Northumbria University
NEWCASTLE

FACULTY OF COMPUTER AND INFORMATION SCIENCES

KV7006 MACHINE LEARNING

CLASSIFICATION OF BREAST CANCER IMAGES

By:
Name: Shlok Gola
Student ID: W21026488
MSc. Data Science

Word Count: 2170
(excluding content page, front page, references, appendix, tables)

Contents

1. Introduction

Deep Learning has gotten a tremendous amount of attention, notably in medical image analysis. The use of deep learning for medical image analysis, such as CNN, has increased dramatically during the last ten years. Deep learning is widely utilised in healthcare to find patterns, classify and segment cancers, among other things. The classification of breast cancer is well known issue that has drawn the attention of numerous healthcare researchers because breast cancer is the most frequent and severe type of cancer globally, coming in second place after lung cancer in a report on cancer mortality (Stewart and Wild, 2014, Alghodhaifi et al., 2019). Invasive Ductal Carcinoma (IDC) is the most frequent kind of breast cancer, accounting for around 70-80% of all breast cancer diagnosis (Reza and Ma, 2018). IDC is a type of cancer that starts in a milk duct and spreads to the fibrous or fatty breast tissue outside of it. If IDC is detected early, the patient can be treated and has a good chance of surviving, but undetected cancer can spread to the other parts of the body, also surrounding the breast tissues (Karatayev et al., 2021). The manual diagnosis of breast cancer through histopathology is very tedious and subjective, causing inter-observer discrepancies even among senior pathologists. Computer-aided diagnosis (CAD) solutions based on machine learning platforms are being developed to assist physicians during diagnosis and reduce clinician workload. Deep learning applied to pathology images allows for the discovery of hidden patterns in these images. In the automated segmentation and classification of illnesses on histopathology images, deep learning algorithms produce spectacular results (Alghodhaifi et al., 2019). The convolutional neural networks (CNN), a type of deep learning models which is group of multi-layer neural networks had been widely applied for automatic mitosis detection in breast cancer histopathology images (Malon et al., 2008). In this project two pre-trained CNN models, EfficientNetB7, RestNet50 will used and four CNN models will be designed from scratch to automatically analyse IDC histology images for cancer risk factors, a task that took pathologists hours to complete.

2. Data Exploration

The data has been collected from the provided Kaggle Repository. The data consists of 5547 RGB digital images of breast histology images. The aim is to classify cancerous images (IDC: Invasive Ductal Carcinoma) and non-IDC images. The dataset is in NumPy array form that are extracted from small patches of digital images of breast tissue samples. Patches on breast tissue samples that contains characteristics of invasive ductal carcinoma are labelled as '1' and the rest are labelled as '0'. The RGB values of images given and stored in a file named 'X.npy' and label of images are stored in file named 'Y.npy', shown in figure 2. First dataset will be loaded in jupyter notebook using *load* function as shown in figure 1.

Figure 1. Loading Data

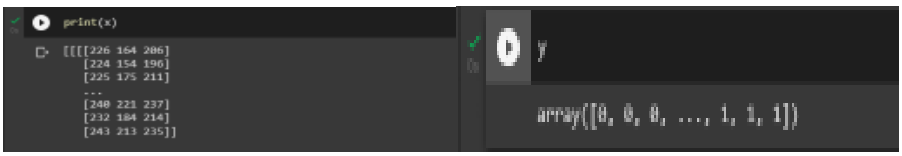


Figure 2. Printing Data in X & Y

After loading data, shape of the files is found out using *shape* class. There are total 5547 images of 50*50 pixel RGB, shown in figure 3.

```
[ ]  print(x.shape)
     print(y.shape)

     (5547, 50, 50, 3)
     (5547,)
```

Figure 3. Shape of X & Y

The number of images that are non-IDC are 2759 and that has IDC are 2788. Therefore, there is no biased-ness in the dataset as both type of images are almost in equal numbers shown in figure 4.

```
[ ] print('No of Negative Images:', len(x[y==0])) # images with label 0 = Non-IDC, Negative Images
    print('No of Positive Images:', len(x[y==1])) # images with label 1 = IDC, Positive Images

    No of Negative Images: 2759
    No of Positive Images: 2788
```

Figure 4. Number of Non-IDC & IDC Images

In figure-5, some examples of non-cancerous and cancerous images are shown.

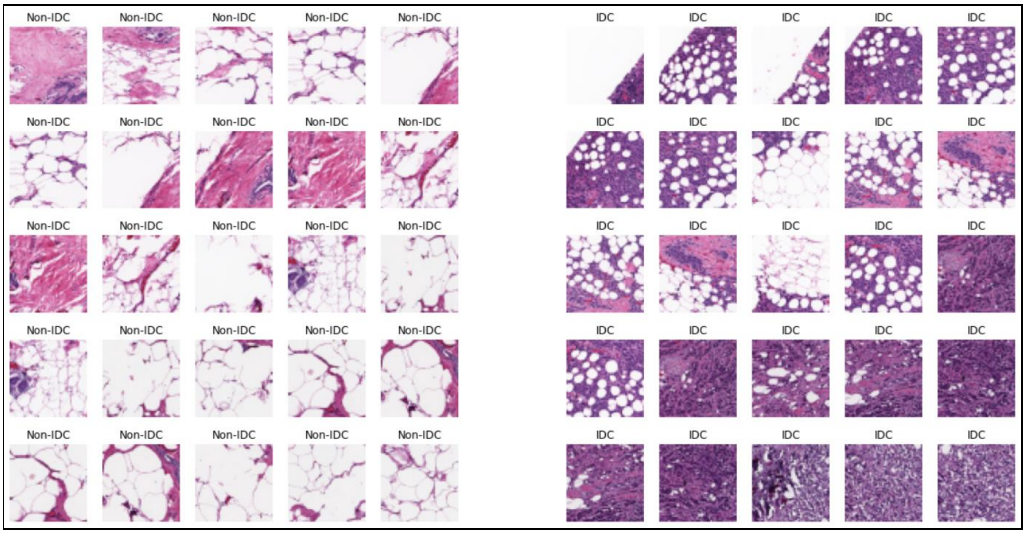


Figure 5. Non-IDC vs IDC Images

3. Data Pre-Processing

Once data exploration is completed, data pre-processing will be done to fed the data to deep learning models. In data exploration it was found out that the number of non-IDC and IDC images are almost similar (non-IDC:2759 and IDC:2788), this shows that both the classes are balanced. In binary classification in medical domain, classification of minority class i.e., patient having cancer is more important than the classification of majority class of cancer free people. In convolutional process, Convo2D layer works better on images as it moves in two directions horizontal and vertical where-as Convo1D layer moves only in vertical direction therefore, inputs of convolutional networks should be in 2D array. The labels of the images are in 1D, so it will be converted to 2D array by performing one-hot encoding using *to_categorical* class, shown in figure 6.

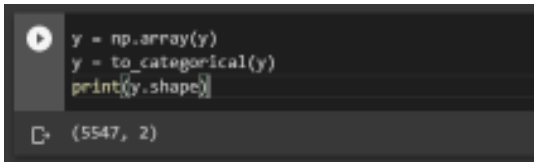


Figure 6. Converting Y to 2D Array

The original pixel of images is 50*50, it will be changed to 32*32 and 64*64 to get different pixel data for same deep learning models and pre-trained models. The size of image is changed using *resize* class, shown below

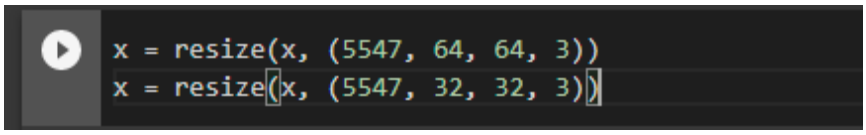


Figure 7. Resizing Image

For evaluating models on different pixel size three different .ipynb file has been made and attached in the folder. Before building the models, the dataset is divided into two subsets: training and testing dataset. The training set trains the model and testing set tests the model. So, that efficiency and effectiveness of the model can be evaluated. The dataset is divided into 80/20 ratio, shown in figure 8.

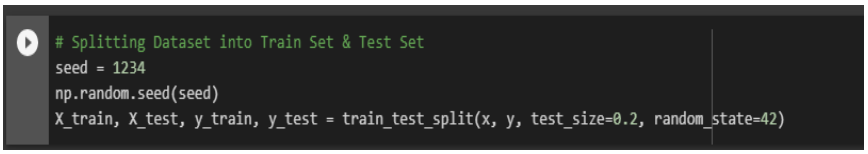


Figure 8. Splitting Dataset into Training and Testing Set

4. Model Building and Fine Tuning

Here, for the classification of cancerous and non-cancerous images two pre-trained models were used and four different convolutional neural network (CNN) models were built. Convolutional neural network applies the local feature detectors to the entire image to quantify the similarities between specific image patches and signature patterns within the training set. After that, max pooling function is used to minimise the dimension of the features space (Lecun et al., 1998). Then the model uses fully connected layer, to do so the CNN output of the model is flattened into single vector (Haq et al., 2022). In fully connected output layer different activation function and optimizers are used. To compare these models transfer learning is applied using two pre-trained models i.e., EfficientNetB7 and RestNet50. Transfer learning is a process of using weights that learn on features of one problem, and are used to leverage them on a new problem. Transfer learning has become prevalent in the field of image analysis, major existing models have been trained on ImageNet dataset for months and the trained weights are then used to work on different types of image classification problems.

Components used in layers for Building CNN Models:

1. Convo2D layer: The convolution layer of 2 dimensions has the filter that works on the image pixel by pixel depending on the kernel size and filter size.
2. Max pooling: CNN operates by extracting the image's characteristics. Max pooling layer is one of the type of down sampling that considers a matrix of particular size and substitute the maximum value from pixels.

3. Dense: A dense layer is one that is totally connected to the layer before it, and it takes all of the neurons information from the prior layer and adds bias to the value. The value is then provided to the activation function, which decides this node's value.
4. Dropout: The dropout layer operates by changing the input values of the neurons to 0. It prevents overfitting by replacing the values with 0 and changing the other values with 1/(1-rate) so that the sum of all inputs remains the same.
5. Filter Size: As the filters pass across a picture, the filter size extracts the features and delivers the value by multiplying an n-dimensional matrix with only the values along the diagonal, which are 1. It extracts those characteristics and returns the value. The n-th filter size is determined by the training data and type of features.
6. Input Shape: The input shape is determined by the pixel size of the images.
7. Activation Functions:
   A. Sigmoid: The sigmoid function returns values ranging from 0 to 1. It employs a function that considers the 1/1+exp equation (-x). As a result, it returns a number close to zero if x is little and close to 1 if x is large.
   B. ReLu: Rectified Linear unit is a function that returns the greatest of two values, the first of which is 0 and the second of which is x. (the input vector). This gives all the positive numbers.
   C. Softmax: The SoftMax function is based on the probability distribution concept and returns an output vector with values between 0 and 1. Furthermore, the sum of these output vectors equals one. This is mostly utilised for classification in the model's final layer.
8. Optimizers:
   A. Adam: Adam is a stochastic gradient descent method that is based on adaptive estimating. It is by far the most popular optimizer. It has a 0.001 learning rate. Because Adam is so powerful, the model tends to overfit, resulting in low validation accuracy.
   B. RMSprop: It maintains a moving (discounted) average of the square of gradients and uses this average to estimate the variance.

Using above mentioned components different CNN models were trained on same dataset with different pixel values. The same training is done with pre-trained models EfficientNetB7 and RestNet50 as well. Details of models trained on are shown below in table 1.

S. No	Model Name	Layers	Activation	Optimizer
1.	Model-2	6	Sigmoid	Adam (0.000005)
2.	Model-4	5	Sigmoid	RMSprop
3.	**Model-5**	**4**	**Softmax**	**Adam**
4.	Model-9	5	Softmax	RMSprop
5.	EfficientNetB7	-	-	Adam (0.0005)
6.	RestNet50	-	-	Adam (0.001)

Table 1. Models Summary

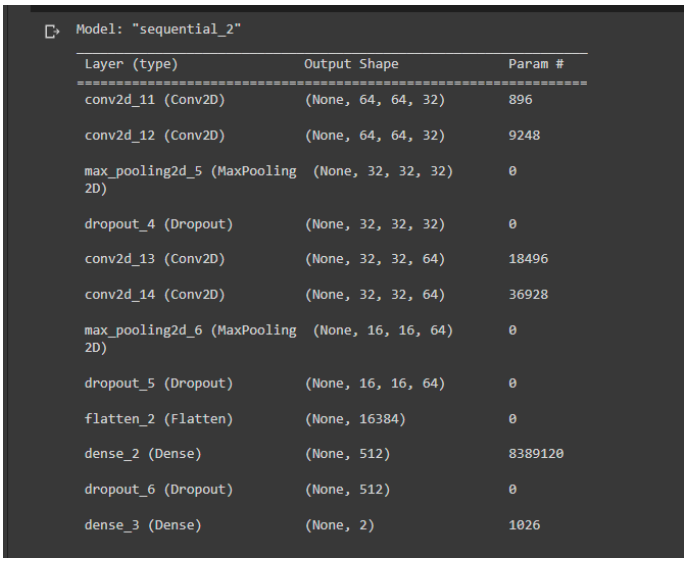
Firstly, model-2 with 6 layers, sigmoid as activation function, Adam optimizer with learning rate 0.000005 and model-4 with 5 layers, sigmoid activation function and RMSprop as optimizer were trained along with EfficientNetB7 and RestNet50. On three different pixel values both model-2 and model-4 performed pretty good attaining accuracy between 73%-75% but the pre-trained models didn’t get accuracy except EfficientNetB7 for 50*50 pixel value and RestNet50 for 64*64 pixel value, shown in table 2.

S. No	Model Name\ Pixel Value	50*50	64*64	32*32
1.	Model-2	75	75	73
2.	Model-4	73	73	74
3.	**Model-5**	**77**	**76**	**75**
4.	Model-9	76	73	74
5.	EfficientNetB7	77	54	51
6.	RestNet50	61	72	63

Table 2. Accuracy Table for Different Pixel Values

Fine Tuning

The model tuning is generally done by overriding the present hypermeters in the models. If the categorization accuracy is not adequate, to achieve satisfactory classification accuracy the hyperparameters are changed in such a way that the model produces better result than the previous one. Here model-2 and model-4 are fine tuned by introducing/reducing new layers and changing activation function and optimizer. For tuning model-2 two layers were reduced, activation function was changed to softmax and learning rate was also removed. This model is named as model-5 which gives better accuracy in all three pixel values. Then in model-4 layers and optimizers are kept same but activation function was changed to softmax. This model is named as model-9 giving better accuracy in 50*50 pixel values but remain same for other two pixel values. Model-5 gives the best accuracy compared to other models created and pre-trained models that were used. Structure of Model-5 is given below:



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 64, 64, 32)        896

conv2d_12 (Conv2D)           (None, 64, 64, 32)        9248

max_pooling2d_5 (MaxPooling  (None, 32, 32, 32)        0
2D)

dropout_4 (Dropout)          (None, 32, 32, 32)        0

conv2d_13 (Conv2D)           (None, 32, 32, 64)        18496

conv2d_14 (Conv2D)           (None, 32, 32, 64)        36928

max_pooling2d_6 (MaxPooling  (None, 16, 16, 64)        0
2D)

dropout_5 (Dropout)          (None, 16, 16, 64)        0

flatten_2 (Flatten)          (None, 16384)             0

dense_2 (Dense)              (None, 512)               8389120

dropout_6 (Dropout)          (None, 512)               0

dense_3 (Dense)              (None, 2)                 1026
```

Figure 9. Structure of Model-5

5. Evaluation and Visualization

Using fine-tuned model-5, it is expected that a good classification can be achieved. After training the model, it is applied to test set to predict IDC images. Therefore, after predicting the confusion matrix was produced that focuses on the strength of the classifiers. The classification report was also executed that shows the precision, recall and F1-score of target class. The confusion matrix and classification report of model-5 for all three pixel values is shown below:

50*50

```
[[439 127]
 [133 411]]
              precision    recall  f1-score   support

           0       0.77      0.78      0.77       566
           1       0.76      0.76      0.76       544

    accuracy                           0.77      1110
   macro avg       0.77      0.77      0.77      1110
weighted avg       0.77      0.77      0.77      1110
```

64*64

```
[[393 173]
 [ 96 448]]
              precision    recall  f1-score   support

           0       0.80      0.69      0.75       566
           1       0.72      0.82      0.77       544

    accuracy                           0.76      1110
   macro avg       0.76      0.76      0.76      1110
weighted avg       0.76      0.76      0.76      1110
```

32*32

```
[[373 193]
 [ 87 457]]
              precision    recall  f1-score   support

           0       0.81      0.66      0.73       566
           1       0.70      0.84      0.77       544

    accuracy                           0.75      1110
   macro avg       0.76      0.75      0.75      1110
weighted avg       0.76      0.75      0.75      1110
```

Figure 10. Confusion Matrix and Classification Report of Model-5

For better understanding of model performance, history of model training is plotted. There are two graphs: one is the accuracy on training and test set over training epochs and second is loss on training and test set over training epochs. Here graphs of model-5 for three pixel values are shown in figure 11, the graphs of other models is shown in Appendix-2.

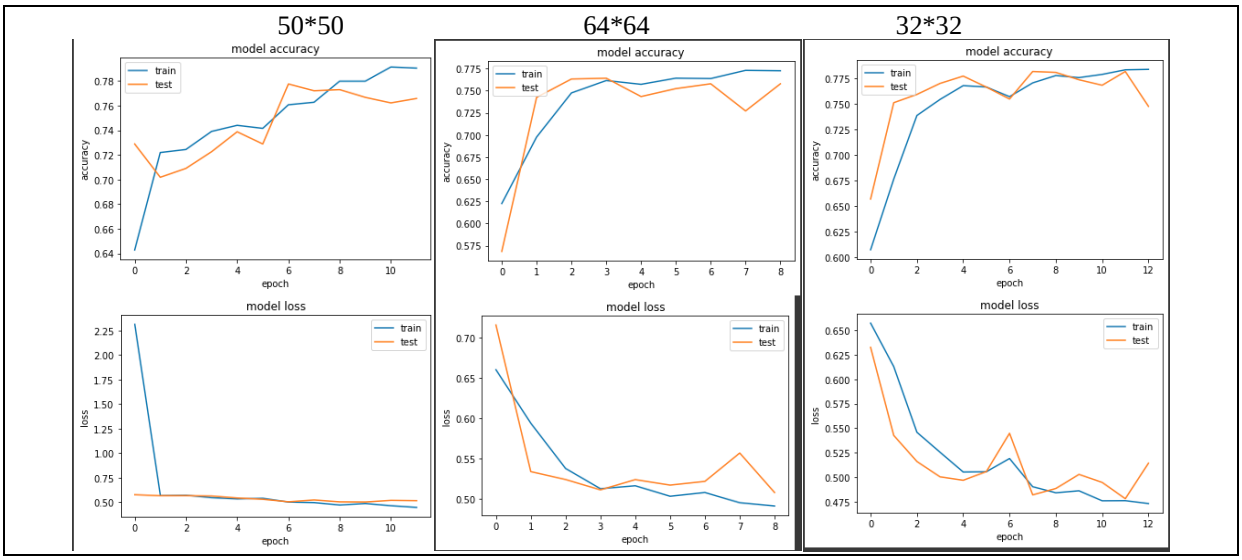


Figure 11. Model Accuracy & Model Loss Graph of Model-5

From the plot of accuracy, it can be seen that for 50*50 and 64*64 pixel value model could be trained little more as trend of accuracy is till rising but not in 32*32 pixel value. From the plot of loss, the performance of model on all three pixels values is comparable on both training and test dataset. If the parallel plots start to depart consistently, the model can be overfit and training should be stopped at earlier epochs. Furthermore, ROC (receiver operating characteristics) curve is evaluated, which shows the performance of model on binary classification. This graph plots two parameters: true positive rate and false positive rate. Classifier that gives curve closer to top-left corner indicates better performance. The curve closer to 45-degree diagonal of ROC space, the model is less accurate on test set. In figure 12, ROC curve of model-5 for all three pixel value is shown and ROC curve of other models is shown in Appendix-3.

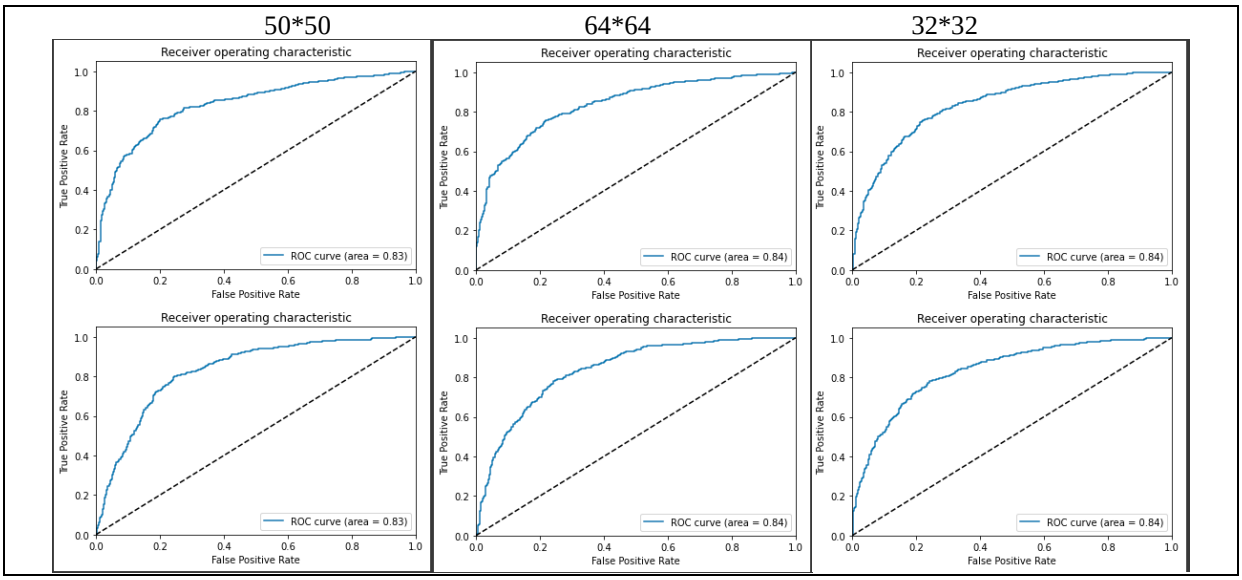


Figure 12. ROC Curve of Model-5

6. Critical Evaluation and Conclusion

In this report, models have been created and pre-trained models were also used to classify IDC or non-IDC images. Using pre-trained models and proposed models, the training has been done through the data and the performance of training has been evaluated. After training the model has been tested with the test set where our proposed CNN model having 4 convolution layers, softmax as activation function and Adam as optimizer has produced the best accuracy for all three pixel values i.e., for 50*50 it gives 77%, for 64*64 it gives 76% and for 32*32 it gives 75% accuracy The classification report and confusion matrix has shown earlier that reflects the performance of the model. For better result, more work can be done on the proposed CNN model to increase the accuracy and efficiency of model. As there were limited resources available and the models have been trained on Google Colab with limited GPU access and memory constraint therefore, there was limited amount of changes that could be applied to the model and pre trained models. The suggestion for higher dimensions of image is also prevalent as if more resources are available.

References

ALGHODHAIFI, H., ALGHODHAIFI, A. & ALGHODHAIFI, M. 2019. Predicting Invasive Ductal Carcinoma in breast histology images using Convolutional Neural Network.

HAQ, A. U., LI, J. P., ZAFARALI, KHAN, I., KHAN, A., UDDIN, M. I., Y.AGBLEY, B. L. & KHAN, R. U. 2022. Stacking approach for accurate Invasive Ductal Carcinoma classification. *Computers and Electrical Engineering,* 100.

KARATAYEV, M., KHALYK, S., ADAI, S., LEE, M.-H. & DEMIRCI, M. F. 2021. Breast Cancer Histopathology Image Classification using CNN. *International Conference on Electronics Computer and Computation.*

LECUN, Y., BOTTOU, L., BENGIO, Y. & HAFFNER, P. 1998. Gradient-Based Learning Applied to Document Recognition. *IEEE***,** 2278-2324.

MALON, C., MILLER, M., BURGER, H. C., COSATTO, E. & GRAF, H. P. Identifying histological elements with convolutional neural networks. Proccedings 5th International Conference on Soft Computing As Transdisciplinary Science and Technology, 2008 New York, NY, USA. 450-456.

REZA, M. S. & MA, J. 2018. Imbalanced Histopathological Breast Cancer Image Classification with Convolutional Neural Network.

STEWART, B. W. & WILD, C. 2014. World cancer report 2014. International Agency for Research on Cancer.

Appendix-1

Libraries used for this Project

```
import tensorflow as tf
import numpy as np
import keras
from keras.models import Sequential
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, Dense, Dropout, MaxPooling2D, Flatten
from keras.layers import BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from skimage.transform import resize
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc
```

Model-2

```
# Model-2
model2 = Sequential()
model2.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(50, 50, 3), activation='relu'))
model2.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model2.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model2.add(Conv2D(filters=256,kernel_size=(3,3), activation = 'relu'))
model2.add(Conv2D(filters=256,kernel_size=(3,3), activation = 'relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Conv2D(filters=512,kernel_size=(3,3), activation = 'relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Flatten())
model2.add(Dense(2, activation='sigmoid'))

model2.compile(optimizer=tf.keras.optimizers.Adam(0.000005),
               loss='binary_crossentropy',
               metrics=['accuracy'])
print(model2.summary())

callback = EarlyStopping(monitor='val_accuracy', patience=5)

seed = 1234
np.random.seed(seed)
history2 = model2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=30, batch_size=32, callbacks=[callback])
```

Model-4

```
[ ] # Model-4
model4 = Sequential()

model4.add(Conv2D(64,(3,3), activation = 'relu', padding = 'same', input_shape = (50,50,3)))
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Dropout(0.20))
model4.add(Conv2D(128,(3,3), activation = 'relu', padding = 'same'))
model4.add(Dropout(0.25))
model4.add(Conv2D(256,(3,3), activation = 'relu', padding = 'same'))
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Dropout(0.25))
model4.add(Conv2D(512,(3,3), activation = 'relu', padding = 'same'))
model4.add(Dropout(0.25))
model4.add(Conv2D(1024,(3,3), activation = 'relu', padding = 'same'))
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Flatten())
model4.add(Dense(2, activation = 'sigmoid'))


model4.summary()
model4.compile(loss='binary_crossentropy', optimizer ='RMSprop', metrics=['accuracy'])

callback = EarlyStopping(monitor='val_accuracy', patience=5)

seed = 1234
np.random.seed(seed)
history4 = model4.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=30, batch_size=50, callbacks=[callback])
```

Model-5

```
[ ] # Model-5
model5 = Sequential()
model5.add(Conv2D(32,(3,3), activation = 'relu', padding = 'same', input_shape = (50,50,3)))
model5.add(Conv2D(32,(3,3), activation = 'relu', padding = 'same'))
model5.add(MaxPooling2D(pool_size=(2,2)))
model5.add(Dropout(0.25))
model5.add(Conv2D(64,(3,3), activation = 'relu', padding = 'same'))
model5.add(Conv2D(64,(3,3), activation = 'relu', padding = 'same'))
model5.add(MaxPooling2D(pool_size=(2,2)))
model5.add(Dropout(0.25))
model5.add(Flatten())
model5.add(Dense(512, activation = 'relu'))
model5.add(Dropout(0.5))
model5.add(Dense(2, activation = 'softmax'))


model5.compile(optimizer='adam',
               loss= "binary_crossentropy",
               metrics=['accuracy'])
print(model5.summary())

callback = EarlyStopping(monitor='val_accuracy', patience=5)

seed = 1234
np.random.seed(seed)
history5 = model5.fit(X_train, y_train, epochs=30, validation_data=(X_test, y_test), verbose = 1, callbacks=[callback])
```

Model-9

```
[ ]  # Model-9
     model9 = Sequential()

     model9.add(Conv2D(64,(3,3), activation = 'relu', padding = 'same', input_shape = (50,50,3)))
     model9.add(MaxPooling2D(pool_size=(2,2)))
     model9.add(Dropout(0.20))
     model9.add(Conv2D(128,(3,3), activation = 'relu', padding = 'same'))
     model9.add(Dropout(0.25))
     model9.add(Conv2D(256,(3,3), activation = 'relu', padding = 'same'))
     model9.add(MaxPooling2D(pool_size=(2,2)))
     model9.add(Dropout(0.25))
     model9.add(Conv2D(512,(3,3), activation = 'relu', padding = 'same'))
     model9.add(Dropout(0.25))
     model9.add(Conv2D(1024,(3,3), activation = 'relu', padding = 'same'))
     model9.add(MaxPooling2D(pool_size=(2,2)))
     model9.add(Flatten())
     model9.add(Dense(2, activation = 'softmax'))


     model9.summary()
     model9.compile(loss='binary_crossentropy', optimizer ='RMSprop', metrics=['accuracy'])

     callback = EarlyStopping(monitor='val_accuracy', patience=5)

     seed = 1234
     np.random.seed(seed)
     history9 = model9.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=50, callbacks=[callback])
```

EfficientNetB7

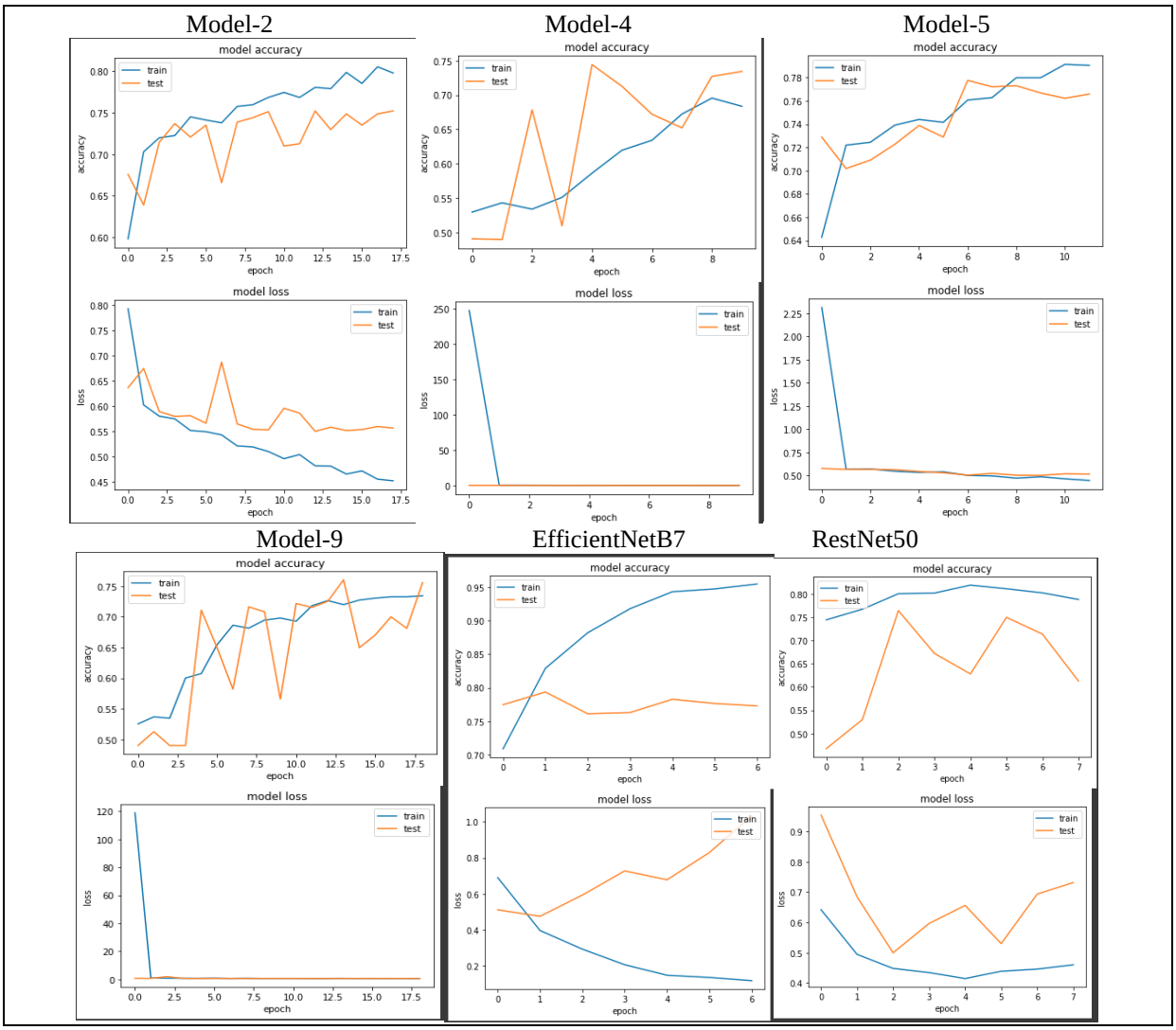
```
EfficientNetB7=tf.keras.applications.efficientnet.EfficientNetB7(input_shape=(50,50,3),
                                                 include_top=False,
                                                 weights='imagenet')
mod1=EfficientNetB7.output
mod1=tf.keras.layers.Flatten()(mod1)
mod1=tf.keras.layers.Dense(units=512, activation=tf.nn.relu)(mod1)
output1=tf.keras.layers.Dense(units=2, activation=tf.nn.softmax)(mod1)
model= tf.keras.models.Model(inputs=EfficientNetB7.inputs,outputs=output1)

model.compile(optimizer=tf.keras.optimizers.Adam(0.0005),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits= False),
              metrics=['accuracy'])
model.summary()

callback = EarlyStopping(monitor='val_accuracy', patience=5)

seed = 1234
np.random.seed(seed)
hist = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32, callbacks=[callback])
```

RestNet50

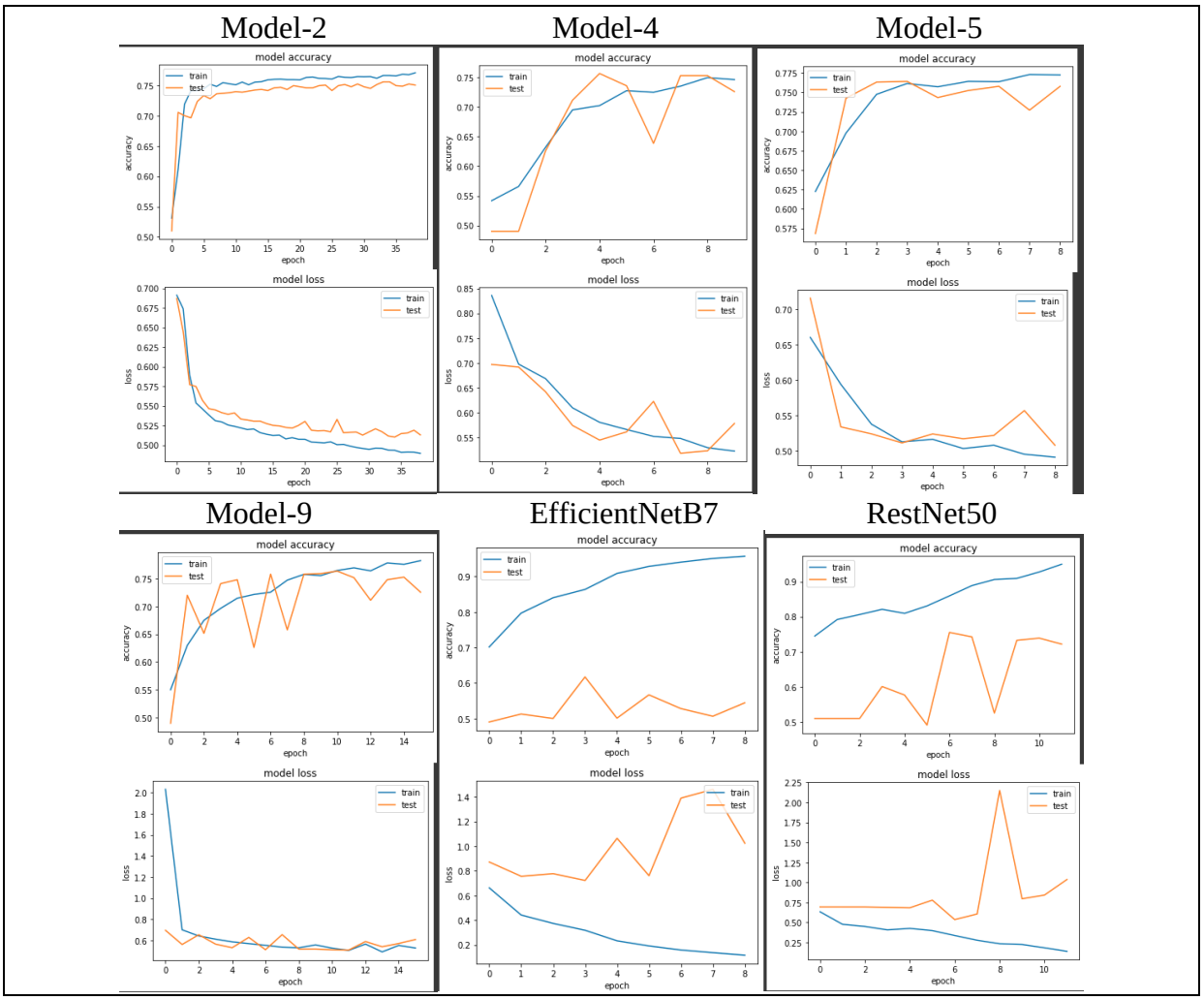
```
RestNet50=tf.keras.applications.resnet.ResNet50(input_shape=(50,50,3),
                                                 include_top=False,
                                                 weights='imagenet')
mod1=RestNet50.output
mod1=tf.keras.layers.Flatten()(mod1)
mod1=tf.keras.layers.Dense(units=256, activation=tf.nn.relu)(mod1)
output1=tf.keras.layers.Dense(units=2, activation=tf.nn.sigmoid)(mod1)
model10= tf.keras.models.Model(inputs=RestNet50.inputs,outputs=output1)

model10.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits= False),
              metrics=['accuracy'])
model10.summary()

callback = EarlyStopping(monitor='val_accuracy', patience=5)

seed = 1234
np.random.seed(seed)
hist10 = model10.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32, callbacks=[callback])
```

Appendix-2

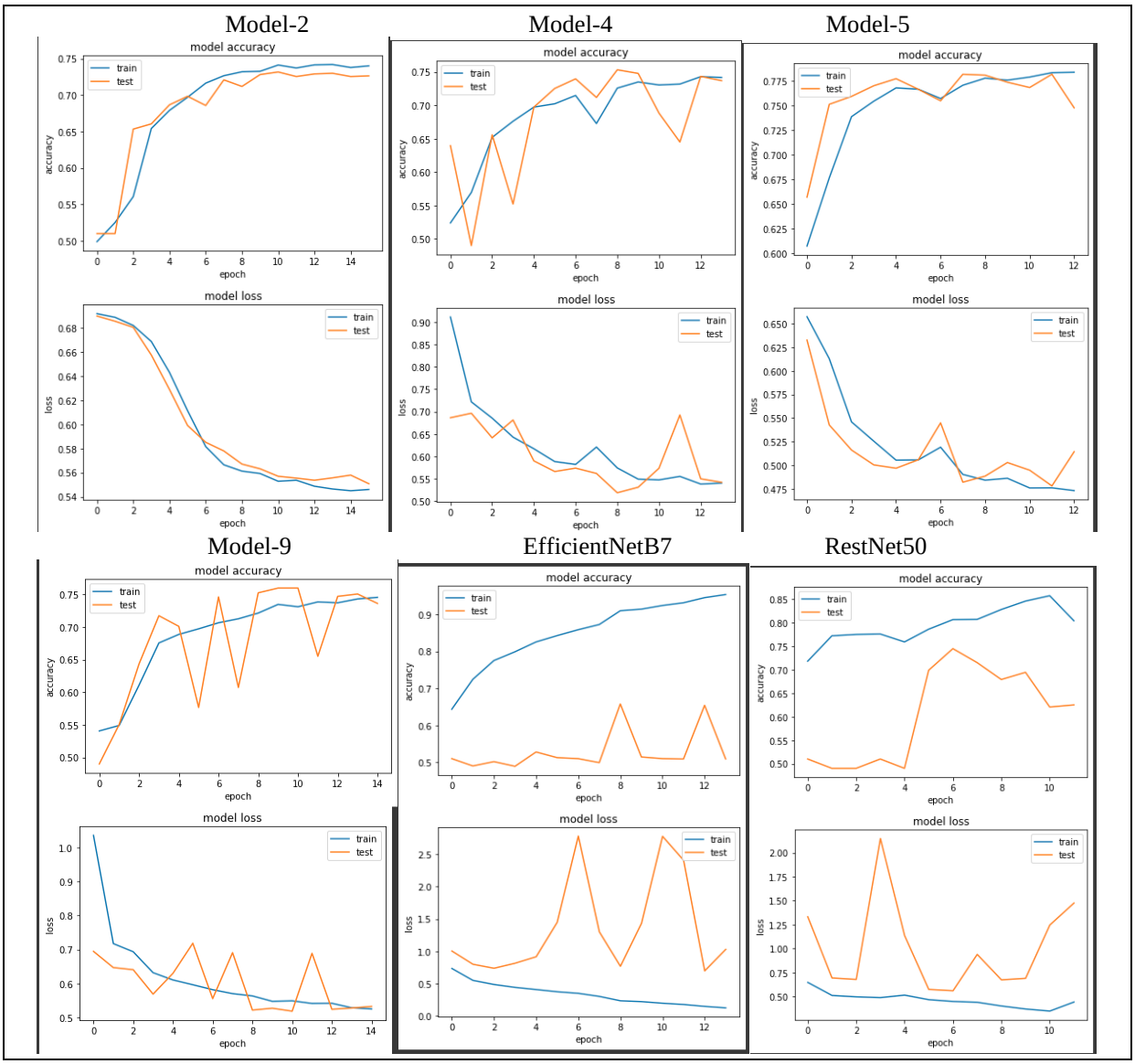
Model Accuracy & Model Loss Graph of all Models for 50*50 Pixel



Model Accuracy & Model Loss Graph of all Models for 64*64 Pixel



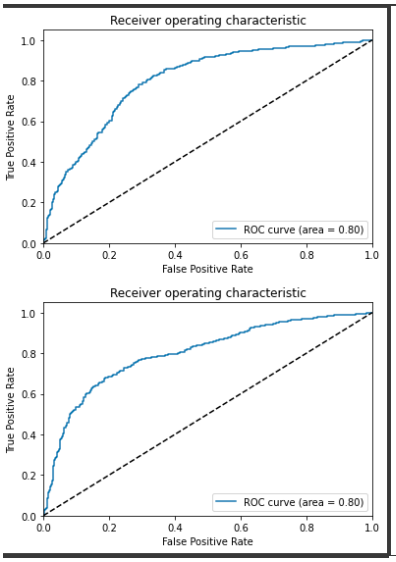
Model Accuracy & Model Loss Graph of all Models for 32*32 Pixel



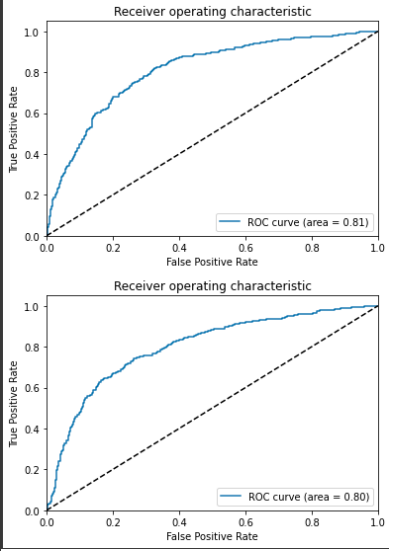
Appendix-3

ROC Curve Graph of all Models for 50*50 Pixel

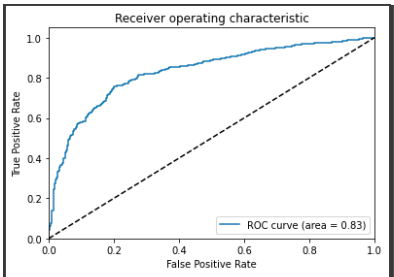
Model-2

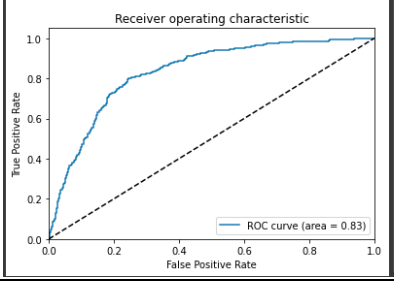


Model-4

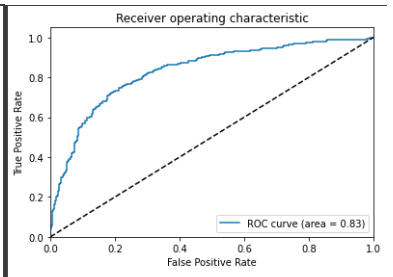


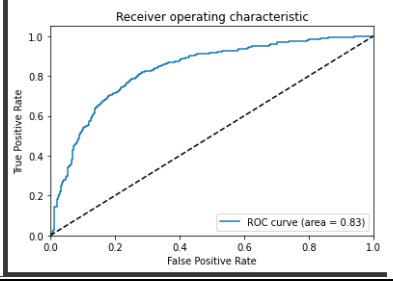
Model-5





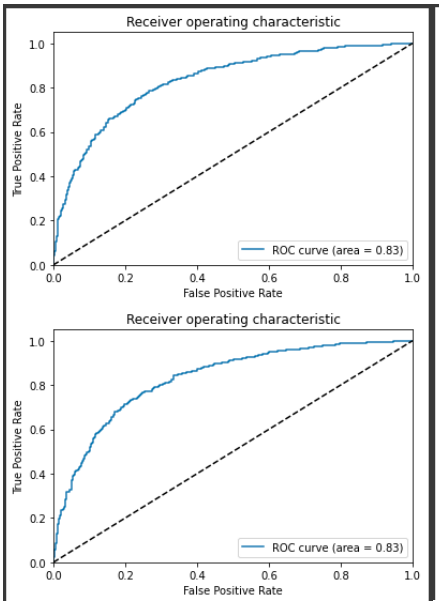
Model-9



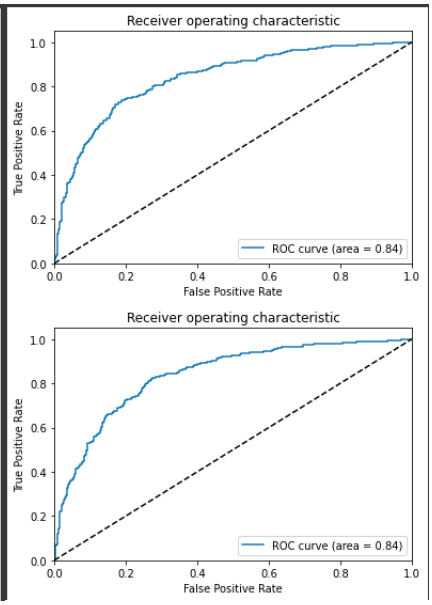


ROC Curve Graph of all Models for 64*64 Pixel

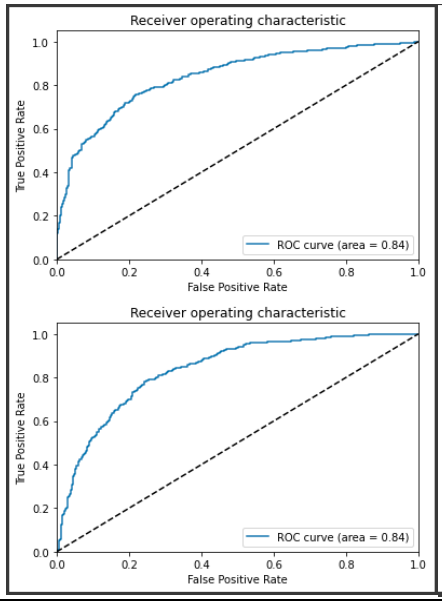
Model-2



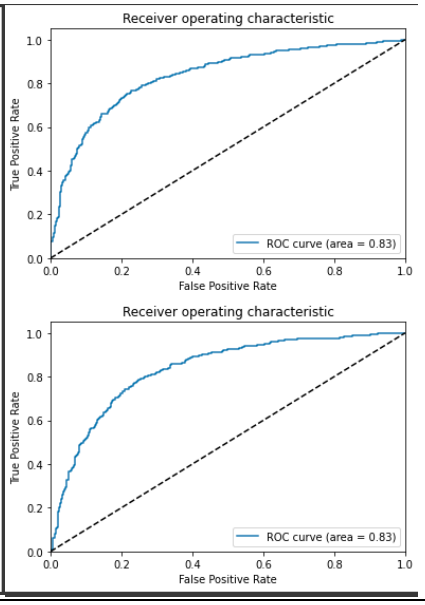
Model-4



Model-5

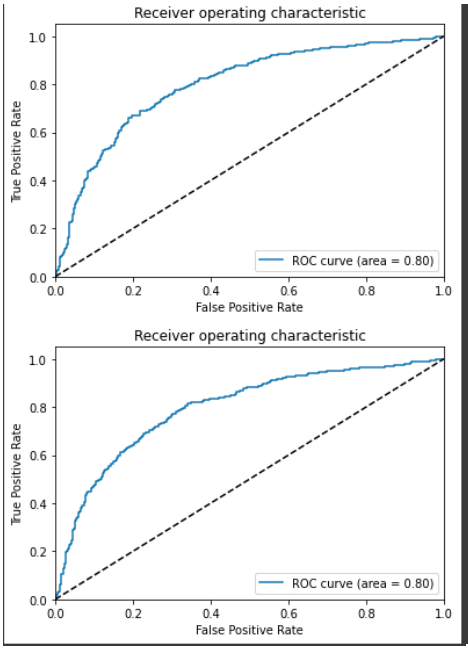


Model-9

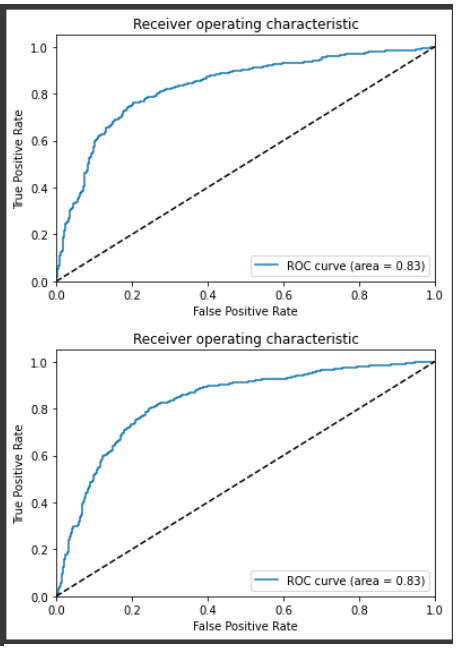


ROC Curve Graph of all Models for 32*32 Pixel

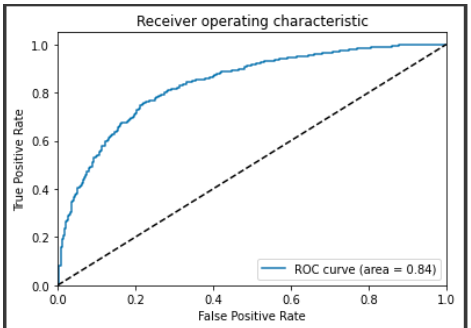
Model-2

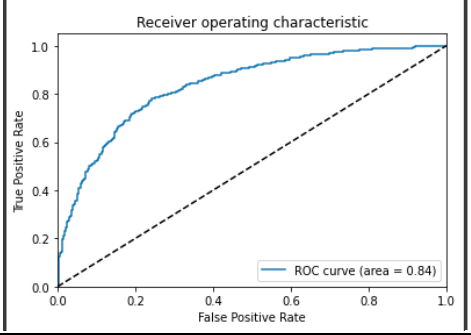


Model-4



Model-5





Model-9