuracy'])
model.summary()

# Plot the model layer and save it as a file
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
SVG(model_to_dot(model).create(prog='dot', format='svg'))

from keras.utils.vis_utils import plot_model
plot_model(model, to_file=MODEL_LAYERS_SIMPLIFIED, show_shapes=True,
show_layer_names=True, expand_nested=False)
plot_model(model, to_file=MODEL_LAYERS, show_shapes=True,
show_layer_names=True, expand_nested=True)

# fit the model on data generator
history = model.fit(training_generator,
        steps_per_epoch=len(training_generator),
        validation_data=validation_generator,
        validation_steps=len(validation_generator),
        epochs=EPOCHS,
        verbose=VERBOSITY,
        callbacks=[PlotLossesKeras(),
                ModelCheckpoint(MODEL_FILE,
                        monitor='val_accuracy',
                        verbose=VERBOSITY,
                        save_best_only=True,
                        mode='max'),
                CSVLogger(TRAINING_LOGS_FILE,
                        append=False,
                        separator=',')])

model.load_weights(MODEL_FILE)

# Training plots
epochs = [i for i in range(1, len(history.history['loss'])+1)]
plt.style.use('dark_background')
plt.plot(epochs, history.history['loss'], color='blue', label="training_loss")
plt.plot(epochs, history.history['val_loss'], color='red', label="validation_loss")
plt.legend(loc='best')
plt.title('training')
plt.xlabel('epoch')
plt.savefig(TRAINING_PLOT_FILE, bbox_inches='tight')
plt.show()
plt.plot(epochs, history.history['accuracy'], color='blue', label="training_accuracy")
plt.plot(epochs, history.history['val_accuracy'], color='red',label="validation_accuracy")
plt.legend(loc='best')
plt.title('validation')
plt.xlabel('epoch')
plt.savefig(VALIDATION_PLOT_FILE, bbox_inches='tight')
plt.show()
roc_validation_generator =
ImageDataGenerator(rescale=1./255).flow_from_directory(validation_path,

target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                        batch_size=BATCH_SIZE,
                                        class_mode='binary',
                                        shuffle=False)
predictions = model.predict(roc_validation_generator,
steps=len(roc_validation_generator), verbose=VERBOSITY)
false_positive_rate, true_positive_rate, threshold =
roc_curve(roc_validation_generator.classes, predictions)
area_under_curve = auc(false_positive_rate, true_positive_rate)

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(false_positive_rate, true_positive_rate, label='AUC =
{:.3f}'.format(area_under_curve))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.savefig(ROC_PLOT_FILE, bbox_inches='tight')
plt.show()

y_act = roc_validation_generator.classes
y_pred = [1 if i > 0.5 else 0 for i in predictions]
class_labels = list(roc_validation_generator.class_indices.keys())
cm=confusion_matrix(y_true=y_act, y_pred=y_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
print(classification_report(y_act, y_pred,target_names=class_labels))
```

5.3 Model 3 - InceptionV3 with LSTM

```
%matplotlib inline

!pip install livelossplot

# Imports
import numpy as np
import pandas as pd
from glob import glob
import os
import shutil
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix,
classification_report, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import seaborn as sn
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout, Flatten, Dense, GlobalMaxPooling2D, Input,
Concatenate, LSTM, SimpleRNN, Conv2D, MaxPooling2D, Reshape
from keras.models import Model, Sequential
from keras.callbacks import CSVLogger, ReduceLROnPlateau, ModelCheckpoint,
EarlyStopping
from keras.optimizers import Adam
from keras.applications.inception_v3 import InceptionV3
from livelossplot import PlotLossesKeras

# Output files
TRAINING_LOGS_FILE = "training_logs.csv"
MODEL_FILE = "histopathologic_cancer_detector.h5"
TRAINING_PLOT_FILE = "training.png"
VALIDATION_PLOT_FILE = "validation.png"
ROC_PLOT_FILE = "roc.png"
MODEL_LAYERS_SIMPLIFIED = "model_plot_simplified.png"
MODEL_LAYERS = "model_plot.png"
INPUT_DIR = '../input/histopathologic-cancer-detection/'
training_path = '../training'
validation_path = '../validation'

# Hyperparams
SAMPLE_COUNT = 85000
TRAINING_RATIO = 0.9
IMAGE_SIZE = 96
EPOCHS = 16
BATCH_SIZE = 216
VERBOSITY = 1
TESTING_BATCH_SIZE = 5000

# Data setup
training_dir = INPUT_DIR + 'train/'
```

```
data_frame = pd.DataFrame({'path': glob(os.path.join(training_dir,'*.tif'))})
data_frame['id'] = data_frame.path.map(lambda x: x.split('/')[4].split('.')[0])
labels = pd.read_csv(INPUT_DIR + 'train_labels.csv')
data_frame = data_frame.merge(labels, on = 'id')
negatives = data_frame[data_frame.label == 0].sample(SAMPLE_COUNT)
positives = data_frame[data_frame.label == 1].sample(SAMPLE_COUNT)
data_frame = pd.concat([negatives, positives]).reset_index()
data_frame = data_frame[['path', 'id', 'label']]

# Create folder for training and validation and sub folders for splitting the data
# as binary classification
for folder in [training_path, validation_path]:
    for subfolder in ['0', '1']:
        path = os.path.join(folder, subfolder)
        os.makedirs(path, exist_ok=True)

training, validation = train_test_split(data_frame, train_size=TRAINING_RATIO,
stratify=data_frame['label'])
data_frame.set_index('id', inplace=True)
for images_and_path in [(training, training_path), (validation, validation_path)]:
    images = images_and_path[0]
    path = images_and_path[1]
    for image in images['id'].values:
        file_name = image + '.tif'
        label = str(data_frame.loc[image,'label'])
        destination = os.path.join(path, label, file_name)
        if not os.path.exists(destination):
            source = os.path.join(INPUT_DIR + 'train', file_name)
            shutil.copyfile(source, destination)

# Data generation
training_generator = ImageDataGenerator(rescale=1./255,
                                        horizontal_flip=True,
                                        vertical_flip=True,
                                        rotation_range=90,
                                        zoom_range=0.2,
                                        width_shift_range=0.1,
                                        height_shift_range=0.1,
                                        shear_range=0.05,
                                        channel_shift_range=0.1).flow_from_directory(training_path,

target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                                            batch_size=BATCH_SIZE,
                                                                            class_mode='binary')
validation_generator =
ImageDataGenerator(rescale=1./255).flow_from_directory(validation_path,

target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                                  batch_size=BATCH_SIZE,
                                                                  class_mode='binary')
```

```
input_shape = (IMAGE_SIZE, IMAGE_SIZE, 3)
model = Sequential()
model.add(Input(input_shape))
model.add(InceptionV3(include_top=False, input_shape=input_shape))
model.add(GlobalMaxPooling2D())
model.add(Reshape((1, -1)))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# compile the model
model.compile(optimizer=Adam(learning_rate=0.001,
decay=0.0001,beta_1=0.9,epsilon=1.0),
        loss='binary_crossentropy',
        metrics=['accuracy'])
model.summary()

# Plot the model layer and save it as a file
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
SVG(model_to_dot(model).create(prog='dot', format='svg'))

from keras.utils.vis_utils import plot_model
plot_model(model, to_file=MODEL_LAYERS_SIMPLIFIED, show_shapes=True,
show_layer_names=True, expand_nested=False)
plot_model(model, to_file=MODEL_LAYERS, show_shapes=True,
show_layer_names=True, expand_nested=True)

# fit the model on data generator
history = model.fit(training_generator,
      steps_per_epoch=len(training_generator),
      validation_data=validation_generator,
      validation_steps=len(validation_generator),
      epochs=EPOCHS,
      verbose=VERBOSITY,
      callbacks=[PlotLossesKeras(),
            ModelCheckpoint(MODEL_FILE,
                  monitor='val_accuracy',
                  verbose=VERBOSITY,
                  save_best_only=True,
                  mode='max'),
            CSVLogger(TRAINING_LOGS_FILE,
                  append=False,
                  separator=',')])

# Training plots
epochs = [i for i in range(1, len(history.history['loss'])+1)]
plt.style.use('dark_background')
plt.plot(epochs, history.history['loss'], color='blue', label="training_loss")
plt.plot(epochs, history.history['val_loss'], color='red', label="validation_loss")
plt.legend(loc='best')
plt.title('training')
plt.xlabel('epoch')
plt.savefig(TRAINING_PLOT_FILE, bbox_inches='tight')
plt.show()

plt.plot(epochs, history.history['accuracy'], color='blue', label="training_accuracy")
plt.plot(epochs, history.history['val_accuracy'], color='red',label="validation_accuracy")
plt.legend(loc='best')
plt.title('validation')
plt.xlabel('epoch')
plt.savefig(VALIDATION_PLOT_FILE, bbox_inches='tight')
plt.show()

model.load_weights(MODEL_FILE)

roc_validation_generator =
ImageDataGenerator(rescale=1./255).flow_from_directory(validation_path,

target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                        batch_size=BATCH_SIZE,
                                                        class_mode='binary',
                                                        shuffle=False)
predictions = model.predict(roc_validation_generator,
steps=len(roc_validation_generator), verbose=VERBOSITY)

roc_validation_generator =
ImageDataGenerator(rescale=1./255).flow_from_directory(validation_path,

target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                        batch_size=BATCH_SIZE,
                                                        class_mode='binary',
                                                        shuffle=False)
# roc_valid_gen = custom_generator(roc_validation_generator)
predictions = model.predict_generator(roc_validation_generator,
steps=len(roc_validation_generator), verbose=VERBOSITY)
false_positive_rate, true_positive_rate, threshold =
roc_curve(roc_validation_generator.classes, predictions)
area_under_curve = auc(false_positive_rate, true_positive_rate)

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(false_positive_rate, true_positive_rate, label='AUC =
{:.3f}'.format(area_under_curve))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.savefig(ROC_PLOT_FILE, bbox_inches='tight')
plt.show()

y_act = roc_validation_generator.classes
y_pred = [1 if i > 0.5 else 0 for i in predictions]
class_labels = list(roc_validation_generator.class_indices.keys())
cm=confusion_matrix(y_true=y_act, y_pred=y_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
print(classification_report(y_act, y_pred,target_names=class_labels))
```

CHAPTER 6
OUTPUT SNAPSHOTS

6.1 Model 1 - Inception V3 architecture

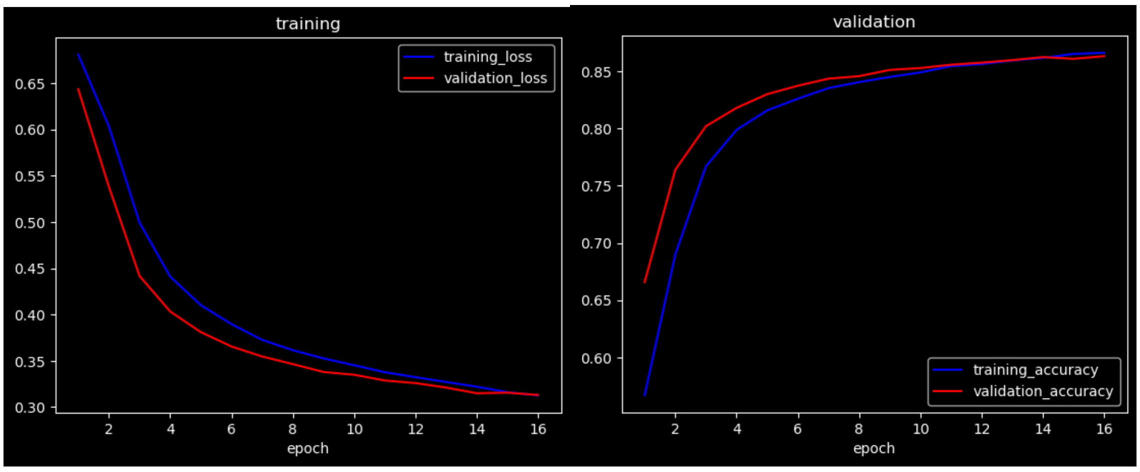
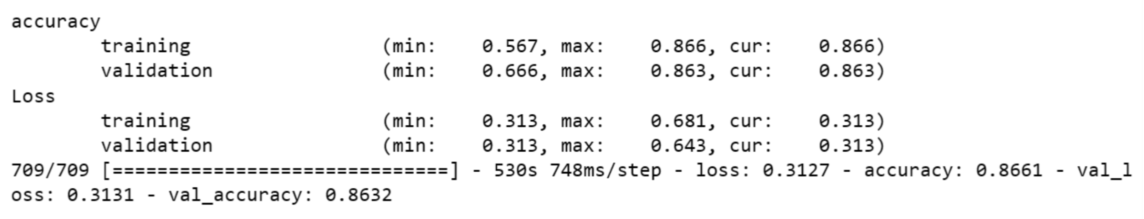


Fig 6.1 The diagrams show training and validation loss(left) and accuracy(right) of model 1 which runs with a maximum epoch of 16.



```
accuracy
        training                 (min:    0.567, max:    0.866, cur:    0.866)
        validation               (min:    0.666, max:    0.863, cur:    0.863)
Loss
        training                 (min:    0.313, max:    0.681, cur:    0.313)
        validation               (min:    0.313, max:    0.643, cur:    0.313)
709/709 [==============================] - 530s 748ms/step - loss: 0.3127 - accuracy: 0.8661 - val_l
oss: 0.3131 - val_accuracy: 0.8632
```

Fig 6.2 The Figure gives the minimum and maximum accuracy and loss for the training and validation set for the model 1.

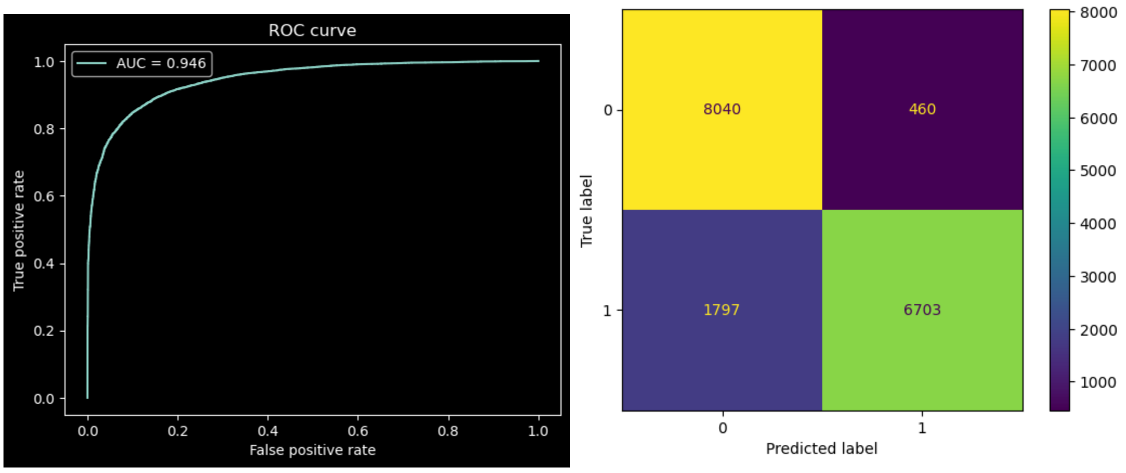
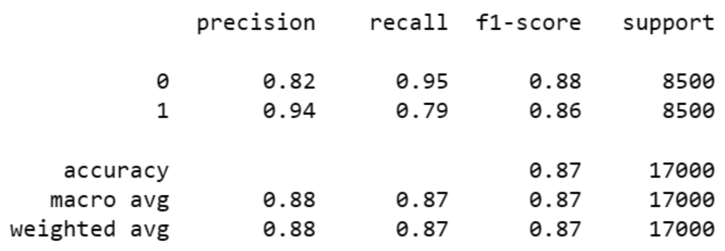


Fig 6.3 The diagram shows the ROC curve of model 1 which has the Area Under the Curve (AUC) value (left) and Confusion matrix for the model(right).



```
              precision    recall  f1-score   support

           0       0.82      0.95      0.88      8500
           1       0.94      0.79      0.86      8500

    accuracy                           0.87     17000
   macro avg       0.88      0.87      0.87     17000
weighted avg       0.88      0.87      0.87     17000
```

Fig 6.4 The figure shows the classification report for Model 1

6.2 Model 2 - InceptionV3 with Simple RNN

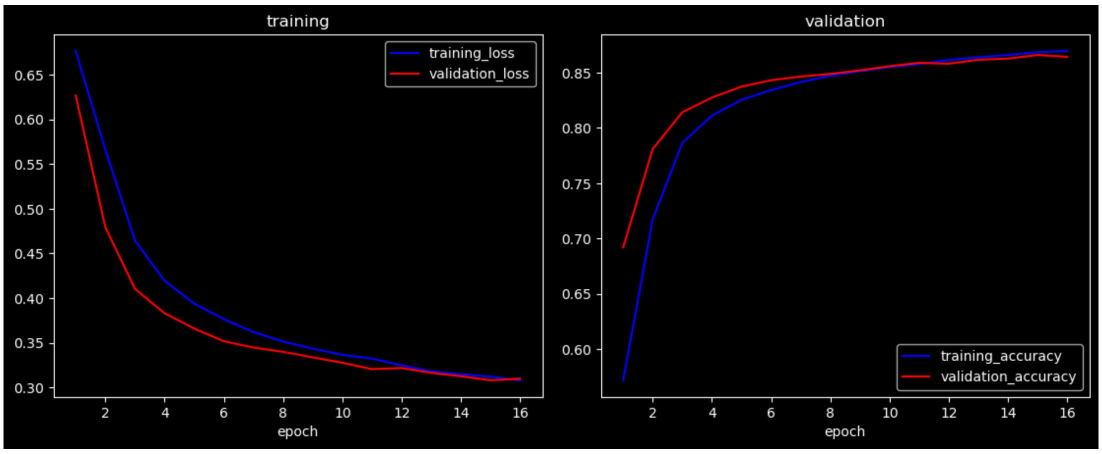


Fig 6.5 The diagrams show training and validation loss(left) and accuracy(right) of model 2 which runs with a maximum epoch of 16.

```
accuracy
        training                 (min:    0.572, max:    0.870, cur:    0.870)
        validation               (min:    0.692, max:    0.866, cur:    0.864)
Loss
        training                 (min:    0.308, max:    0.676, cur:    0.308)
        validation               (min:    0.308, max:    0.627, cur:    0.310)

Epoch 16: val_accuracy did not improve from 0.86594
709/709 [==============================] - 564s 796ms/step - loss: 0.3081 - accuracy: 0.8696 - val_lo
ss: 0.3099 - val_accuracy: 0.8643
```

Fig 6.6 The Figure gives the minimum and maximum accuracy and loss for the training and validation set for the model 2.

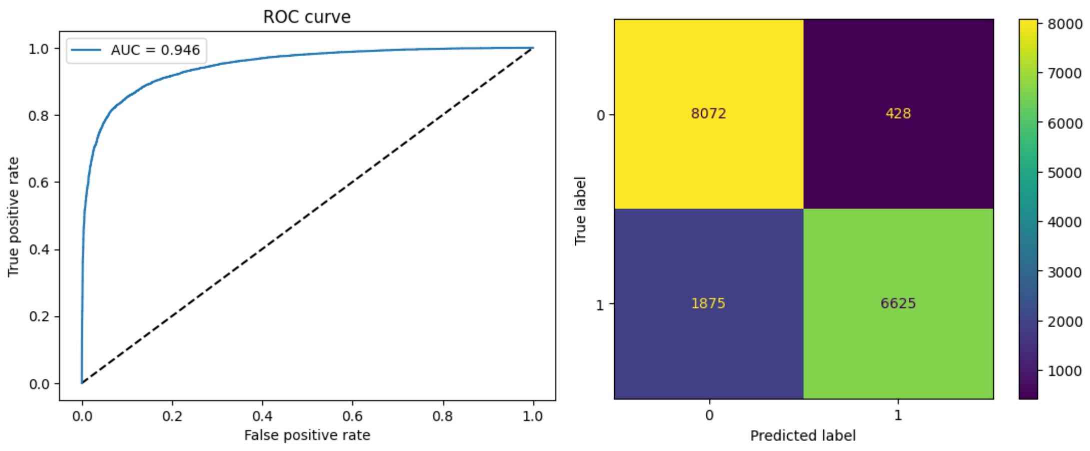
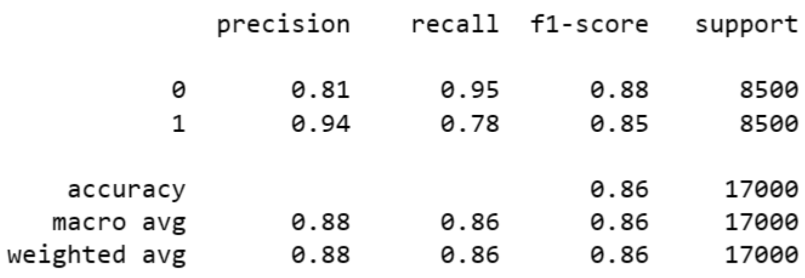


Fig 6.7 The diagram shows the ROC curve of model 1 which has the Area Under the Curve (AUC) value (left) and Confusion matrix for the model(right).



```
              precision    recall  f1-score   support

           0       0.81      0.95      0.88      8500
           1       0.94      0.78      0.85      8500

    accuracy                           0.86     17000
   macro avg       0.88      0.86      0.86     17000
weighted avg       0.88      0.86      0.86     17000
```

Fig 6.8 This figure shows the classification report for Model 2

6.3 Model 3 - InceptionV3 with LSTM

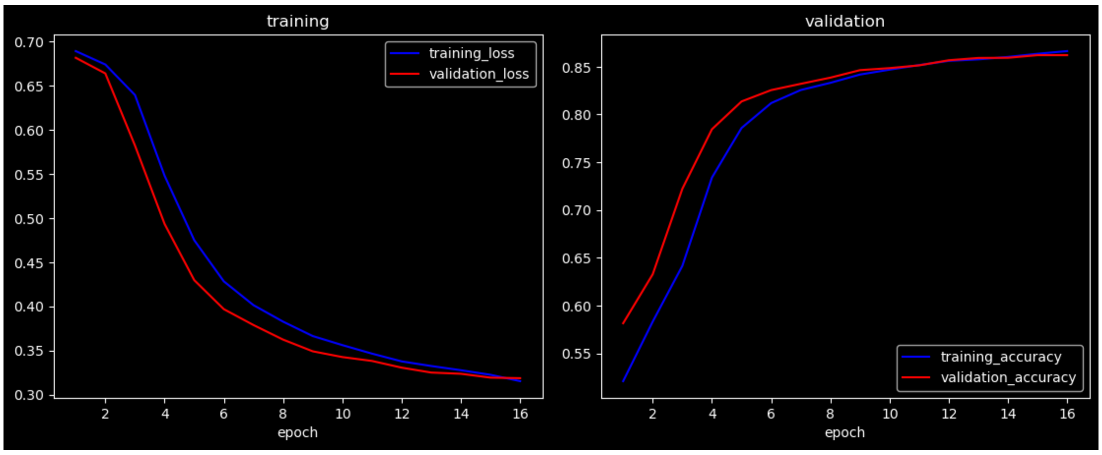


Fig 6.9 The diagrams show training and validation loss(left) and accuracy(right) of model 3 which runs with a maximum epoch of 16.

```
accuracy
        training                  (min:    0.521, max:    0.866, cur:    0.866)
        validation                (min:    0.582, max:    0.862, cur:    0.862)
Loss
        training                  (min:    0.315, max:    0.689, cur:    0.315)
        validation                (min:    0.319, max:    0.682, cur:    0.319)

Epoch 16: val_accuracy did not improve from 0.86212
709/709 [==============================] - 528s 744ms/step - loss: 0.3153 - accuracy: 0.8664 - val_lo
ss: 0.3186 - val_accuracy: 0.8621
```

Fig 6.10 The Figure gives the minimum and maximum accuracy and loss for the training and validation set for the model 3.

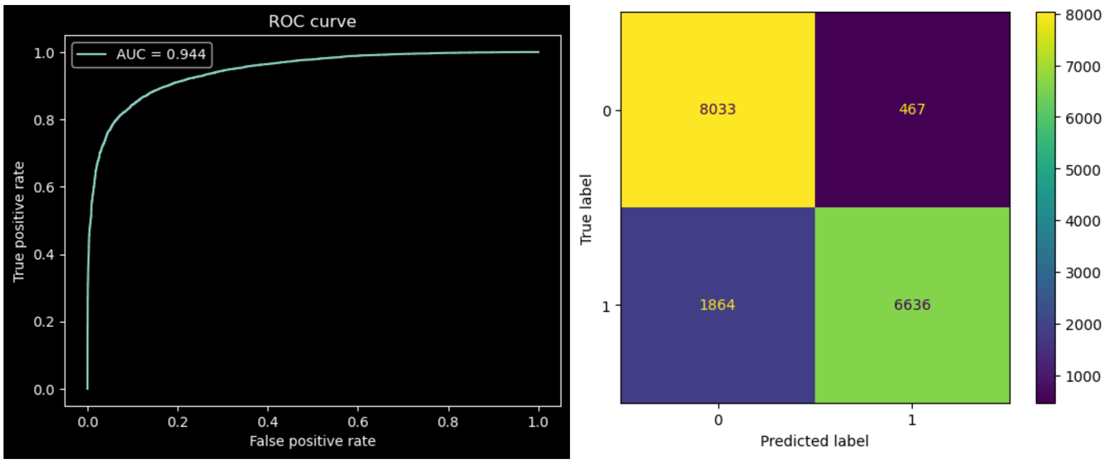
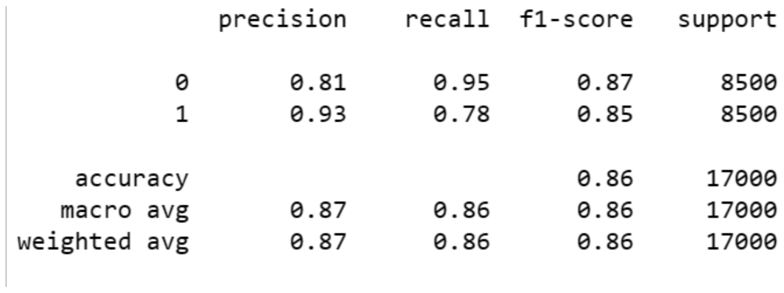


Fig 6.11 The diagram shows the ROC curve of model 1 which has the Area Under the Curve (AUC) value (left) and Confusion matrix for the model(right).



```
              precision    recall  f1-score   support

           0       0.81      0.95      0.87      8500
           1       0.93      0.78      0.85      8500

    accuracy                           0.86     17000
   macro avg       0.87      0.86      0.86     17000
weighted avg       0.87      0.86      0.86     17000
```

Fig 6.12 The figure shows the classification report for Model 3

CHAPTER 7

7.1 CONCLUSION

It is to be noted that each model takes a huge time to train the data. Especially model 3 which took more than ~3.5 hours to train. Model 1 and 2 took approx. ~15 mins for each epoch and hence ~3 hours for both models (16 epochs) to train the data. Model 2 has the highest accuracy and the lowest loss in both training and validation set compared to the other models thus model 2 can be used to classify adenocarcinoma and adenoma. Table 5.1 shows the overall accuracy and loss of each model that is used.

	MODEL 1 (InceptionV3)	**MODEL 2 (InceptionV3 with Simple RNN)**	**MODEL 3 (InceptionV3 with LSTM)**
Training accuracy	0.866	0.870	0.866
Validation accuracy	0.863	0.864	0.862
Training loss	0.313	0.308	0.315
Validation loss	0.313	0.310	0.319

Table 7.1 Accuracy and loss of all three models obtained