WORD COUND: 1900

SAMSON ALABI ORODELE

W22018376

Classification of Breast Cancer Images ¶

Introduction

Breast cancer is a type of tumour that develops in the breat cells and is one of the prevalent causes of death in women (Wang et al, 2022; Amin et al., 2022; Chaudury). The development of computer vision and machine learning have helped the reliability of classification of histology images, which can be advantageous in the early detection of breast cancer(Vo, 2019). This study will focus on the classification of histology images into Invasive ductal Carcinoma(IDC) or non-IDC. The dataset used for this study contains 5547 breast histology images with a size of 50 x50 x3. Six machine learning models will be explored in this classification task. These contain three variants of Convolutional Neural Networks(CNN), a Multilayer Perceptron(MLP), which is a type of Artificial Neural Network(ANN), a pretrained model(MobileNetV2) and a classical supervised model(Support Vector Machine – SVM.)

The performance of these algorithms will be compared to determine the best performing algorithms suited for detecting IDC from histology images. The outcome of this investigation has a high likelihood of influencing early detection of breast cancer which will invariably translate into early treatment.

Literature Review

Quite a number of studies have been conducted on breast cancer detection using deep learning models such as CNN, ANN and pre-trained models like VGG-16. Samriddha et al. (2023) used an ensemble of three standard CNN models namely GoogleNet, VGG11 and MobileNet V3 which yielded a very impressive accuracy of 96.95%. ConvNet C obtained an accuracy of 88.7% and a sensitivity of 92.6% when Guptal et al(2022) classified a hundred thousand histology images.

Preparing the Tools

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, MaxPooling2D, Conv2D,In
put,Dropout
from keras_tuner import RandomSearch
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV

from tensorflow import image
from keras.optimizers import Adam

from skimage.transform import resize

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

import random
from keras.layers import BatchNormalization
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import StratifiedKFold

# Setting seed For reproducibility
seed = 42
np.random.seed(seed)

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import GlobalAveragePooling2D
from kerastuner.tuners import RandomSearch #For tuning hyperparameters
from tensorflow.keras.preprocessing.image import ImageDataGenerator # For i
mage augmentation
```

Data

The data utilised in this study is a publicly available dataset downloaded from Kaggle.

Importing the Dataset

In [ ]:
```
# This loads the data
x_input = np.load('X.npy')
y_target= np.load('Y.npy')
```

Exploratory Data Analysis

To gain some understanding and insight on the data of the data, some exploratory analysis were performed. This dataset contains 5547 breast histology images with each having a dimension of 50 by 50 by 3(rows, columns, RGB channels). The dataset can be considered to have a balanced class as 2759 negative images and 2788 positive images were present. A glimpse was taken as to how the images looked like by plotting five random positive images and five random negative images. Investigation was also made to determine the intensity range, colour distribution and contrast of the images.

Rationale and Objective of Study: This study is an object recogintion task which will focus on investigating machine learning models that can be suitable to successfully classify IDC cases as either positive or negative based on the presence of some characteristic features in the images.

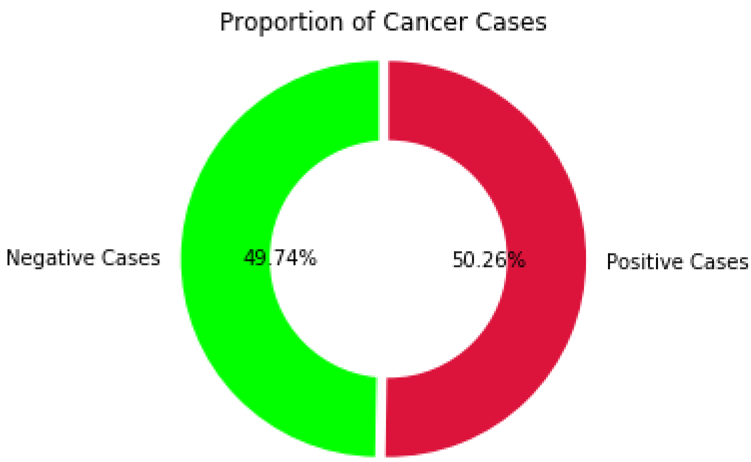
In [ ]:
```
#To determine the dimensions of the data
print('x_input dimension= {}'.format(x_input.shape))
print('y_labels dimension= {}'.format(y_target.shape))
```

```
x_input dimension= (5547, 50, 50, 3)
y_labels dimension= (5547,)
```

In [ ]:
```
#To further gain more insight on the data by checking the balance and stati
stics:
print('Total number of images= {}'.format(len(x_input)))
print('Quantity of Negative Images= {}'.format(np.sum(y_target==0)))
print('Quantity of Positive Images= {}'.format(np.sum(y_target==1)))
print('Percentage of positive images= {:.2f}%'.format(100*np.sum(y_target==
1)/len(x_input)))
print('Percentage of Negative images= {:.2f}%'.format(100*np.sum(y_target==
0)/len(x_input)))
print('Shape of one image(rows, columns, RGB channels)= {}'.format(x_input
[22].shape))
```

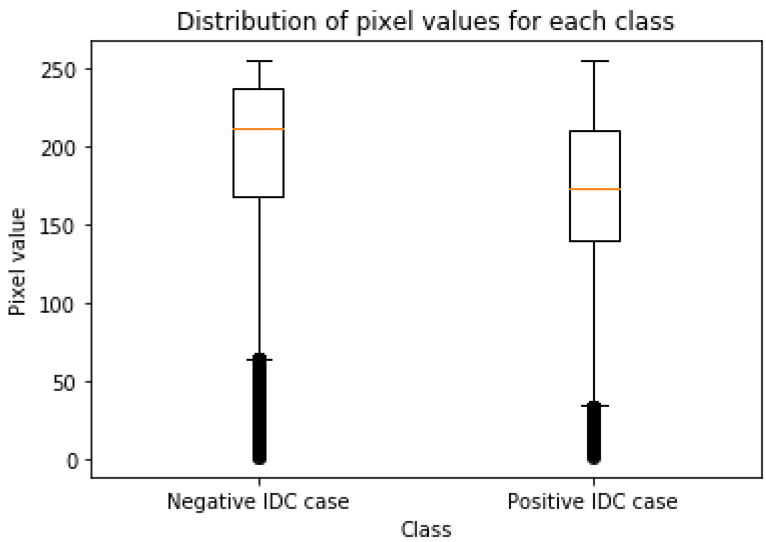
```
Total number of images= 5547
Quantity of Negative Images= 2759
Quantity of Positive Images= 2788
Percentage of positive images= 50.26%
Percentage of Negative images= 49.74%
Shape of one image(rows, columns, RGB channels)= (50, 50, 3)
```

In [ ]:
```
# Checking if the data is balcanced by checking the proportion.

#To enumerate the number of cases
num_negative = np.sum(y_target == 0)
num_positive = np.sum(y_target == 1)

# Creating the pie chart for visualization
labels = ['Negative Cases', 'Positive Cases']
pie_sizes = [num_negative, num_positive]
colours = ['lime','crimson']
explode = (0.05, 0)

plt.pie(pie_sizes, labels=labels, colors = colours, explode=explode,
        autopct='%1.2f%%', startangle=90, pctdistance=0.5, labeldistance=1.
1)

#To make it doughnut-like
centre_circle = plt.Circle((0,0),0.6,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

# To  make pie drawn in a circle
plt.axis('equal')
plt.title('Proportion of Cancer Cases')
plt.show();
```

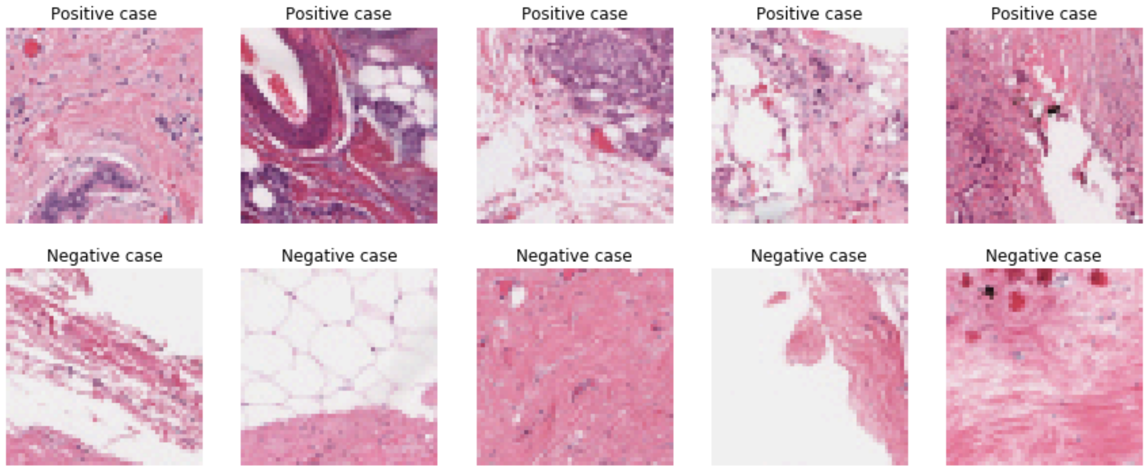


In [ ]:
```
# To get the number of samples in each of the classes
class_qty = np.bincount(y_target)

# Definining colors for each class
colors = ['lime', 'tomato']

# Making a bar chart
plt.bar(['Negative', 'Positive'], class_qty, color=colors)

# labels and title
plt.xlabel('Class')
plt.ylabel('Number of samples')
plt.title('Class distribution');
```

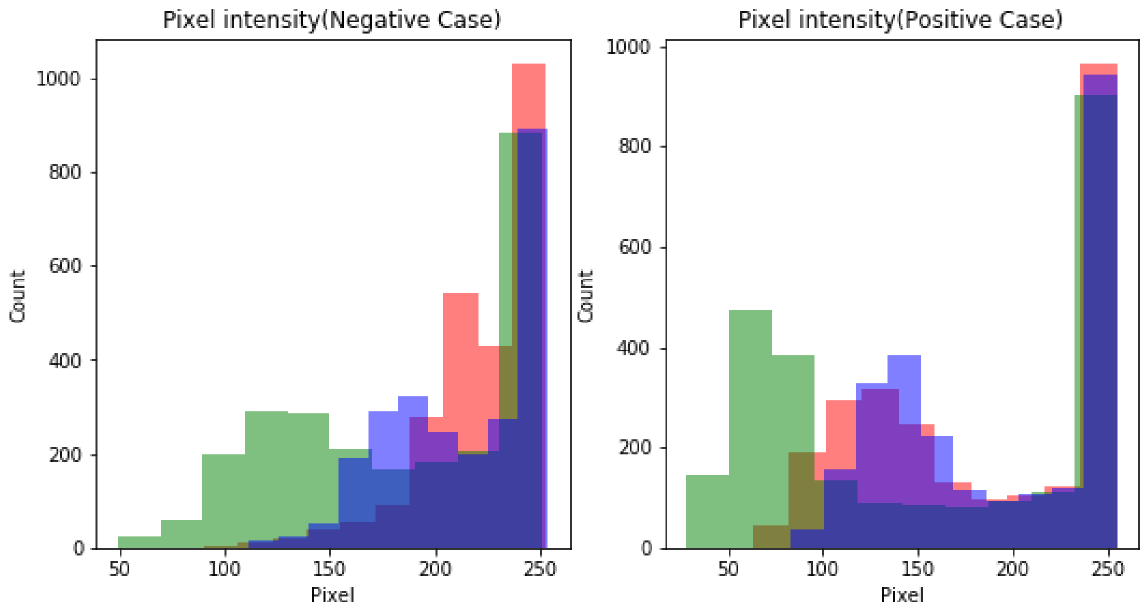
In [ ]:
```
# Boxplot to show the distribution of the pixel values
plt.boxplot([x_input[y_target==0],
             x_input[y_target==1]],
            labels=['Negative IDC case', 'Positive IDC case'])
plt.xlabel('Class')
plt.ylabel('Pixel value')
plt.title('Distribution of pixel values for each class');
```



In [ ]:
```
#To view examples of a positive and a negative case

# To get 5 random positive images
positive_cases = x_input[y_target == 1]
positive_idx = random.sample(range(len(positive_cases)), 5)
positive_imgs = positive_cases[positive_idx]

# To get 5 random negative images
negative_cases = x_input[y_target == 0]
negative_indexes = random.sample(range(len(negative_cases)), 5)
negative_imgs = negative_cases[negative_indexes]


fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(15, 6))
for i, image in enumerate(positive_imgs):
    axes[0, i].imshow(image)
    axes[0, i].set_title('Positive case')
    axes[0, i].axis('off')

for i, image in enumerate(negative_imgs):
    axes[1, i].imshow(image)
    axes[1, i].set_title('Negative case')
    axes[1, i].axis('off');
```



In [ ]:
```
# To determine the distribution of the pixel intensity in each class
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
imgs_0 = x_input[y_target == 0]
imgs_1 = x_input[y_target == 1]

#Pixel intensity for a negative case
axes[0].hist(imgs_0[1,:,:,0].flatten(), bins=10, lw=0, color='r', alpha=0.
5);
axes[0].hist(imgs_0[1,:,:,1].flatten(), bins=10, lw=0, color='g', alpha=0.
5);
axes[0].hist(imgs_0[1,:,:,2].flatten(), bins=10, lw=0, color='b', alpha=0.
5);
axes[0].set_title('Pixel intensity(Negative Case)')
axes[0].set_xlabel('Pixel')
axes[0].set_ylabel('Count')

#Pixel intensity for positive case
axes[1].hist(imgs_1[1,:,:,0].flatten(), bins=10, lw=0, color='r', alpha=0.
5);
axes[1].hist(imgs_1[1,:,:,1].flatten(), bins=10, lw=0, color='g', alpha=0.
5);
axes[1].hist(imgs_1[1,:,:,2].flatten(), bins=10, lw=0, color='b', alpha=0.
5);
axes[1].set_title('Pixel intensity(Positive Case)')
axes[1].set_xlabel('Pixel')
axes[1].set_ylabel('Count');
```



In [ ]:
```
#Statistical Propertis of all the Images

#To determine the statistical properties of RGB channels
mean_r, var_r, std_r = x_input[:,:,:,0].mean(), x_input[:,:,:,0].var(), x_i
nput[:,:,:,0].std()
mean_g, var_g, std_g = x_input[:,:,:,1].mean(), x_input[:,:,:,1].var(), x_i
nput[:,:,:,1].std()
mean_b, var_b, std_b = x_input[:,:,:,2].mean(), x_input[:,:,:,2].var(), x_i
nput[:,:,:,2].std()

# To Print the RGB properties
print("Red channel - mean: ", round(mean_r,2), "  |variance: ", round(var_
r,2), "|standard deviation: ", round(std_r,2))
print("Green channel - mean: ", round(mean_g,2), "|variance: ", round(var_
g,2), "|standard deviation: ", round(std_g,2))
print("Blue channel - mean: ", round(mean_b,2), " |variance: ", round(var_
b,2), "|standard deviation: ", round(std_b,2))
```

```
Red channel - mean:  205.79   |variance:  1317.28 |standard deviation:  3
6.29
Green channel - mean:  161.87 |variance:  2909.54 |standard deviation:  5
3.94
Blue channel - mean:  187.44  |variance:  1496.98 |standard deviation:  3
8.69
```

In [ ]:
```
#For the negative-case images

# To determine the statistical properties of RGB
mean_r, var_r, std_r = negative_cases[:,:,:,0].mean(), negative_cases
[:,:,:,0].var(), negative_cases[:,:,:,0].std()
mean_g, var_g, std_g = negative_cases[:,:,:,1].mean(), negative_cases
[:,:,:,1].var(), negative_cases[:,:,:,1].std()
mean_b, var_b, std_b = negative_cases[:,:,:,2].mean(), negative_cases
[:,:,:,2].var(), negative_cases[:,:,:,2].std()

#Printing the RGB properties
print("Red channel - mean: ", round(mean_r,2), " |variance: ", round(var_r,
2), "|standard deviation:", round(std_r,2))
print("Green channel - mean:", round(mean_g,2), "|variance:", round(var_g,
2), " |standard deviation:", round(std_g,2))
print("Blue channel - mean:", round(mean_b,2), " |variance:", round(var_b,
2), "|standard deviation:", round(std_b,2))
```

```
Red channel - mean:  218.09  |variance:  929.85 |standard deviation: 30.49
Green channel - mean: 177.35 |variance: 2807.3  |standard deviation: 52.98
Blue channel - mean: 197.39  |variance: 1505.39 |standard deviation: 38.8
```

In [ ]:
```
#For the Positive-case images

# To determine the statistical properties of RGB
mean_r, var_r, std_r = positive_cases[:,:,:,0].mean(), positive_cases
[:,:,:,0].var(), positive_cases[:,:,:,0].std()
mean_g, var_g, std_g = positive_cases[:,:,:,1].mean(), positive_cases
[:,:,:,1].var(), positive_cases[:,:,:,1].std()
mean_b, var_b, std_b = positive_cases[:,:,:,2].mean(), positive_cases
[:,:,:,2].var(), positive_cases[:,:,:,2].std()

#Printing the RGB properties
print("Red channel - mean:", round(mean_r,2), "|variance:", round(var_r,2),
"|standard deviation:", round(std_r,2))
print("Green channel - mean:", round(mean_g,2), "|variance:", round(var_g,
2), "|standard deviation:", round(std_g,2))
print("Blue channel - mean:", round(mean_b,2), "|variance:", round(var_b,
2), "|standard deviation:", round(std_b,2))
```

```
Red channel - mean: 193.62 |variance: 1402.65 |standard deviation: 37.45
Green channel - mean: 146.55 |variance: 2538.87 |standard deviation: 50.39
Blue channel - mean: 177.6 |variance: 1293.71 |standard deviation: 35.97
```

Method and Analysis

Six different machine learning models were built in this study. They consist of three neural networks and one classical supervised machine learning model. The neural networks include: Convolutional Neural Network(CNN), Multilayer Perceptron, which is a type of Artificial Neural Network(ANN). Researchers such as Alkhatib et al. (2023), Hafiz et al. (2023), and Chen et al. (2021) have suggested that CNN is extremely powerful in image classification tasks. Jin (2022) has also stated the same for the case of MLP. This findings constitute the reason for the choice of the models in this task.

Convolutional Neural Networks: CNN algortithms are used to handle data with a grid structure, like images. CNN is intended to automatically and adaptively learn a range of features, from low- to high-level structures. Convolution, pooling, and fully linked layers are the three types of layers that make up a standard CNN. The pixel values are retained in a two-dimensional grid since a feature can exist anywhere in a digital image, and an optimizable feature extractor known as the kernel is applied at each image point. Because of this, CNNs are quite effective for processing images. As one layer feeds its output into the next layer, extracted features may gradually and hierarchically become more complex. Through the use of optimisation techniques like backpropagation and gradient descent, among others, training is the act of reducing the discrepancy between outputs and ground truth labels. It requires tweaking variables like kernels. A typical CNN architecture is shown below:

Image Title

CNN1: The first proposed model two convolutional layers. The data was splitted with 75% for training set ans 25% for the test set. The input layer was computed with 32 filters, a kernel size of (3,3) and a rectified linear unit (relu) activation. The maximum value from the area of the image that the Kernel has covered is returned by Max Pooling which was applied after each convolutional layer. By using the Flatten layer, the output of the max pooling was flattened to a 1D array. A dense layer which outputs the classification was set with one unit and a sigmoid activation. The model was compiled with a learning rate of 0.01, thirty epochs and an adam optimizer. Overall, the model contained 1,993,633 total parameters of which all are trainable with no non-trainable parameters. The model was further fitted on the training data with an early stopping with a patience set to 4. The model performed with a a train accuracy of 88.5% and a validation accuracy of 73.9% and a validation loss of 0.6605. This means that the model performed better on seen data than unseen data, which suggests possibility of overfitting. It is also evident from the training curve that the model suffered from overfitting.

CNN II: Just for experimental purpose, perhaps the performance of these models can improve if the images are resized to a bigger size. Some researchers have suggested that there is a possibility that the models will be able to capture more details of the images if the sizes are larger than they already are(Rikiya et al., 2018). Can an accuracy of above 80% be achieved without generating more images for training data or without tuning the hyperparameters? In this trial, the input image was resized to (75,75,3). The data was splitted with 80% for training and 20% for testing. It also contained 2 convolutional layers with a kernel size of (4,4). A (2,2) maxpooling layer was added after each convolutional layer. The output layer contained a unit dense layer and sigmoid function which produces the binary classification or either 0 or 1. The model contained 4,746,817 parameters of which all are trainable with no non-trainable parameter.

CNN III:

In this trial, the data was split into 80% for training and 20% for the testing. This model contains a total of 9 layers which includes two convolutional layers of 32 filters and relu activation. Maxpooling of size (3,3) was added after the convolutional layers. Two dropout layers of rate 0.25 and 0.5 respectively. The Dropout layer is an agent which randomly sets input units to 0 at each step during training time, which helps prevent overfitting (Wongsuphasawat et al., 2018). The dense layer at the end has one unit and a sigmoid activation. The epoch was set to 40 with a loss function of binary cross entropy and Adam optimizer. It contained a total of 43041 parameters, all of which are trainable. The model was further tuned with Keras Tuner with different hyperparameters set for the filters in the input Conv2D layer, the optimizers were also tuned with choices od Adam, rmsprop and sdg. After that, the best model was picked and was fed into the image datagenerator, where the generator generated batches of augmented images on the fly with 85% training data.

MobileNetV2

This Pretrained model contained 5 layers which includes a drop out layer of rate 0.2 which was introduced in order to help prevent overfitting. The data was split such that the training set has 80% of the data. The data was further scaled with a min-max normalization. The output layer was computed with a dense of one unit and a sigmoid activation. The model was compiled with Adam optimizer and loss function of binary cross entropy. The model was trained with the epoch set to 30 and was validated on the test set. Image augmentation was further utilized in the hope of getting a better performance.

Multilayer Perceptron This sequential model contains three dense layers and two dropout layers. The data was split with 75% for training set and 25% for the testing set. The data was reshaped into 2D in order to fit in the Standard Scalar requirement. The model was then instantiated. With the input dimension set to 7500 and a relu activation. The two drop out layers were set to a rate of 0.2 which mean that 20% of the units will be set to zero during each training phase. The model was compiled with a loss function of binary cross entropy and an adam optimizer.

Support Vector Machines

The data was split with 75% for training data and 25% for the test data. Since SVM is a classical supervised learning which can not take in 4D data, the data was reshaped to 2D to suit the requirement. The standard scaling technique was implemented. The model was trained with the kernel set to the Radial Basis Function(rbf). All other parameters were set to default. The model was further trained on the training set. Hyper parameter tuning was further implemented with the use of RandomSearchCV. The Penalty parameter of the error term(C), gamma and the kernel were all tuned. The best hyper parameter was chosen. The classification report was printed, the confusion matrix, ROC and PR curve were plotted.

Data Preprocessing

In [ ]:
```
#Checking for Missing Values in the Numpy Array
missing_x = np.sum(np.isnan(x_input))
print("Sum of missing values in x_input:", missing_x)
```

```
Sum of missing values in x_input: 0
```

In [ ]:
```
#Checking for Missing Values in the Numpy Array
np.sum(np.isnan(y_target))
missing_y = np.sum(np.isnan(x_input))
print("Sum of missing values in y_target:", missing_y)
```

```
Sum of missing values in y_target: 0
```

In [ ]:
```
#To split the data with 75% for training and 25% for testing
X_train, X_test, y_train, y_test = train_test_split(x_input, y_target, test
_size=0.25, random_state=0)
```

In [ ]:
```
#To determine the number of classes in the target variable
num_classes = len(np.unique(y_train))
print("Distinct number of classes values in y_train:", num_classes)
```

```
Distinct number of classes values in y_train: 2
```

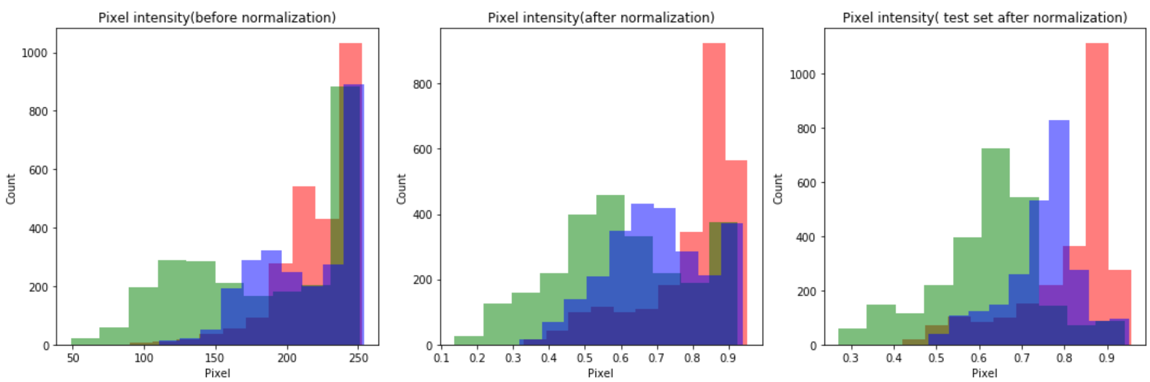
In [ ]:
```
#To take a glimpse at the dimension of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
print('Dimension of Train set:', X_train.shape)
print('Dimension of Test set:', X_test.shape)
print('Dimension of Train targets:' , y_train.shape)
print('Dimension of Test targets:', y_test.shape)
print('Maximum RGB pixel intensity:', X_train.max())
print('Minimum RGB pixel intensity:', X_train.min())
```

```
Dimension of Train set: (4160, 50, 50, 3)
Dimension of Test set: (1387, 50, 50, 3)
Dimension of Train targets: (4160,)
Dimension of Test targets: (1387,)
Maximum RGB pixel intensity: 255
Minimum RGB pixel intensity: 2
```

Data Normalization

In [ ]:
```
# Scale pixel values to range [0, 1]
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [ ]:
```
# To visualize the training intensity before and after normalization

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

x_input
#Pixel intensity of all the data before normalization
axs[0].hist(x_input[1,:,:,0].flatten(), bins=10, lw=0, color='r', alpha=0.
5);
axs[0].hist(x_input[1,:,:,1].flatten(), bins=10, lw=0, color='g', alpha=0.
5);
axs[0].hist(x_input[1,:,:,2].flatten(), bins=10, lw=0, color='b', alpha=0.
5);
axs[0].set_title('Pixel intensity(before normalization)')
axs[0].set_xlabel('Pixel')
axs[0].set_ylabel('Count')

#Pixel intensity of training data after normalization
axs[1].hist(X_train[1,:,:,0].flatten(), bins=10, lw=0, color='r', alpha=0.
5);
axs[1].hist(X_train[1,:,:,1].flatten(), bins=10, lw=0, color='g', alpha=0.
5);
axs[1].hist(X_train[1,:,:,2].flatten(), bins=10, lw=0, color='b', alpha=0.
5);
axs[1].set_title('Pixel intensity(after normalization)')
axs[1].set_xlabel('Pixel')
axs[1].set_ylabel('Count')

#Pixel intensity of test data after normalization
axs[2].hist(X_test[1,:,:,0].flatten(), bins=10, lw=0, color='r', alpha=0.
5);
axs[2].hist(X_test[1,:,:,1].flatten(), bins=10, lw=0, color='g', alpha=0.
5);
axs[2].hist(X_test[1,:,:,2].flatten(), bins=10, lw=0, color='b', alpha=0.
5);
axs[2].set_title('Pixel intensity( test set after normalization)')
axs[2].set_xlabel('Pixel')
axs[2].set_ylabel('Count')

plt.tight_layout()
plt.show()
```



MODEL1: CONVOLUTIONAL NEURAL NETWORK I

In [ ]:
```
#Initializing the model:
model_1= Sequential()
model_1.add(Conv2D(filters = 32,
                 kernel_size=(3,3),
                 input_shape=(50,50,3),
                 activation='relu'))
model_1.add(MaxPooling2D(2,2))
model_1.add(Conv2D(filters = 32, kernel_size=(3,3), activation = 'relu'))
model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Flatten())
model_1.add(Dense(units=512, activation='relu'))
model_1.add(Dense(units=1, activation='sigmoid'))

# To compile the model
lr = 0.01 # learning rate
epochs = 30
model_1.compile(loss = 'binary_crossentropy',
               optimizer = 'adam',  metrics=['accuracy'])

print(model_1.summary())
```

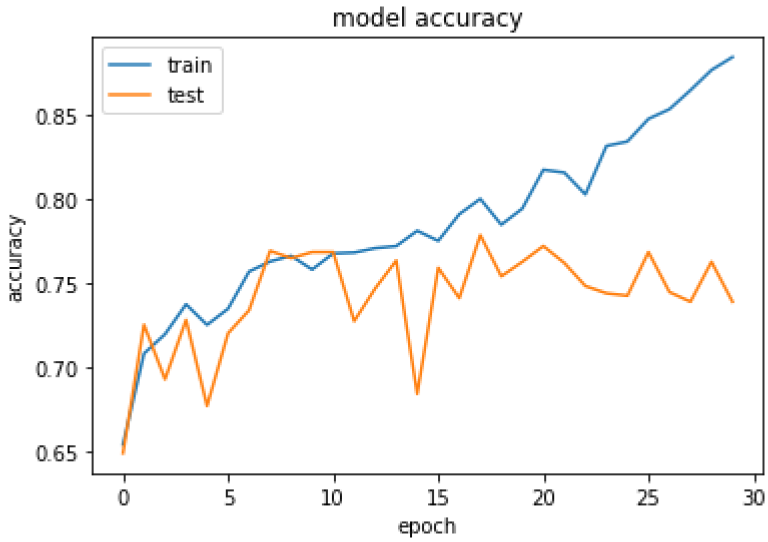
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 48, 48, 32)        896       
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 24, 24, 32)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 22, 22, 32)        9248      
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 11, 11, 32)       0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 3872)              0         
                                                                 
 dense_2 (Dense)             (None, 512)               1982976   
                                                                 
 dense_3 (Dense)             (None, 1)                 513       
                                                                 
=================================================================
Total params: 1,993,633
Trainable params: 1,993,633
Non-trainable params: 0
_________________________________________________________________
None
```

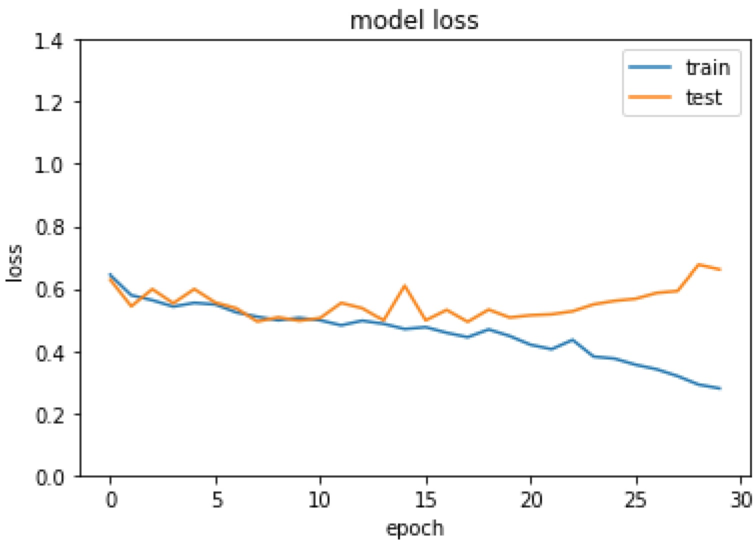
In [ ]:
```
# To set a random seed for reproducibility and to train the model
seed = 42

callback = EarlyStopping(monitor='loss', patience=4)

np.random.seed(seed)

history = model_1.fit(X_train,
                      y_train,
                      validation_data=(X_test, y_test),
                      epochs=epochs,
                      callbacks=[callback],
                      batch_size=64)


print(len(history.history['loss']))
```

```
Epoch 1/30
65/65 [==============================] - 5s 65ms/step - loss: 0.6436 - acc
uracy: 0.6541 - val_loss: 0.6289 - val_accuracy: 0.6489
Epoch 2/30
65/65 [==============================] - 4s 60ms/step - loss: 0.5789 - acc
uracy: 0.7082 - val_loss: 0.5435 - val_accuracy: 0.7253
Epoch 3/30
65/65 [==============================] - 4s 60ms/step - loss: 0.5630 - acc
uracy: 0.7197 - val_loss: 0.5987 - val_accuracy: 0.6929
Epoch 4/30
65/65 [==============================] - 4s 60ms/step - loss: 0.5425 - acc
uracy: 0.7375 - val_loss: 0.5532 - val_accuracy: 0.7282
Epoch 5/30
65/65 [==============================] - 4s 62ms/step - loss: 0.5545 - acc
uracy: 0.7252 - val_loss: 0.5989 - val_accuracy: 0.6770
Epoch 6/30
65/65 [==============================] - 4s 61ms/step - loss: 0.5494 - acc
uracy: 0.7349 - val_loss: 0.5555 - val_accuracy: 0.7203
Epoch 7/30
65/65 [==============================] - 4s 60ms/step - loss: 0.5238 - acc
uracy: 0.7570 - val_loss: 0.5372 - val_accuracy: 0.7340
Epoch 8/30
65/65 [==============================] - 4s 61ms/step - loss: 0.5098 - acc
uracy: 0.7630 - val_loss: 0.4943 - val_accuracy: 0.7693
Epoch 9/30
65/65 [==============================] - 4s 60ms/step - loss: 0.4986 - acc
uracy: 0.7663 - val_loss: 0.5080 - val_accuracy: 0.7650
Epoch 10/30
65/65 [==============================] - 4s 60ms/step - loss: 0.5063 - acc
uracy: 0.7582 - val_loss: 0.4968 - val_accuracy: 0.7686
Epoch 11/30
65/65 [==============================] - 4s 61ms/step - loss: 0.4984 - acc
uracy: 0.7678 - val_loss: 0.5058 - val_accuracy: 0.7686
Epoch 12/30
65/65 [==============================] - 4s 60ms/step - loss: 0.4822 - acc
uracy: 0.7683 - val_loss: 0.5543 - val_accuracy: 0.7275
Epoch 13/30
65/65 [==============================] - 4s 62ms/step - loss: 0.4972 - acc
uracy: 0.7709 - val_loss: 0.5372 - val_accuracy: 0.7469
Epoch 14/30
65/65 [==============================] - 4s 61ms/step - loss: 0.4870 - acc
uracy: 0.7721 - val_loss: 0.4983 - val_accuracy: 0.7635
Epoch 15/30
65/65 [==============================] - 4s 60ms/step - loss: 0.4703 - acc
uracy: 0.7812 - val_loss: 0.6099 - val_accuracy: 0.6842
Epoch 16/30
65/65 [==============================] - 4s 60ms/step - loss: 0.4763 - acc
uracy: 0.7752 - val_loss: 0.4984 - val_accuracy: 0.7592
Epoch 17/30
65/65 [==============================] - 4s 61ms/step - loss: 0.4582 - acc
uracy: 0.7911 - val_loss: 0.5317 - val_accuracy: 0.7412
Epoch 18/30
65/65 [==============================] - 4s 61ms/step - loss: 0.4439 - acc
uracy: 0.8002 - val_loss: 0.4935 - val_accuracy: 0.7787
Epoch 19/30
65/65 [==============================] - 4s 61ms/step - loss: 0.4689 - acc
uracy: 0.7849 - val_loss: 0.5328 - val_accuracy: 0.7541
Epoch 20/30
65/65 [==============================] - 4s 62ms/step - loss: 0.4478 - acc
uracy: 0.7945 - val_loss: 0.5073 - val_accuracy: 0.7628
Epoch 21/30
```

```
65/65 [==============================] - 4s 65ms/step - loss: 0.4196 - acc
uracy: 0.8175 - val_loss: 0.5145 - val_accuracy: 0.7722
Epoch 22/30
65/65 [==============================] - 4s 61ms/step - loss: 0.4053 - acc
uracy: 0.8159 - val_loss: 0.5177 - val_accuracy: 0.7621
Epoch 23/30
65/65 [==============================] - 4s 62ms/step - loss: 0.4352 - acc
uracy: 0.8029 - val_loss: 0.5275 - val_accuracy: 0.7484
Epoch 24/30
65/65 [==============================] - 4s 62ms/step - loss: 0.3815 - acc
uracy: 0.8317 - val_loss: 0.5496 - val_accuracy: 0.7441
Epoch 25/30
65/65 [==============================] - 4s 61ms/step - loss: 0.3750 - acc
uracy: 0.8344 - val_loss: 0.5610 - val_accuracy: 0.7426
Epoch 26/30
65/65 [==============================] - 4s 61ms/step - loss: 0.3553 - acc
uracy: 0.8478 - val_loss: 0.5676 - val_accuracy: 0.7686
Epoch 27/30
65/65 [==============================] - 4s 61ms/step - loss: 0.3408 - acc
uracy: 0.8536 - val_loss: 0.5864 - val_accuracy: 0.7448
Epoch 28/30
65/65 [==============================] - 4s 62ms/step - loss: 0.3188 - acc
uracy: 0.8649 - val_loss: 0.5927 - val_accuracy: 0.7390
Epoch 29/30
65/65 [==============================] - 4s 61ms/step - loss: 0.2914 - acc
uracy: 0.8769 - val_loss: 0.6760 - val_accuracy: 0.7628
Epoch 30/30
65/65 [==============================] - 4s 61ms/step - loss: 0.2791 - acc
uracy: 0.8846 - val_loss: 0.6605 - val_accuracy: 0.7390
30
```

In [ ]:
```
# To summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.ylim([0,1])
# To summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.ylim([0,1.4])
plt.show()
```





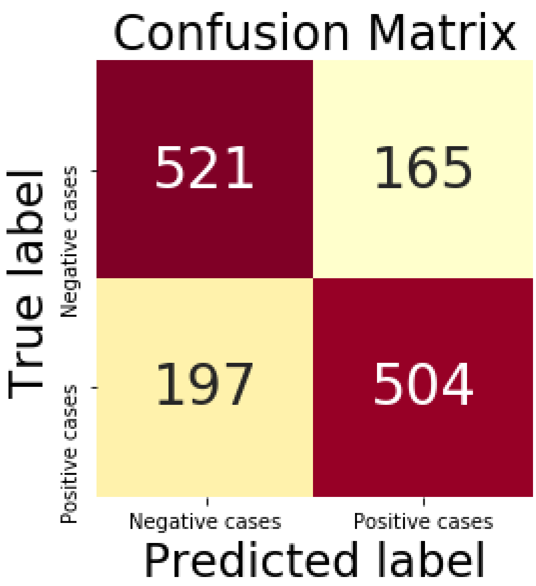
In [ ]:
```
# Evaluate model accuracy
score = model_1.evaluate(X_test, y_test, verbose=0)
print('Model_1: CNNI accuracy:', score[1], '\n')

# Make predictions and convert probabilities to binary predictions
preds = model_1.predict(X_test)
preds1 = np.round(preds)

# Print classification report
print(classification_report(y_test, preds1))
```

```
Model_1: CNNI accuracy: 0.7390050292015076

44/44 [==============================] - 0s 7ms/step
              precision    recall  f1-score   support

           0       0.73      0.76      0.74       686
           1       0.75      0.72      0.74       701

    accuracy                           0.74      1387
   macro avg       0.74      0.74      0.74      1387
weighted avg       0.74      0.74      0.74      1387
```

In [ ]:
```
#To evaluate the confusion matrix
conf_matrix = confusion_matrix(y_test, preds1)
conf_matrix
```

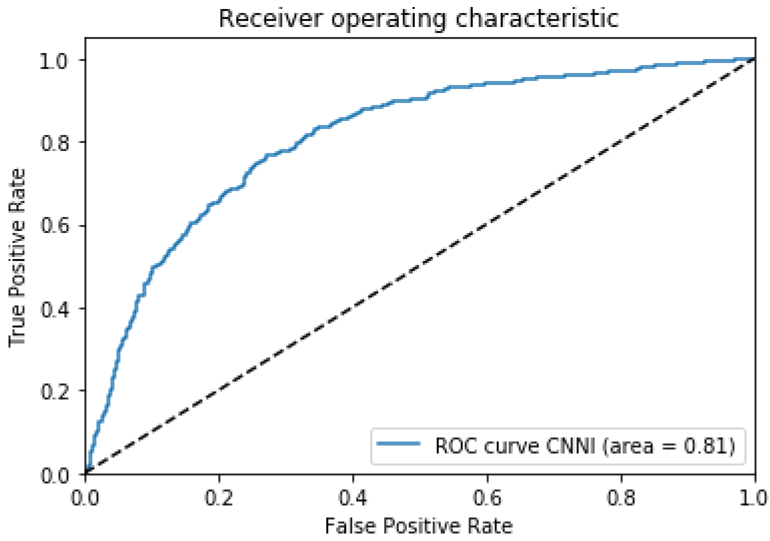
Out[ ]:
```
array([[521, 165],
       [197, 504]], dtype=int64)
```

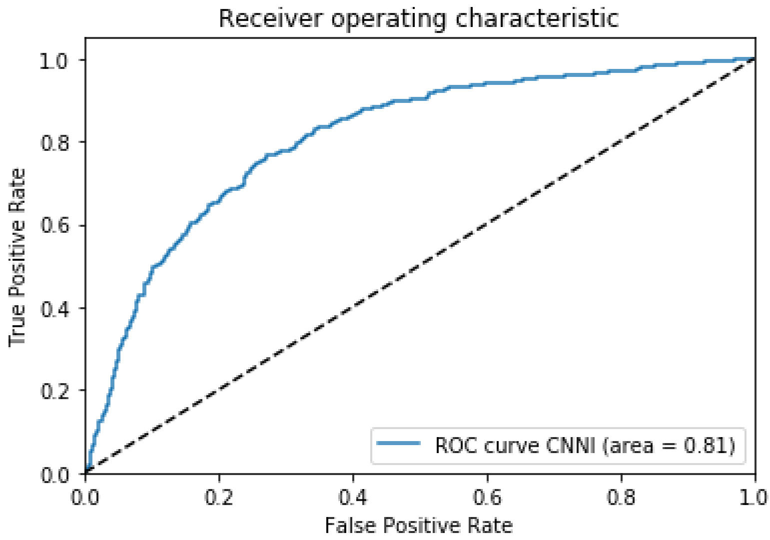
In [ ]:
```
# To plot the confusion matrix
mat = confusion_matrix(y_test, preds1)

sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Negative cases', 'Positive cases'],
            yticklabels=['Negative cases', 'Positive cases'],
            cmap='YlOrRd', annot_kws={"fontsize": 28})

plt.xlabel('Predicted label', fontsize=23)
plt.ylabel('True label', fontsize=23)
plt.title('Confusion Matrix', fontsize=24)

plt.show()
```



In [ ]:
```
from sklearn.metrics import roc_curve, auc

y_score = model_1.predict(X_test) # get the prediction probabilities

fpr1 = dict()
tpr1 = dict()
roc_auc1 = dict()
for i in range(num_classes):
    fpr1[i], tpr1[i], _ = roc_curve(y_test, preds)
    roc_auc1[i] = auc(fpr1[i], tpr1[i])
# Plot of a ROC curve for a specific class
for i in range(num_classes):
    plt.figure()
    plt.plot(fpr1[i], tpr1[i], label='ROC curve CNNI (area = %0.2f)' % roc_
auc1[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show();
```

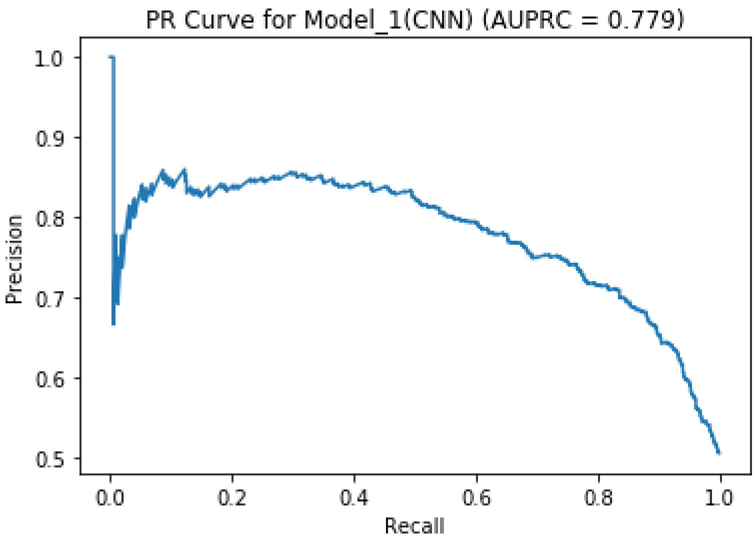
```
44/44 [==============================] - 0s 7ms/step
```





In [ ]:
```
# To determine the probabilities for positive class
y_scores = model_1.predict(X_test)
precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)

# To calculate area under the PRC
auprc = average_precision_score(y_test, y_scores)
print('AUPRC1:', auprc)

# To Plot the PR curve
plt.plot(recalls, precisions)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PR Curve for Model_1(CNN) (AUPRC = {:.3f})'.format(auprc))
plt.show()
```

```
44/44 [==============================] - 0s 7ms/step
AUPRC1: 0.7793772110950611
```



MODEL 2: CONVOLUTIONAL NEURAL NETWORKS II

In [ ]:
```
x_resized = np.zeros((x_input.shape[0], 75, 75, 3))
for i in range(x_input.shape[0]):
    x_resized[i] = resize(x_input[i], (75, 75,3), anti_aliasing=True)

print('x_input dimension: x_resized= {}'.format(x_resized.shape))
```

```
x_input dimension: x_resized= (5547, 75, 75, 3)
```

The preserve_range is set to False by default, which automatically scales the pixels of the images between 0 t0 1 (5. Image data types and what they mean — skimage 0.21.0rc1.dev0 documentation, no date)

In [ ]:
```
#To split the data with 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(x_resized,
                                                    y_target, test_size=0.2
0,
                                                    random_state=0)
```

In [ ]:
```
#To create the model:
model_2= Sequential()
model_2.add(Conv2D(filters = 32,
                 kernel_size=(4,4),
                 input_shape=(75,75,3),
                 activation='relu'))
model_2.add(MaxPooling2D(2,2))
model_2.add(Conv2D(filters = 32, kernel_size=(3,3), activation = 'relu'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Flatten())
model_2.add(Dense(units=512, activation='relu'))
model_2.add(Dense(units=1, activation='sigmoid'))
# To compile the model
lr = 0.01 # Learning rate
epochs = 25
model_2.compile(loss = 'binary_crossentropy',
               optimizer = 'adam',  metrics=['accuracy'])



print(model_2.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 72, 72, 32)        1568

 max_pooling2d_6 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 conv2d_7 (Conv2D)           (None, 34, 34, 32)        9248

 max_pooling2d_7 (MaxPooling  (None, 17, 17, 32)       0
 2D)

 flatten_3 (Flatten)         (None, 9248)              0

 dense_6 (Dense)             (None, 512)               4735488

 dense_7 (Dense)             (None, 1)                 513

=================================================================
Total params: 4,746,817
Trainable params: 4,746,817
Non-trainable params: 0
_________________________________________________________________
None
```

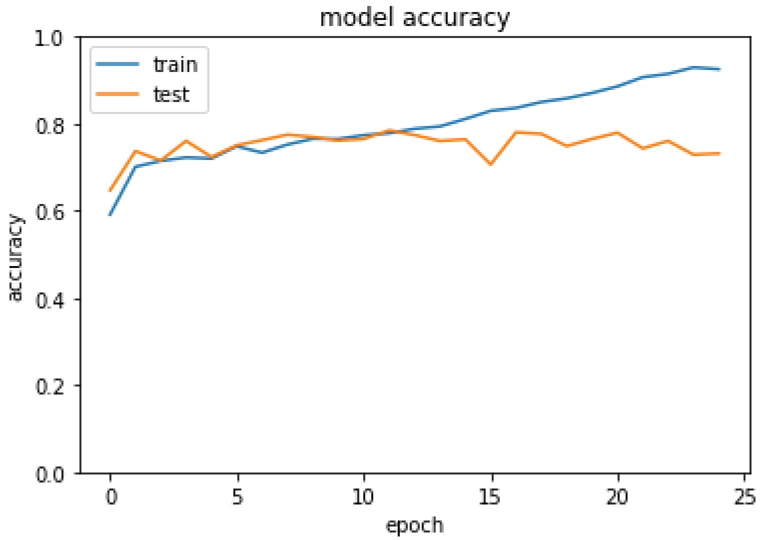
In [ ]:
```
callback = EarlyStopping(monitor='loss', patience=4)
```

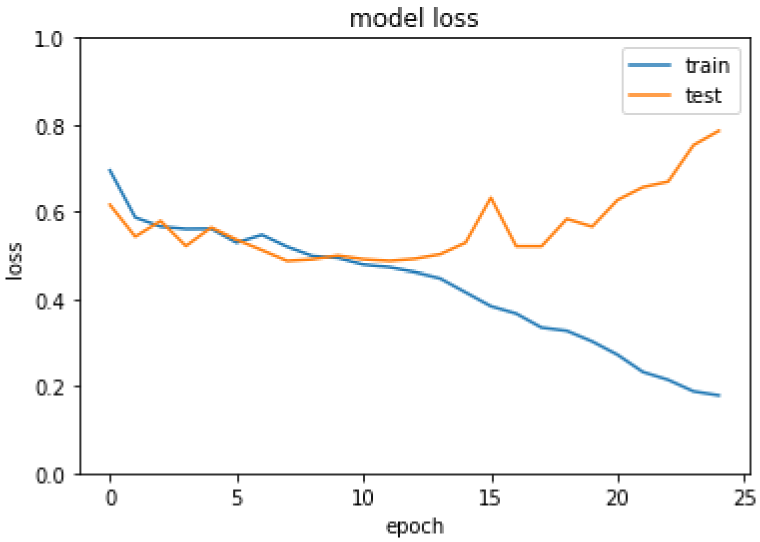
In [ ]:
```
#Fitting the model
history = model_2.fit(X_train,
                      y_train,
                      validation_data=(X_test, y_test),
                      epochs=epochs,
                      callbacks=[callback],
                      batch_size=64)


print(len(history.history['loss']))
```

```
Epoch 1/25
70/70 [==============================] - 10s 135ms/step - loss: 0.6950 - a
ccuracy: 0.5909 - val_loss: 0.6158 - val_accuracy: 0.6468
Epoch 2/25
70/70 [==============================] - 9s 128ms/step - loss: 0.5868 - ac
curacy: 0.7009 - val_loss: 0.5426 - val_accuracy: 0.7369
Epoch 3/25
70/70 [==============================] - 9s 131ms/step - loss: 0.5658 - ac
curacy: 0.7144 - val_loss: 0.5788 - val_accuracy: 0.7153
Epoch 4/25
70/70 [==============================] - 9s 135ms/step - loss: 0.5603 - ac
curacy: 0.7223 - val_loss: 0.5211 - val_accuracy: 0.7604
Epoch 5/25
70/70 [==============================] - 9s 131ms/step - loss: 0.5608 - ac
curacy: 0.7199 - val_loss: 0.5637 - val_accuracy: 0.7234
Epoch 6/25
70/70 [==============================] - 10s 136ms/step - loss: 0.5288 - a
ccuracy: 0.7480 - val_loss: 0.5357 - val_accuracy: 0.7505
Epoch 7/25
70/70 [==============================] - 10s 136ms/step - loss: 0.5468 - a
ccuracy: 0.7334 - val_loss: 0.5116 - val_accuracy: 0.7622
Epoch 8/25
70/70 [==============================] - 10s 138ms/step - loss: 0.5194 - a
ccuracy: 0.7521 - val_loss: 0.4867 - val_accuracy: 0.7748
Epoch 9/25
70/70 [==============================] - 10s 136ms/step - loss: 0.4980 - a
ccuracy: 0.7654 - val_loss: 0.4906 - val_accuracy: 0.7694
Epoch 10/25
70/70 [==============================] - 9s 129ms/step - loss: 0.4938 - ac
curacy: 0.7649 - val_loss: 0.4992 - val_accuracy: 0.7613
Epoch 11/25
70/70 [==============================] - 9s 129ms/step - loss: 0.4786 - ac
curacy: 0.7735 - val_loss: 0.4907 - val_accuracy: 0.7649
Epoch 12/25
70/70 [==============================] - 9s 130ms/step - loss: 0.4726 - ac
curacy: 0.7782 - val_loss: 0.4868 - val_accuracy: 0.7847
Epoch 13/25
70/70 [==============================] - 9s 130ms/step - loss: 0.4616 - ac
curacy: 0.7884 - val_loss: 0.4918 - val_accuracy: 0.7739
Epoch 14/25
70/70 [==============================] - 9s 129ms/step - loss: 0.4474 - ac
curacy: 0.7938 - val_loss: 0.5022 - val_accuracy: 0.7604
Epoch 15/25
70/70 [==============================] - 9s 129ms/step - loss: 0.4156 - ac
curacy: 0.8109 - val_loss: 0.5287 - val_accuracy: 0.7640
Epoch 16/25
70/70 [==============================] - 9s 131ms/step - loss: 0.3838 - ac
curacy: 0.8296 - val_loss: 0.6323 - val_accuracy: 0.7063
Epoch 17/25
70/70 [==============================] - 9s 128ms/step - loss: 0.3668 - ac
curacy: 0.8362 - val_loss: 0.5205 - val_accuracy: 0.7802
Epoch 18/25
70/70 [==============================] - 9s 129ms/step - loss: 0.3347 - ac
curacy: 0.8499 - val_loss: 0.5204 - val_accuracy: 0.7766
Epoch 19/25
70/70 [==============================] - 9s 133ms/step - loss: 0.3271 - ac
curacy: 0.8582 - val_loss: 0.5834 - val_accuracy: 0.7486
Epoch 20/25
70/70 [==============================] - 9s 129ms/step - loss: 0.3028 - ac
curacy: 0.8706 - val_loss: 0.5656 - val_accuracy: 0.7649
Epoch 21/25
```

```
70/70 [==============================] - 9s 130ms/step - loss: 0.2722 - accuracy: 0.8857 - val_loss: 0.6269 - val_accuracy: 0.7793
Epoch 22/25
70/70 [==============================] - 9s 130ms/step - loss: 0.2326 - accuracy: 0.9069 - val_loss: 0.6563 - val_accuracy: 0.7432
Epoch 23/25
70/70 [==============================] - 9s 130ms/step - loss: 0.2145 - accuracy: 0.9146 - val_loss: 0.6692 - val_accuracy: 0.7604
Epoch 24/25
70/70 [==============================] - 9s 129ms/step - loss: 0.1880 - accuracy: 0.9295 - val_loss: 0.7531 - val_accuracy: 0.7288
Epoch 25/25
70/70 [==============================] - 9s 130ms/step - loss: 0.1787 - accuracy: 0.9254 - val_loss: 0.7860 - val_accuracy: 0.7315
25
```

In [ ]:
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.ylim([0, 1])  # Set y-axis limits
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.ylim([0, 1])  # Set y-axis limits
plt.legend(['train', 'test'], loc='upper right')
plt.show()
```





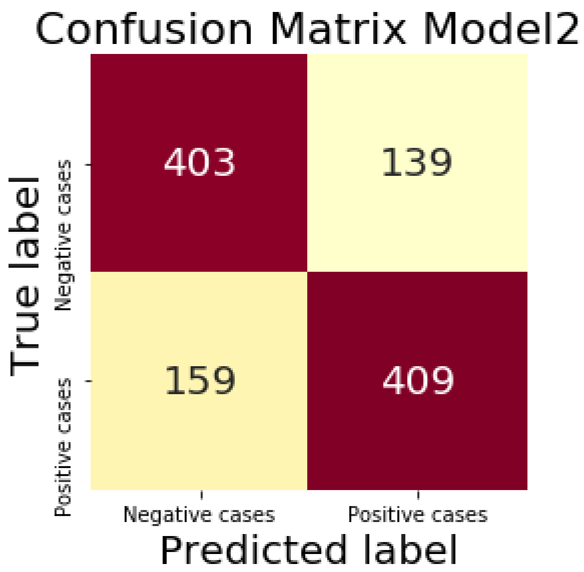
In [ ]:
```
#To evaluate the model accuracy:
score = model_2.evaluate(X_test, y_test, verbose=0)
print('model_2: CNN accuracy:', score[1], '\n')

preds = model_2.predict(X_test)
preds1 = np.round(preds)
print(preds.shape) # which means the predictions return in one-hot encoding format
print(preds.shape)
print(y_test.shape)
print(classification_report(y_test, preds1))
```

```
model_2: CNN accuracy: 0.73153156042099

35/35 [==============================] - 1s 14ms/step
(1110, 1)
(1110, 1)
(1110,)
              precision    recall  f1-score   support

           0       0.72      0.74      0.73       542
           1       0.75      0.72      0.73       568

    accuracy                           0.73      1110
   macro avg       0.73      0.73      0.73      1110
weighted avg       0.73      0.73      0.73      1110
```

In [ ]:
```
#To evaluate the confusion matrix
conf_matrix = confusion_matrix(y_test, preds1)
conf_matrix
```

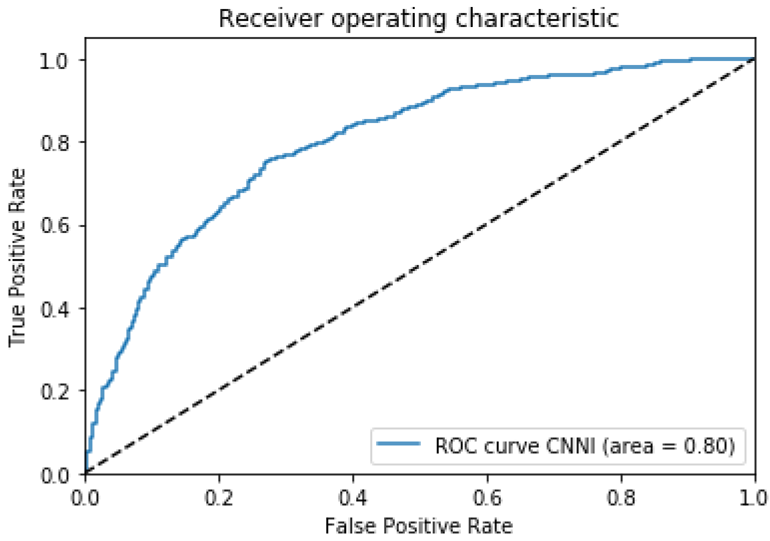
Out[ ]:
```
array([[403, 139],
       [159, 409]], dtype=int64)
```

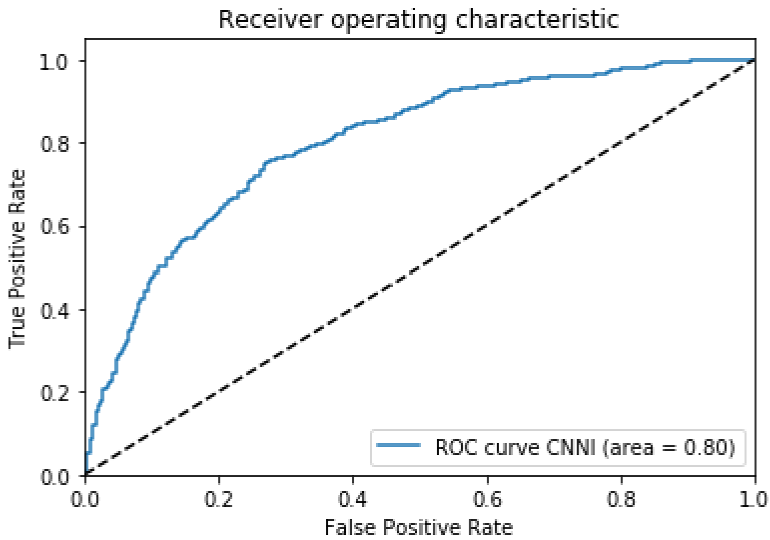
In [ ]:
```
# To plot the confusion matrix
mat = confusion_matrix(y_test, preds1)

sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Negative cases', 'Positive cases'],
            yticklabels=['Negative cases', 'Positive cases'],
            cmap='YlOrRd', annot_kws={"fontsize": 20})

plt.xlabel('Predicted label', fontsize=20)
plt.ylabel('True label', fontsize=20)
plt.title('Confusion Matrix Model2', fontsize=22)

plt.show()
```



In [ ]:
```
# To plot the ROC

y_score = model_2.predict(X_test) # get the prediction probabilities

fpr2 = dict()
tpr2 = dict()
roc_auc2 = dict()
for i in range(num_classes):
    fpr2[i], tpr2[i], _ = roc_curve(y_test, preds)
    roc_auc2[i] = auc(fpr2[i], tpr2[i])
# Plot of a ROC curve for a specific class
for i in range(num_classes):
    plt.figure()
    plt.plot(fpr2[i], tpr2[i], label='ROC curve CNNI (area = %0.2f)' % roc_
auc2[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show();
```

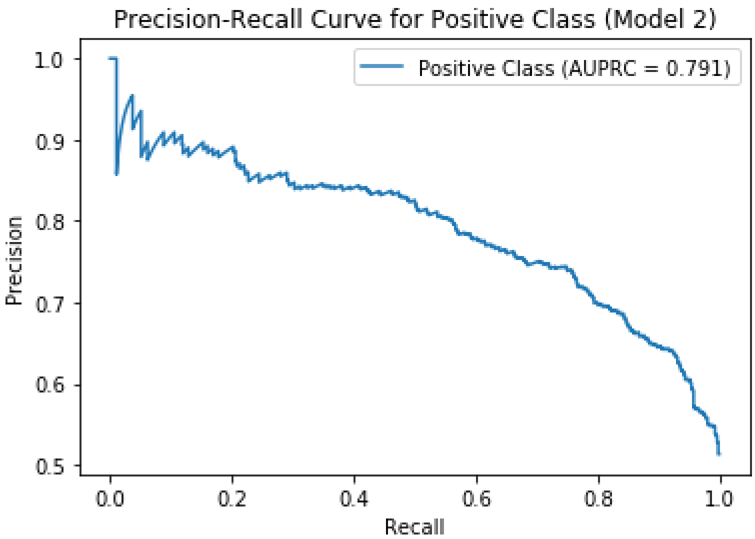
```
35/35 [==============================] - 1s 14ms/step
```





In [ ]:
```
# To plot the Precision-Recall Curve

y_scores = model_2.predict(X_test) # predicts the probability for the test data
y_test = y_test.reshape((-1, 1)) # reshape the label to match y_scores
precisions_pos, recalls_pos, thresholds_pos = precision_recall_curve(y_test, y_scores)
auprc2b = average_precision_score(y_test, y_scores)
print('AUPRC for Positive Class:', auprc2b)
plt.plot(recalls_pos, precisions_pos, label='Positive Class (AUPRC = {:.3f})'.format(auprc2b))
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Positive Class (Model 2)')
plt.legend();
```

```
35/35 [==============================] - 0s 14ms/step
AUPRC for Positive Class: 0.7908419293396383
```



MODEL 3: CONVOLUTIONAL NEURAL NETWORKS III

In [ ]:
```
#To split the data with 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(x_input, y_target, test_size=0.20, random_state=0)
#Normalization
X_train = X_train / 255.0
print('X Shape:', X_train.shape)

X_test = X_test / 255.0
print('X Shape:', X_test.shape)
```

```
X Shape: (4437, 50, 50, 3)
X Shape: (1110, 50, 50, 3)
```

In [ ]:
```
# Create the model
model_3 = Sequential()
model_3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 3)))
# first layer : convolution
model_3.add(MaxPooling2D(pool_size=(3, 3))) # second layer : pooling (reduc
e the size of the image per 3)
model_3.add(Conv2D(32, (3, 3), activation='relu'))
model_3.add(MaxPooling2D(pool_size=(3, 3)))
model_3.add(Dropout(0.25)) #Drop out with rate 0.25
model_3.add(Flatten())
model_3.add(Dense(64, activation='relu'))
model_3.add(Dropout(0.5)) #Drop out with rate 0.5
model_3.add(Dense(units=1, activation='sigmoid')) # output layer to give va
lue between 0 and 1
model_3.summary()
epochs3 = 40
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 48, 48, 32)        896       
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 16, 16, 32)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 14, 14, 32)        9248      
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 4, 4, 32)         0         
 2D)                                                             
                                                                 
 dropout_2 (Dropout)         (None, 4, 4, 32)          0         
                                                                 
 flatten_1 (Flatten)         (None, 512)               0         
                                                                 
 dense_2 (Dense)             (None, 64)                32832     
                                                                 
 dropout_3 (Dropout)         (None, 64)                0         
                                                                 
 dense_3 (Dense)             (None, 1)                 65        
                                                                 
=================================================================
Total params: 43,041
Trainable params: 43,041
Non-trainable params: 0
_________________________________________________________________
```

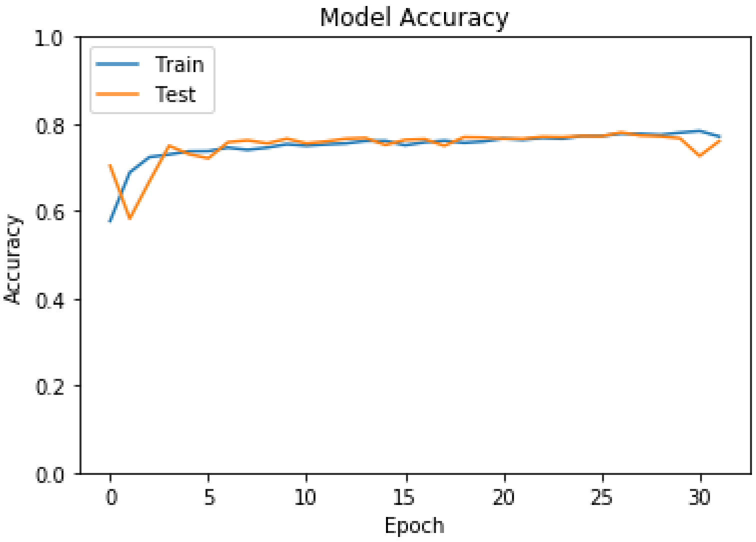
In [ ]:
```
model_3.compile(loss = 'binary_crossentropy',
               optimizer = 'adam',
                metrics=['accuracy'])
```

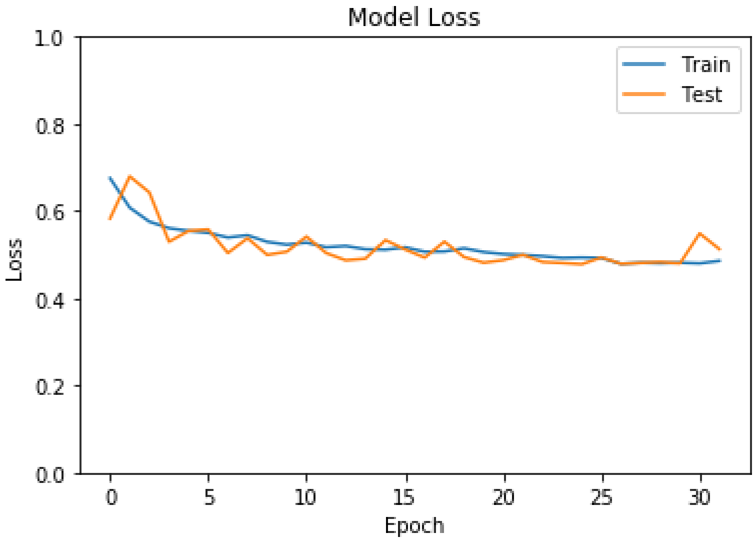
In [ ]:
```
callback = EarlyStopping(monitor='loss', patience=5)

history = model_3.fit(X_train,
                    y_train,
                    validation_data=(X_test, y_test),
                    epochs=epochs3,
                    callbacks=[callback])


print(len(history.history['loss']))
```

```
Epoch 1/40
139/139 [==============================] - 3s 19ms/step - loss: 0.6749 - a
ccuracy: 0.5767 - val_loss: 0.5824 - val_accuracy: 0.7036
Epoch 2/40
139/139 [==============================] - 2s 18ms/step - loss: 0.6075 - a
ccuracy: 0.6883 - val_loss: 0.6793 - val_accuracy: 0.5820
Epoch 3/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5753 - a
ccuracy: 0.7235 - val_loss: 0.6424 - val_accuracy: 0.6676
Epoch 4/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5603 - a
ccuracy: 0.7295 - val_loss: 0.5291 - val_accuracy: 0.7495
Epoch 5/40
139/139 [==============================] - 3s 18ms/step - loss: 0.5541 - a
ccuracy: 0.7368 - val_loss: 0.5541 - val_accuracy: 0.7306
Epoch 6/40
139/139 [==============================] - 3s 18ms/step - loss: 0.5496 - a
ccuracy: 0.7372 - val_loss: 0.5575 - val_accuracy: 0.7207
Epoch 7/40
139/139 [==============================] - 3s 18ms/step - loss: 0.5389 - a
ccuracy: 0.7453 - val_loss: 0.5033 - val_accuracy: 0.7577
Epoch 8/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5440 - a
ccuracy: 0.7401 - val_loss: 0.5372 - val_accuracy: 0.7622
Epoch 9/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5288 - a
ccuracy: 0.7453 - val_loss: 0.4989 - val_accuracy: 0.7550
Epoch 10/40
139/139 [==============================] - 3s 18ms/step - loss: 0.5225 - a
ccuracy: 0.7532 - val_loss: 0.5062 - val_accuracy: 0.7658
Epoch 11/40
139/139 [==============================] - 3s 18ms/step - loss: 0.5271 - a
ccuracy: 0.7496 - val_loss: 0.5409 - val_accuracy: 0.7550
Epoch 12/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5165 - a
ccuracy: 0.7528 - val_loss: 0.5033 - val_accuracy: 0.7595
Epoch 13/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5193 - a
ccuracy: 0.7550 - val_loss: 0.4865 - val_accuracy: 0.7658
Epoch 14/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5120 - a
ccuracy: 0.7606 - val_loss: 0.4902 - val_accuracy: 0.7676
Epoch 15/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5108 - a
ccuracy: 0.7609 - val_loss: 0.5328 - val_accuracy: 0.7514
Epoch 16/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5163 - a
ccuracy: 0.7507 - val_loss: 0.5113 - val_accuracy: 0.7631
Epoch 17/40
139/139 [==============================] - 2s 18ms/step - loss: 0.5061 - a
ccuracy: 0.7577 - val_loss: 0.4927 - val_accuracy: 0.7649
Epoch 18/40
139/139 [==============================] - 2s 17ms/step - loss: 0.5067 - a
ccuracy: 0.7613 - val_loss: 0.5293 - val_accuracy: 0.7495
Epoch 19/40
139/139 [==============================] - 3s 18ms/step - loss: 0.5141 - a
ccuracy: 0.7564 - val_loss: 0.4940 - val_accuracy: 0.7694
Epoch 20/40
139/139 [==============================] - 2s 17ms/step - loss: 0.5054 - a
ccuracy: 0.7595 - val_loss: 0.4810 - val_accuracy: 0.7685
Epoch 21/40
```

```
139/139 [==============================] - 2s 18ms/step - loss: 0.5008 - a
ccuracy: 0.7663 - val_loss: 0.4868 - val_accuracy: 0.7658
Epoch 22/40
139/139 [==============================] - 3s 18ms/step - loss: 0.4988 - a
ccuracy: 0.7634 - val_loss: 0.4990 - val_accuracy: 0.7658
Epoch 23/40
139/139 [==============================] - 2s 18ms/step - loss: 0.4960 - a
ccuracy: 0.7676 - val_loss: 0.4825 - val_accuracy: 0.7703
Epoch 24/40
139/139 [==============================] - 2s 18ms/step - loss: 0.4918 - a
ccuracy: 0.7656 - val_loss: 0.4803 - val_accuracy: 0.7694
Epoch 25/40
139/139 [==============================] - 2s 18ms/step - loss: 0.4928 - a
ccuracy: 0.7721 - val_loss: 0.4775 - val_accuracy: 0.7721
Epoch 26/40
139/139 [==============================] - 3s 18ms/step - loss: 0.4918 - a
ccuracy: 0.7719 - val_loss: 0.4931 - val_accuracy: 0.7712
Epoch 27/40
139/139 [==============================] - 2s 18ms/step - loss: 0.4784 - a
ccuracy: 0.7776 - val_loss: 0.4783 - val_accuracy: 0.7802
Epoch 28/40
139/139 [==============================] - 3s 18ms/step - loss: 0.4808 - a
ccuracy: 0.7769 - val_loss: 0.4805 - val_accuracy: 0.7730
Epoch 29/40
139/139 [==============================] - 2s 18ms/step - loss: 0.4797 - a
ccuracy: 0.7746 - val_loss: 0.4825 - val_accuracy: 0.7712
Epoch 30/40
139/139 [==============================] - 2s 18ms/step - loss: 0.4810 - a
ccuracy: 0.7796 - val_loss: 0.4794 - val_accuracy: 0.7667
Epoch 31/40
139/139 [==============================] - 3s 18ms/step - loss: 0.4794 - a
ccuracy: 0.7836 - val_loss: 0.5482 - val_accuracy: 0.7261
Epoch 32/40
139/139 [==============================] - 3s 18ms/step - loss: 0.4851 - a
ccuracy: 0.7708 - val_loss: 0.5123 - val_accuracy: 0.7604
32
```

In [ ]:
```
# visualize the results
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.ylim([0, 1])
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.ylim([0, 1])
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```





In [ ]:
```
#To compute the Classification report
score = model_3.evaluate(X_test, y_test, verbose=0)
print('\nKeras MODEL3 - accuracy:', score[1], '\n')

preds = model_3.predict(X_test)
preds1 = np.round(preds).astype(int)

print(preds.shape)
print(preds.shape)
print(y_test.shape)
print(classification_report(y_test, preds1))
conf_matx = confusion_matrix(y_test, preds1)
```

```
Keras MODEL3 - accuracy: 0.7603603601455688

35/35 [==============================] - 0s 5ms/step
(1110, 1)
(1110, 1)
(1110,)
              precision    recall  f1-score   support

           0       0.85      0.62      0.72       542
           1       0.71      0.90      0.79       568

    accuracy                           0.76      1110
   macro avg       0.78      0.76      0.75      1110
weighted avg       0.78      0.76      0.76      1110
```

In [ ]:
```
def build_model(hp):
    model_3= Sequential()
    model_3.add(Conv2D(hp.Choice('filters', [8, 16, 32]),
    kernel_size=(3, 3),
    input_shape=(50, 50, 3),
    activation='relu'))
    model_3.add(MaxPooling2D(pool_size=(3, 3))) # second layer : pooling (r
educe the size of the image per 3)
    model_3.add(Conv2D(32, (3, 3), activation='relu'))
    model_3.add(MaxPooling2D(pool_size=(3, 3)))
    model_3.add(Dropout(0.25))
    model_3.add(Flatten())
    model_3.add(Dense(64, activation='relu'))
    model_3.add(Dropout(0.5))
    model_3.add(Dense(units=1, activation='sigmoid')) # num_classes = 2
    # To compile the model
    lr = hp.Choice('learning_rate', [0.1, 0.01, 0.001]) # Learning rate
    optimizer = hp.Choice('optimizer', ['adam', 'rmsprop', 'sgd'])
    epochs = 30
    model_3.compile(loss = 'binary_crossentropy',
               optimizer = 'adam',
                metrics=['accuracy'])
    return model_3
```

In [ ]:
```
#Setting the parameters for the tuner
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=5);
```

```
INFO:tensorflow:Reloading Tuner from .\untitled_project\tuner0.json
```

In [ ]:
```
# Step 3: start the search
tuner.search(X_train, y_train, epochs=20, validation_data=(X_test, y_test))
best_model = tuner.get_best_models()[0]
```

```
Trial 5 Complete [00h 00m 30s]
val_accuracy: 0.7666666507720947

Best val_accuracy So Far: 0.7756756544113159
Total elapsed time: 00h 03m 00s
INFO:tensorflow:Oracle triggered exit
```

In [ ]:
```
print(tuner.results_summary())
```

```
Results summary
Results in .\untitled_project
Showing 10 best trials
Objective(name="val_accuracy", direction="max")

Trial 4 summary
Hyperparameters:
filters: 16
learning_rate: 0.01
optimizer: sgd
Score: 0.7801801562309265

Trial 1 summary
Hyperparameters:
filters: 8
learning_rate: 0.1
optimizer: rmsprop
Score: 0.7747747898101807

Trial 0 summary
Hyperparameters:
filters: 32
learning_rate: 0.01
optimizer: sgd
Score: 0.7702702879905701

Trial 2 summary
Hyperparameters:
filters: 8
learning_rate: 0.1
optimizer: sgd
Score: 0.7702702879905701

Trial 3 summary
Hyperparameters:
filters: 16
learning_rate: 0.1
optimizer: rmsprop
Score: 0.76846843957901
None
```

In [ ]:
```
# To get the best hperparamters
best_hp = tuner.get_best_hyperparameters()[0]
best_filters = best_hp.get('filters')
best_learning_rate = best_hp.get('learning_rate')
print('Best Filters: {}'.format(best_filters))
print('Best Learning Rate: {}'.format(best_learning_rate))
```

```
Best Filters: 8
Best Learning Rate: 0.01
```

In [ ]:
```
#Train-Test Split
X_train, X_test, y_train, y_test =train_test_split(x_input,
                                                    y_target, test_size = 0.
15, random_state = 5)

#Normalization
X_train = X_train / 255.0
print('X Shape:', X_train.shape)

X_test = X_test / 255.0
print('X Shape:', X_test.shape)
```

```
X Shape: (4714, 50, 50, 3)
X Shape: (833, 50, 50, 3)
```

In [ ]:
```
#Instantiating the augmented images
datagen = ImageDataGenerator(
        rotation_range=50,
        width_shift_range=0.2,
        height_shift_range=0.3,
        vertical_flip=True,
        horizontal_flip=False,
        zoom_range=0.5,
        fill_mode='wrap')
```

In [ ]:
```
# Define custom callback to stop training at desired validation accuracy
class samsoncallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        if logs.get('val_accuracy') > 0.805:
            print('\nStopping training as accuracy has reached 80.5%')
            self.model.stop_training = True

# Train model with callbacks
history = model_3.fit(datagen.flow(X_train, y_train, batch_size=64), epochs
=200,
                      validation_data=(X_test, y_test), verbose=2, shuffle=
True,
                      callbacks=[checkpoint_callback, samsoncallback()])
```

```
Epoch 1/200
70/70 - 3s - loss: 0.4464 - accuracy: 0.7922 - val_loss: 0.4527 - val_accu
racy: 0.7982 - 3s/epoch - 45ms/step
Epoch 2/200
70/70 - 3s - loss: 0.4361 - accuracy: 0.8023 - val_loss: 0.4554 - val_accu
racy: 0.8000 - 3s/epoch - 45ms/step
Epoch 3/200
70/70 - 3s - loss: 0.4350 - accuracy: 0.7974 - val_loss: 0.4533 - val_accu
racy: 0.8000 - 3s/epoch - 44ms/step
Epoch 4/200
70/70 - 3s - loss: 0.4424 - accuracy: 0.7960 - val_loss: 0.4463 - val_accu
racy: 0.7946 - 3s/epoch - 44ms/step
Epoch 5/200
70/70 - 3s - loss: 0.4396 - accuracy: 0.8010 - val_loss: 0.4459 - val_accu
racy: 0.7991 - 3s/epoch - 45ms/step
Epoch 6/200
70/70 - 3s - loss: 0.4484 - accuracy: 0.7942 - val_loss: 0.4986 - val_accu
racy: 0.7667 - 3s/epoch - 44ms/step
Epoch 7/200
70/70 - 3s - loss: 0.4457 - accuracy: 0.7938 - val_loss: 0.4687 - val_accu
racy: 0.7874 - 3s/epoch - 45ms/step
Epoch 8/200
70/70 - 3s - loss: 0.4362 - accuracy: 0.7978 - val_loss: 0.4585 - val_accu
racy: 0.7829 - 3s/epoch - 44ms/step
Epoch 9/200
70/70 - 3s - loss: 0.4483 - accuracy: 0.7956 - val_loss: 0.4543 - val_accu
racy: 0.7946 - 3s/epoch - 44ms/step
Epoch 10/200
70/70 - 3s - loss: 0.4406 - accuracy: 0.7958 - val_loss: 0.4572 - val_accu
racy: 0.7910 - 3s/epoch - 46ms/step
Epoch 11/200
70/70 - 3s - loss: 0.4484 - accuracy: 0.7981 - val_loss: 0.4767 - val_accu
racy: 0.7595 - 3s/epoch - 45ms/step
Epoch 12/200
70/70 - 3s - loss: 0.4472 - accuracy: 0.7940 - val_loss: 0.4584 - val_accu
racy: 0.7973 - 3s/epoch - 45ms/step
Epoch 13/200
70/70 - 3s - loss: 0.4433 - accuracy: 0.7960 - val_loss: 0.4793 - val_accu
racy: 0.7793 - 3s/epoch - 45ms/step
Epoch 14/200
70/70 - 3s - loss: 0.4508 - accuracy: 0.7895 - val_loss: 0.4711 - val_accu
racy: 0.7820 - 3s/epoch - 45ms/step
Epoch 15/200
70/70 - 3s - loss: 0.4522 - accuracy: 0.7918 - val_loss: 0.4486 - val_accu
racy: 0.7991 - 3s/epoch - 45ms/step
Epoch 16/200
70/70 - 3s - loss: 0.4397 - accuracy: 0.8053 - val_loss: 0.4528 - val_accu
racy: 0.7901 - 3s/epoch - 44ms/step
Epoch 17/200
70/70 - 3s - loss: 0.4417 - accuracy: 0.7933 - val_loss: 0.4612 - val_accu
racy: 0.7982 - 3s/epoch - 45ms/step
Epoch 18/200
70/70 - 3s - loss: 0.4443 - accuracy: 0.7999 - val_loss: 0.4514 - val_accu
racy: 0.7955 - 3s/epoch - 44ms/step
Epoch 19/200
70/70 - 3s - loss: 0.4372 - accuracy: 0.7918 - val_loss: 0.4472 - val_accu
racy: 0.8009 - 3s/epoch - 44ms/step
Epoch 20/200
70/70 - 3s - loss: 0.4412 - accuracy: 0.7954 - val_loss: 0.4507 - val_accu
racy: 0.7973 - 3s/epoch - 45ms/step
Epoch 21/200
```

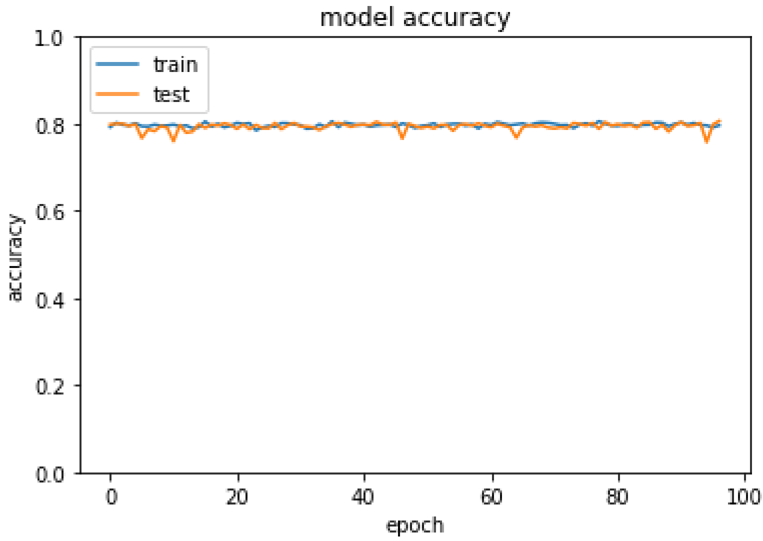
```
70/70 - 3s - loss: 0.4336 - accuracy: 0.8019 - val_loss: 0.4530 - val_accu
racy: 0.7883 - 3s/epoch - 46ms/step
Epoch 22/200
70/70 - 3s - loss: 0.4463 - accuracy: 0.7994 - val_loss: 0.4581 - val_accu
racy: 0.7982 - 3s/epoch - 46ms/step
Epoch 23/200
70/70 - 3s - loss: 0.4437 - accuracy: 0.8012 - val_loss: 0.4511 - val_accu
racy: 0.7874 - 3s/epoch - 46ms/step
Epoch 24/200
70/70 - 3s - loss: 0.4516 - accuracy: 0.7848 - val_loss: 0.4450 - val_accu
racy: 0.7973 - 3s/epoch - 46ms/step
Epoch 25/200
70/70 - 3s - loss: 0.4474 - accuracy: 0.7915 - val_loss: 0.4637 - val_accu
racy: 0.7901 - 3s/epoch - 46ms/step
Epoch 26/200
70/70 - 3s - loss: 0.4442 - accuracy: 0.7949 - val_loss: 0.4645 - val_accu
racy: 0.7892 - 3s/epoch - 45ms/step
Epoch 27/200
70/70 - 3s - loss: 0.4462 - accuracy: 0.7927 - val_loss: 0.4567 - val_accu
racy: 0.8018 - 3s/epoch - 46ms/step
Epoch 28/200
70/70 - 3s - loss: 0.4418 - accuracy: 0.8014 - val_loss: 0.4541 - val_accu
racy: 0.7874 - 3s/epoch - 45ms/step
Epoch 29/200
70/70 - 3s - loss: 0.4395 - accuracy: 0.8012 - val_loss: 0.4488 - val_accu
racy: 0.7964 - 3s/epoch - 46ms/step
Epoch 30/200
70/70 - 3s - loss: 0.4461 - accuracy: 0.7985 - val_loss: 0.4508 - val_accu
racy: 0.8018 - 3s/epoch - 45ms/step
Epoch 31/200
70/70 - 3s - loss: 0.4417 - accuracy: 0.7967 - val_loss: 0.4615 - val_accu
racy: 0.7937 - 3s/epoch - 47ms/step
Epoch 32/200
70/70 - 3s - loss: 0.4513 - accuracy: 0.7884 - val_loss: 0.4606 - val_accu
racy: 0.7937 - 3s/epoch - 46ms/step
Epoch 33/200
70/70 - 3s - loss: 0.4523 - accuracy: 0.7895 - val_loss: 0.4723 - val_accu
racy: 0.7919 - 3s/epoch - 45ms/step
Epoch 34/200
70/70 - 3s - loss: 0.4479 - accuracy: 0.7987 - val_loss: 0.4533 - val_accu
racy: 0.7856 - 3s/epoch - 45ms/step
Epoch 35/200
70/70 - 3s - loss: 0.4428 - accuracy: 0.7922 - val_loss: 0.4622 - val_accu
racy: 0.7937 - 3s/epoch - 46ms/step
Epoch 36/200
70/70 - 3s - loss: 0.4375 - accuracy: 0.8055 - val_loss: 0.4641 - val_accu
racy: 0.7991 - 3s/epoch - 46ms/step
Epoch 37/200
70/70 - 3s - loss: 0.4457 - accuracy: 0.7922 - val_loss: 0.4775 - val_accu
racy: 0.8018 - 3s/epoch - 46ms/step
Epoch 38/200
70/70 - 3s - loss: 0.4329 - accuracy: 0.8026 - val_loss: 0.4546 - val_accu
racy: 0.7982 - 3s/epoch - 45ms/step
Epoch 39/200
70/70 - 3s - loss: 0.4403 - accuracy: 0.7999 - val_loss: 0.4508 - val_accu
racy: 0.7928 - 3s/epoch - 46ms/step
Epoch 40/200
70/70 - 3s - loss: 0.4356 - accuracy: 0.7967 - val_loss: 0.4659 - val_accu
racy: 0.7973 - 3s/epoch - 46ms/step
Epoch 41/200
70/70 - 3s - loss: 0.4427 - accuracy: 0.7981 - val_loss: 0.4654 - val_accu
```

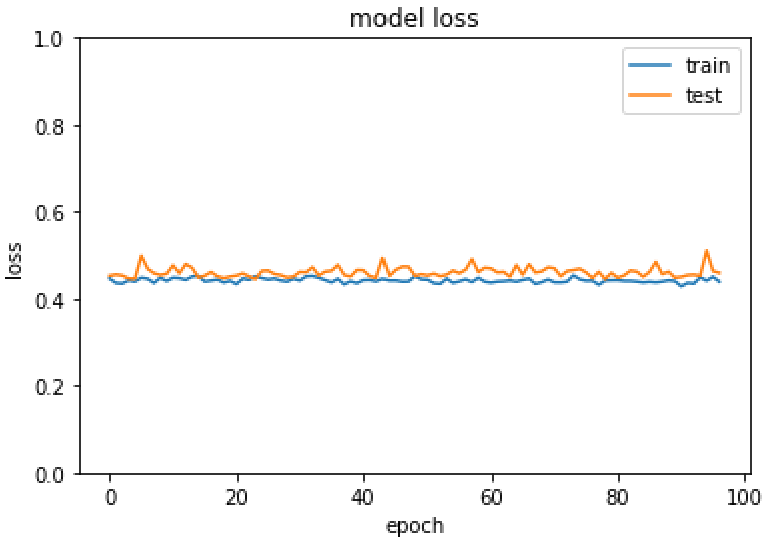
```
racy: 0.7982 - 3s/epoch - 45ms/step
Epoch 42/200
70/70 - 3s - loss: 0.4434 - accuracy: 0.7933 - val_loss: 0.4521 - val_accu
racy: 0.7964 - 3s/epoch - 45ms/step
Epoch 43/200
70/70 - 3s - loss: 0.4400 - accuracy: 0.7960 - val_loss: 0.4480 - val_accu
racy: 0.8045 - 3s/epoch - 45ms/step
Epoch 44/200
70/70 - 3s - loss: 0.4455 - accuracy: 0.7969 - val_loss: 0.4934 - val_accu
racy: 0.7982 - 3s/epoch - 45ms/step
Epoch 45/200
70/70 - 3s - loss: 0.4418 - accuracy: 0.7983 - val_loss: 0.4526 - val_accu
racy: 0.7973 - 3s/epoch - 45ms/step
Epoch 46/200
70/70 - 3s - loss: 0.4419 - accuracy: 0.7927 - val_loss: 0.4666 - val_accu
racy: 0.8027 - 3s/epoch - 45ms/step
Epoch 47/200
70/70 - 3s - loss: 0.4395 - accuracy: 0.8008 - val_loss: 0.4735 - val_accu
racy: 0.7658 - 3s/epoch - 45ms/step
Epoch 48/200
70/70 - 3s - loss: 0.4399 - accuracy: 0.7972 - val_loss: 0.4732 - val_accu
racy: 0.8000 - 3s/epoch - 45ms/step
Epoch 49/200
70/70 - 3s - loss: 0.4519 - accuracy: 0.7902 - val_loss: 0.4525 - val_accu
racy: 0.7946 - 3s/epoch - 45ms/step
Epoch 50/200
70/70 - 3s - loss: 0.4446 - accuracy: 0.7947 - val_loss: 0.4561 - val_accu
racy: 0.7901 - 3s/epoch - 46ms/step
Epoch 51/200
70/70 - 3s - loss: 0.4438 - accuracy: 0.7967 - val_loss: 0.4535 - val_accu
racy: 0.7937 - 3s/epoch - 45ms/step
Epoch 52/200
70/70 - 3s - loss: 0.4356 - accuracy: 0.8008 - val_loss: 0.4576 - val_accu
racy: 0.7892 - 3s/epoch - 46ms/step
Epoch 53/200
70/70 - 3s - loss: 0.4344 - accuracy: 0.7924 - val_loss: 0.4517 - val_accu
racy: 0.7973 - 3s/epoch - 46ms/step
Epoch 54/200
70/70 - 3s - loss: 0.4460 - accuracy: 0.7981 - val_loss: 0.4551 - val_accu
racy: 0.7973 - 3s/epoch - 45ms/step
Epoch 55/200
70/70 - 3s - loss: 0.4365 - accuracy: 0.7996 - val_loss: 0.4644 - val_accu
racy: 0.7838 - 3s/epoch - 46ms/step
Epoch 56/200
70/70 - 3s - loss: 0.4399 - accuracy: 0.7999 - val_loss: 0.4585 - val_accu
racy: 0.7973 - 3s/epoch - 46ms/step
Epoch 57/200
70/70 - 3s - loss: 0.4447 - accuracy: 0.7956 - val_loss: 0.4666 - val_accu
racy: 0.7991 - 3s/epoch - 46ms/step
Epoch 58/200
70/70 - 3s - loss: 0.4382 - accuracy: 0.7987 - val_loss: 0.4911 - val_accu
racy: 0.7955 - 3s/epoch - 45ms/step
Epoch 59/200
70/70 - 3s - loss: 0.4481 - accuracy: 0.7890 - val_loss: 0.4611 - val_accu
racy: 0.8009 - 3s/epoch - 46ms/step
Epoch 60/200
70/70 - 3s - loss: 0.4392 - accuracy: 0.7999 - val_loss: 0.4711 - val_accu
racy: 0.7955 - 3s/epoch - 45ms/step
Epoch 61/200
70/70 - 3s - loss: 0.4370 - accuracy: 0.7969 - val_loss: 0.4694 - val_accu
racy: 0.7919 - 3s/epoch - 45ms/step
```

```
Epoch 62/200
70/70 - 3s - loss: 0.4395 - accuracy: 0.8037 - val_loss: 0.4604 - val_accu
racy: 0.8000 - 3s/epoch - 47ms/step
Epoch 63/200
70/70 - 3s - loss: 0.4402 - accuracy: 0.8001 - val_loss: 0.4615 - val_accu
racy: 0.7955 - 3s/epoch - 45ms/step
Epoch 64/200
70/70 - 3s - loss: 0.4418 - accuracy: 0.7967 - val_loss: 0.4504 - val_accu
racy: 0.7919 - 3s/epoch - 45ms/step
Epoch 65/200
70/70 - 3s - loss: 0.4394 - accuracy: 0.7985 - val_loss: 0.4771 - val_accu
racy: 0.7676 - 3s/epoch - 46ms/step
Epoch 66/200
70/70 - 3s - loss: 0.4431 - accuracy: 0.8005 - val_loss: 0.4549 - val_accu
racy: 0.7919 - 3s/epoch - 45ms/step
Epoch 67/200
70/70 - 3s - loss: 0.4463 - accuracy: 0.7960 - val_loss: 0.4791 - val_accu
racy: 0.7964 - 3s/epoch - 46ms/step
Epoch 68/200
70/70 - 3s - loss: 0.4341 - accuracy: 0.8010 - val_loss: 0.4600 - val_accu
racy: 0.7937 - 3s/epoch - 46ms/step
Epoch 69/200
70/70 - 3s - loss: 0.4381 - accuracy: 0.8030 - val_loss: 0.4629 - val_accu
racy: 0.7964 - 3s/epoch - 46ms/step
Epoch 70/200
70/70 - 3s - loss: 0.4443 - accuracy: 0.8021 - val_loss: 0.4721 - val_accu
racy: 0.7910 - 3s/epoch - 47ms/step
Epoch 71/200
70/70 - 3s - loss: 0.4377 - accuracy: 0.7996 - val_loss: 0.4697 - val_accu
racy: 0.7892 - 3s/epoch - 46ms/step
Epoch 72/200
70/70 - 3s - loss: 0.4375 - accuracy: 0.7967 - val_loss: 0.4516 - val_accu
racy: 0.7919 - 3s/epoch - 45ms/step
Epoch 73/200
70/70 - 3s - loss: 0.4393 - accuracy: 0.7972 - val_loss: 0.4634 - val_accu
racy: 0.7892 - 3s/epoch - 47ms/step
Epoch 74/200
70/70 - 3s - loss: 0.4536 - accuracy: 0.7897 - val_loss: 0.4657 - val_accu
racy: 0.8018 - 3s/epoch - 46ms/step
Epoch 75/200
70/70 - 3s - loss: 0.4446 - accuracy: 0.7978 - val_loss: 0.4687 - val_accu
racy: 0.7991 - 3s/epoch - 46ms/step
Epoch 76/200
70/70 - 3s - loss: 0.4412 - accuracy: 0.8010 - val_loss: 0.4603 - val_accu
racy: 0.7955 - 3s/epoch - 46ms/step
Epoch 77/200
70/70 - 3s - loss: 0.4415 - accuracy: 0.7954 - val_loss: 0.4472 - val_accu
racy: 0.8018 - 3s/epoch - 45ms/step
Epoch 78/200
70/70 - 3s - loss: 0.4321 - accuracy: 0.8057 - val_loss: 0.4616 - val_accu
racy: 0.7883 - 3s/epoch - 47ms/step
Epoch 79/200
70/70 - 3s - loss: 0.4419 - accuracy: 0.8001 - val_loss: 0.4451 - val_accu
racy: 0.8036 - 3s/epoch - 46ms/step
Epoch 80/200
70/70 - 3s - loss: 0.4426 - accuracy: 0.7960 - val_loss: 0.4592 - val_accu
racy: 0.7955 - 3s/epoch - 45ms/step
Epoch 81/200
70/70 - 3s - loss: 0.4427 - accuracy: 0.7951 - val_loss: 0.4480 - val_accu
racy: 0.7991 - 3s/epoch - 45ms/step
Epoch 82/200
```

```
70/70 - 3s - loss: 0.4409 - accuracy: 0.7990 - val_loss: 0.4535 - val_accu
racy: 0.7946 - 3s/epoch - 45ms/step
Epoch 83/200
70/70 - 3s - loss: 0.4408 - accuracy: 0.7994 - val_loss: 0.4645 - val_accu
racy: 0.7973 - 3s/epoch - 45ms/step
Epoch 84/200
70/70 - 3s - loss: 0.4396 - accuracy: 0.7945 - val_loss: 0.4622 - val_accu
racy: 0.7910 - 3s/epoch - 45ms/step
Epoch 85/200
70/70 - 3s - loss: 0.4375 - accuracy: 0.7972 - val_loss: 0.4504 - val_accu
racy: 0.8027 - 3s/epoch - 45ms/step
Epoch 86/200
70/70 - 3s - loss: 0.4391 - accuracy: 0.7976 - val_loss: 0.4615 - val_accu
racy: 0.8036 - 3s/epoch - 46ms/step
Epoch 87/200
70/70 - 3s - loss: 0.4376 - accuracy: 0.8028 - val_loss: 0.4845 - val_accu
racy: 0.7883 - 3s/epoch - 46ms/step
Epoch 88/200
70/70 - 3s - loss: 0.4395 - accuracy: 0.8021 - val_loss: 0.4565 - val_accu
racy: 0.7973 - 3s/epoch - 45ms/step
Epoch 89/200
70/70 - 3s - loss: 0.4422 - accuracy: 0.7940 - val_loss: 0.4620 - val_accu
racy: 0.7820 - 3s/epoch - 45ms/step
Epoch 90/200
70/70 - 3s - loss: 0.4406 - accuracy: 0.7996 - val_loss: 0.4484 - val_accu
racy: 0.7964 - 3s/epoch - 46ms/step
Epoch 91/200
70/70 - 3s - loss: 0.4284 - accuracy: 0.8028 - val_loss: 0.4500 - val_accu
racy: 0.8036 - 3s/epoch - 45ms/step
Epoch 92/200
70/70 - 3s - loss: 0.4363 - accuracy: 0.7976 - val_loss: 0.4542 - val_accu
racy: 0.7946 - 3s/epoch - 45ms/step
Epoch 93/200
70/70 - 3s - loss: 0.4348 - accuracy: 0.8017 - val_loss: 0.4550 - val_accu
racy: 0.7964 - 3s/epoch - 46ms/step
Epoch 94/200
70/70 - 3s - loss: 0.4494 - accuracy: 0.7972 - val_loss: 0.4525 - val_accu
racy: 0.8018 - 3s/epoch - 45ms/step
Epoch 95/200
70/70 - 3s - loss: 0.4413 - accuracy: 0.7960 - val_loss: 0.5107 - val_accu
racy: 0.7577 - 3s/epoch - 46ms/step
Epoch 96/200
70/70 - 3s - loss: 0.4503 - accuracy: 0.7915 - val_loss: 0.4637 - val_accu
racy: 0.7982 - 3s/epoch - 47ms/step
Epoch 97/200

Stopping training as accuracy has reached 80.5%
70/70 - 3s - loss: 0.4393 - accuracy: 0.7963 - val_loss: 0.4592 - val_accu
racy: 0.8063 - 3s/epoch - 46ms/step
```

In [ ]:
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim([0, 1])
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.ylim([0, 1])
plt.show()
```

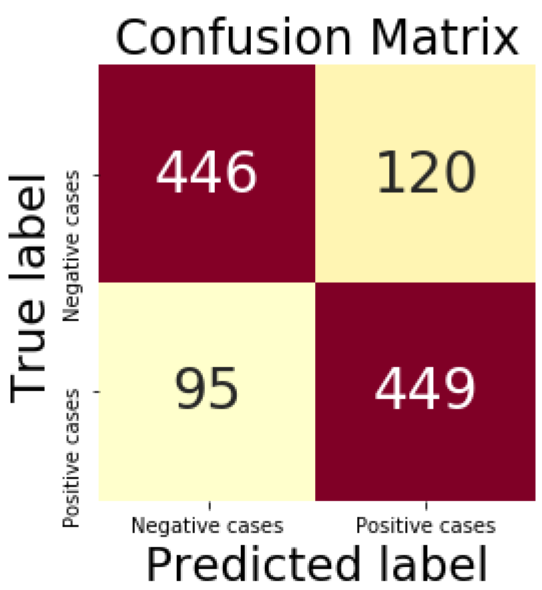




In [ ]:
```
score = model_3.evaluate(X_test, y_test, verbose=0)
print('\nKeras - accuracy:', score[1], '\n')

y_preds = model_3.predict(X_test)
y_preds1= np.round(y_preds)
print(y_preds1.shape) # which means the predictions return in one-hot encod
ing format
y_preds = np.argmax(y_preds1, axis=1)
print(y_preds.shape)
print(classification_report(y_test, y_preds1))
conf_matx = confusion_matrix(y_test, y_preds1)
```

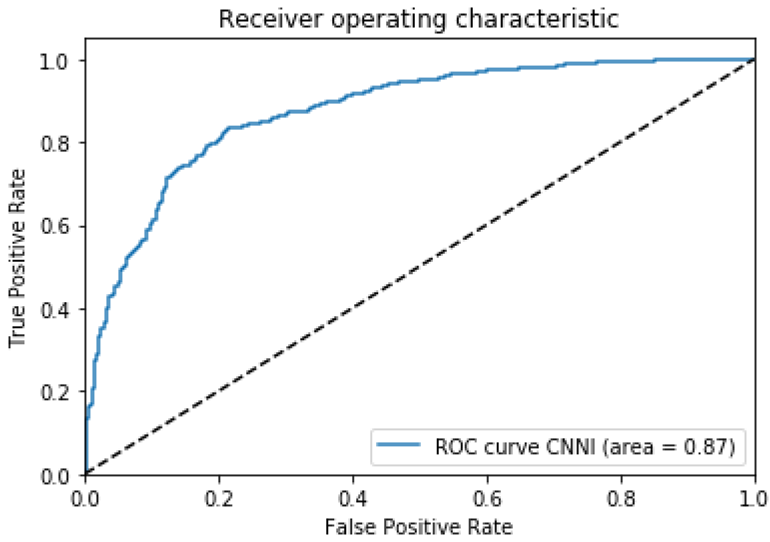
```
Keras - accuracy: 0.8063063025474548

35/35 [==============================] - 0s 5ms/step
(1110, 1)
(1110,)
              precision    recall  f1-score   support

           0       0.82      0.79      0.81       566
           1       0.79      0.83      0.81       544

    accuracy                           0.81      1110
   macro avg       0.81      0.81      0.81      1110
weighted avg       0.81      0.81      0.81      1110
```

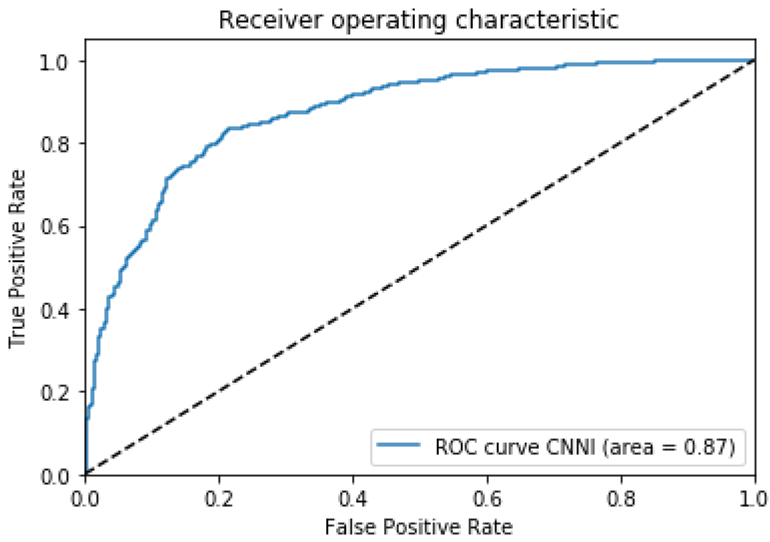
In [ ]:
```
# To plot the confusion matrix
mat = confusion_matrix(y_test, y_preds1)

sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Negative cases', 'Positive cases'],
            yticklabels=['Negative cases', 'Positive cases'],
            cmap='YlOrRd', annot_kws={"fontsize": 28})

plt.xlabel('Predicted label', fontsize=23)
plt.ylabel('True label', fontsize=23)
plt.title('Confusion Matrix', fontsize=24)

plt.show()
```



In [ ]:
```
y_score = model_3.predict(X_test) # get the prediction probabilities

fpr3 = dict()
tpr3 = dict()
roc_auc3 = dict()
for i in range(num_classes):
    fpr3[i], tpr3[i], _ = roc_curve(y_test, y_preds)
    roc_auc3[i] = auc(fpr3[i], tpr3[i])
# Plot of a ROC curve for a specific class
for i in range(num_classes):
    plt.figure()
    plt.plot(fpr3[i], tpr3[i], label='ROC curve CNNI (area = %0.2f)' % roc_
auc3[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show();
```

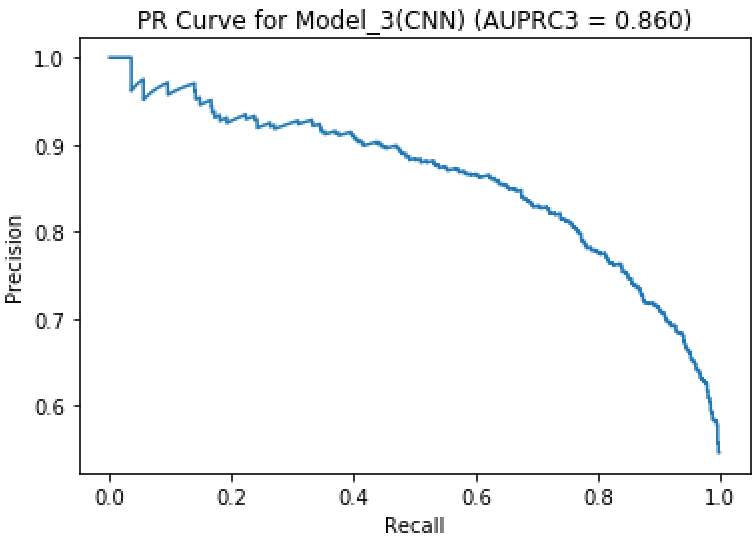
```
35/35 [==============================] - 0s 5ms/step
```





In [ ]:
```
# Get predicted probabilities for positive class
y_scores = model_3.predict(X_test)
precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)

# To calculate area under the PRC
auprc3 = average_precision_score(y_test, y_scores)
print('AUPRC:', auprc3)

# Plot the PR curve
plt.plot(recalls, precisions)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PR Curve for Model_3(CNN) (AUPRC3 = {:.3f})'.format(auprc3))
plt.show()
```

```
44/44 [==============================] - 0s 4ms/step
AUPRC: 0.8602126843280227
```



MODEL 4: PRETRAINED MODEL(MobileNetV2)

In [ ]:
```
#Instantiating the model
base_model = MobileNetV2(weights='imagenet',
                         input_shape=(50, 50, 3),
                         include_top=False)
base_model.trainable = False
inputs = Input(shape=(50, 50, 3))
x = base_model(inputs, training=False)
x1 = GlobalAveragePooling2D()(x)
x2 = Dense(128, activation='relu')(x1) # extra dense layer with 128 units a
nd relu activation
x2 = Dropout(0.2)(x2) # Add a dropout layer with a rate of 0.2
outputs = Dense(1, activation='sigmoid')(x2)
model_4 = Model(inputs, outputs)
model_4.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc
uracy'])
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is
not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will b
e loaded as the default.
```

In [ ]:
```
model_4.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_6 (InputLayer)        [(None, 50, 50, 3)]       0         
                                                                 
 mobilenetv2_1.00_224 (Funct  (None, 2, 2, 1280)       2257984   
 ional)                                                          
                                                                 
 global_average_pooling2d_2   (None, 1280)             0         
 (GlobalAveragePooling2D)                                        
                                                                 
 dense_9 (Dense)             (None, 128)               163968    
                                                                 
 dropout_7 (Dropout)         (None, 128)               0         
                                                                 
 dense_10 (Dense)            (None, 1)                 129       
                                                                 
=================================================================
Total params: 2,422,081
Trainable params: 164,097
Non-trainable params: 2,257,984
_________________________________________________________________
```

In [ ]:
```
#To split the data with 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(x_input, y_target, test_size=0.20, random_state=42)
```

In [ ]:
```
# Scale pixel values to range [0, 1]
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [ ]:
```
model_4.fit(X_train,
          y_train,
          validation_data=(X_test,y_test),
          epochs=30)
```

```
Epoch 1/30
139/139 [==============================] - 8s 41ms/step - loss: 0.5982 - a
ccuracy: 0.7000 - val_loss: 0.5330 - val_accuracy: 0.7459
Epoch 2/30
139/139 [==============================] - 5s 35ms/step - loss: 0.5050 - a
ccuracy: 0.7550 - val_loss: 0.5491 - val_accuracy: 0.7369
Epoch 3/30
139/139 [==============================] - 5s 36ms/step - loss: 0.4742 - a
ccuracy: 0.7807 - val_loss: 0.5440 - val_accuracy: 0.7523
Epoch 4/30
139/139 [==============================] - 5s 35ms/step - loss: 0.4337 - a
ccuracy: 0.7978 - val_loss: 0.5774 - val_accuracy: 0.7261
Epoch 5/30
139/139 [==============================] - 5s 35ms/step - loss: 0.4046 - a
ccuracy: 0.8213 - val_loss: 0.5602 - val_accuracy: 0.7432
Epoch 6/30
139/139 [==============================] - 5s 35ms/step - loss: 0.3659 - a
ccuracy: 0.8413 - val_loss: 0.5722 - val_accuracy: 0.7505
Epoch 7/30
139/139 [==============================] - 5s 35ms/step - loss: 0.3318 - a
ccuracy: 0.8582 - val_loss: 0.5793 - val_accuracy: 0.7468
Epoch 8/30
139/139 [==============================] - 5s 35ms/step - loss: 0.3072 - a
ccuracy: 0.8724 - val_loss: 0.6596 - val_accuracy: 0.7126
Epoch 9/30
139/139 [==============================] - 5s 36ms/step - loss: 0.2836 - a
ccuracy: 0.8862 - val_loss: 0.6177 - val_accuracy: 0.7360
Epoch 10/30
139/139 [==============================] - 5s 36ms/step - loss: 0.2515 - a
ccuracy: 0.9006 - val_loss: 0.6795 - val_accuracy: 0.7225
Epoch 11/30
139/139 [==============================] - 5s 36ms/step - loss: 0.2154 - a
ccuracy: 0.9171 - val_loss: 0.6497 - val_accuracy: 0.7342
Epoch 12/30
139/139 [==============================] - 5s 35ms/step - loss: 0.1929 - a
ccuracy: 0.9295 - val_loss: 0.6799 - val_accuracy: 0.7252
Epoch 13/30
139/139 [==============================] - 5s 35ms/step - loss: 0.1773 - a
ccuracy: 0.9358 - val_loss: 0.7317 - val_accuracy: 0.7225
Epoch 14/30
139/139 [==============================] - 5s 35ms/step - loss: 0.1494 - a
ccuracy: 0.9500 - val_loss: 0.7253 - val_accuracy: 0.7180
Epoch 15/30
139/139 [==============================] - 5s 36ms/step - loss: 0.1400 - a
ccuracy: 0.9506 - val_loss: 0.7539 - val_accuracy: 0.7324
Epoch 16/30
139/139 [==============================] - 5s 36ms/step - loss: 0.1103 - a
ccuracy: 0.9660 - val_loss: 0.8362 - val_accuracy: 0.7153
Epoch 17/30
139/139 [==============================] - 5s 35ms/step - loss: 0.1062 - a
ccuracy: 0.9657 - val_loss: 0.8000 - val_accuracy: 0.7027
Epoch 18/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0904 - a
ccuracy: 0.9748 - val_loss: 0.8659 - val_accuracy: 0.7198
Epoch 19/30
139/139 [==============================] - 5s 37ms/step - loss: 0.0782 - a
ccuracy: 0.9815 - val_loss: 0.9008 - val_accuracy: 0.7225
Epoch 20/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0691 - a
ccuracy: 0.9808 - val_loss: 0.9096 - val_accuracy: 0.7153
Epoch 21/30
```

```
139/139 [==============================] - 5s 36ms/step - loss: 0.0781 - a
ccuracy: 0.9775 - val_loss: 0.9057 - val_accuracy: 0.7297
Epoch 22/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0678 - a
ccuracy: 0.9802 - val_loss: 0.9616 - val_accuracy: 0.7090
Epoch 23/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0573 - a
ccuracy: 0.9835 - val_loss: 1.0194 - val_accuracy: 0.7216
Epoch 24/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0478 - a
ccuracy: 0.9881 - val_loss: 0.9938 - val_accuracy: 0.7297
Epoch 25/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0386 - a
ccuracy: 0.9930 - val_loss: 1.1058 - val_accuracy: 0.7063
Epoch 26/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0370 - a
ccuracy: 0.9908 - val_loss: 1.1053 - val_accuracy: 0.7117
Epoch 27/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0428 - a
ccuracy: 0.9878 - val_loss: 1.0463 - val_accuracy: 0.7252
Epoch 28/30
139/139 [==============================] - 5s 38ms/step - loss: 0.0417 - a
ccuracy: 0.9872 - val_loss: 1.1003 - val_accuracy: 0.7252
Epoch 29/30
139/139 [==============================] - 5s 35ms/step - loss: 0.0513 - a
ccuracy: 0.9844 - val_loss: 1.1826 - val_accuracy: 0.7234
Epoch 30/30
139/139 [==============================] - 5s 36ms/step - loss: 0.0478 - a
ccuracy: 0.9842 - val_loss: 1.1382 - val_accuracy: 0.7153
```

Out[ ]:
```
<keras.callbacks.History at 0x1b90e740508>
```

In [ ]:
```
#Instantiating the augmented images
datagen = ImageDataGenerator(
        rotation_range=50,
        width_shift_range=0.2,
        height_shift_range=0.3,
        vertical_flip=True,
        horizontal_flip=False,
        zoom_range=0.5,
        fill_mode='wrap')
```

In [ ]:
```
#Fitting with the augmented images
model_4.fit(datagen.flow(X_train, y_train, batch_size=64),
          validation_data=(X_test, y_test),
          epochs=25)
```

```
Epoch 1/25
70/70 [==============================] - 6s 71ms/step - loss: 0.7285 - acc
uracy: 0.6863 - val_loss: 0.6064 - val_accuracy: 0.6811
Epoch 2/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5744 - acc
uracy: 0.7081 - val_loss: 0.5748 - val_accuracy: 0.7261
Epoch 3/25
70/70 [==============================] - 5s 75ms/step - loss: 0.5688 - acc
uracy: 0.7192 - val_loss: 0.5620 - val_accuracy: 0.7216
Epoch 4/25
70/70 [==============================] - 5s 74ms/step - loss: 0.5531 - acc
uracy: 0.7235 - val_loss: 0.5463 - val_accuracy: 0.7405
Epoch 5/25
70/70 [==============================] - 5s 74ms/step - loss: 0.5586 - acc
uracy: 0.7176 - val_loss: 0.5531 - val_accuracy: 0.7261
Epoch 6/25
70/70 [==============================] - 5s 77ms/step - loss: 0.5551 - acc
uracy: 0.7230 - val_loss: 0.5578 - val_accuracy: 0.7162
Epoch 7/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5486 - acc
uracy: 0.7280 - val_loss: 0.5436 - val_accuracy: 0.7378
Epoch 8/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5481 - acc
uracy: 0.7262 - val_loss: 0.5537 - val_accuracy: 0.7387
Epoch 9/25
70/70 [==============================] - 5s 75ms/step - loss: 0.5434 - acc
uracy: 0.7223 - val_loss: 0.5451 - val_accuracy: 0.7378
Epoch 10/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5411 - acc
uracy: 0.7298 - val_loss: 0.5446 - val_accuracy: 0.7468
Epoch 11/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5398 - acc
uracy: 0.7325 - val_loss: 0.5391 - val_accuracy: 0.7414
Epoch 12/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5354 - acc
uracy: 0.7395 - val_loss: 0.5425 - val_accuracy: 0.7505
Epoch 13/25
70/70 [==============================] - 5s 74ms/step - loss: 0.5295 - acc
uracy: 0.7379 - val_loss: 0.5398 - val_accuracy: 0.7477
Epoch 14/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5396 - acc
uracy: 0.7352 - val_loss: 0.5440 - val_accuracy: 0.7351
Epoch 15/25
70/70 [==============================] - 5s 74ms/step - loss: 0.5304 - acc
uracy: 0.7437 - val_loss: 0.5345 - val_accuracy: 0.7450
Epoch 16/25
70/70 [==============================] - 5s 76ms/step - loss: 0.5268 - acc
uracy: 0.7433 - val_loss: 0.5334 - val_accuracy: 0.7450
Epoch 17/25
70/70 [==============================] - 5s 76ms/step - loss: 0.5318 - acc
uracy: 0.7422 - val_loss: 0.5347 - val_accuracy: 0.7450
Epoch 18/25
70/70 [==============================] - 6s 81ms/step - loss: 0.5202 - acc
uracy: 0.7516 - val_loss: 0.5392 - val_accuracy: 0.7432
Epoch 19/25
70/70 [==============================] - 6s 80ms/step - loss: 0.5289 - acc
uracy: 0.7510 - val_loss: 0.5344 - val_accuracy: 0.7441
Epoch 20/25
70/70 [==============================] - 6s 79ms/step - loss: 0.5316 - acc
uracy: 0.7431 - val_loss: 0.5343 - val_accuracy: 0.7450
Epoch 21/25
```

```
70/70 [==============================] - 6s 80ms/step - loss: 0.5247 - acc
uracy: 0.7401 - val_loss: 0.5286 - val_accuracy: 0.7514
Epoch 22/25
70/70 [==============================] - 5s 74ms/step - loss: 0.5248 - acc
uracy: 0.7485 - val_loss: 0.5430 - val_accuracy: 0.7414
Epoch 23/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5288 - acc
uracy: 0.7395 - val_loss: 0.5388 - val_accuracy: 0.7477
Epoch 24/25
70/70 [==============================] - 5s 73ms/step - loss: 0.5257 - acc
uracy: 0.7350 - val_loss: 0.5420 - val_accuracy: 0.7369
Epoch 25/25
70/70 [==============================] - 5s 74ms/step - loss: 0.5262 - acc
uracy: 0.7437 - val_loss: 0.5344 - val_accuracy: 0.7387
```

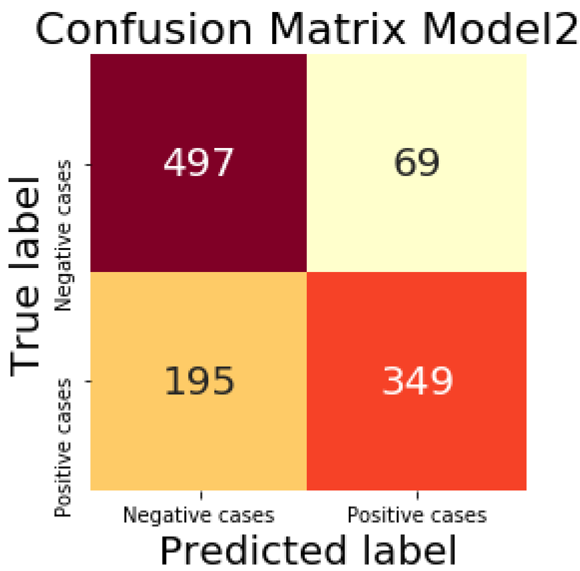
Out[ ]:
```
<keras.callbacks.History at 0x1b91e9c30c8>
```

In [ ]:
```
#To evaluate model_4 accuracy:
score = model_4.evaluate(X_test, y_test, verbose=0)
print('Model_4: MobileNetV2 accuracy:', score[1], '\n')

preds = best_model.predict(X_test)
preds1 = np.round(preds)
print(preds.shape)
print(preds.shape)
print(y_test.shape)
print(classification_report(y_test, preds1))
```

```
Model_4: MobileNetV2 accuracy: 0.7387387156486511

35/35 [==============================] - 0s 4ms/step
(1110, 1)
(1110, 1)
(1110,)
              precision    recall  f1-score   support

           0       0.72      0.88      0.79       566
           1       0.83      0.64      0.73       544

    accuracy                           0.76      1110
   macro avg       0.78      0.76      0.76      1110
weighted avg       0.78      0.76      0.76      1110
```

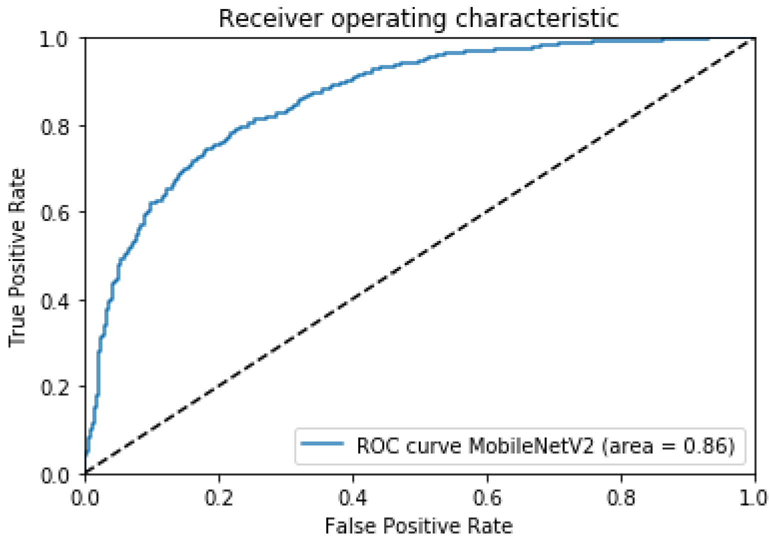
In [ ]:

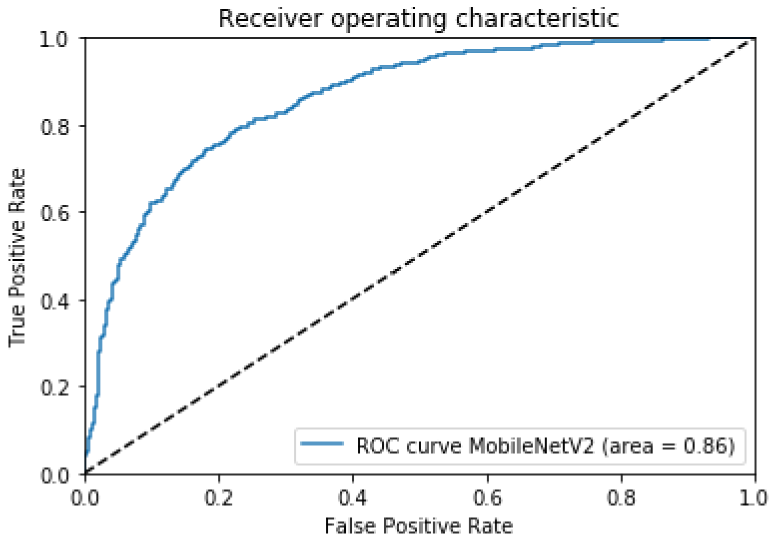
```
# To plot the confusion matrix
mat = confusion_matrix(y_test, preds1)

sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Negative cases', 'Positive cases'],
            yticklabels=['Negative cases', 'Positive cases'],
            cmap='YlOrRd', annot_kws={"fontsize": 20})

plt.xlabel('Predicted label', fontsize=20)
plt.ylabel('True label', fontsize=20)
plt.title('Confusion Matrix Model2', fontsize=22)

plt.show()
```



In [ ]:
```
# To plot the ROC curve
from sklearn.metrics import roc_curve, auc

y_score = model_4.predict(X_test) # get the prediction probabilities

fpr4 = dict()
tpr4 = dict()
roc_auc4 = dict()
for i in range(num_classes):
    fpr4[i], tpr4[i], _ = roc_curve(y_test, preds)
    roc_auc4[i] = auc(fpr4[i], tpr4[i])
# Plot of a ROC curve for a specific class
for i in range(num_classes):
    plt.figure()
    plt.plot(fpr4[i], tpr4[i], label='ROC curve MobileNetV2 (area = %0.2f)'
% roc_auc4[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0, 1.0])
    plt.ylim([0, 1.00]) # set limit for y axis
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show();
```

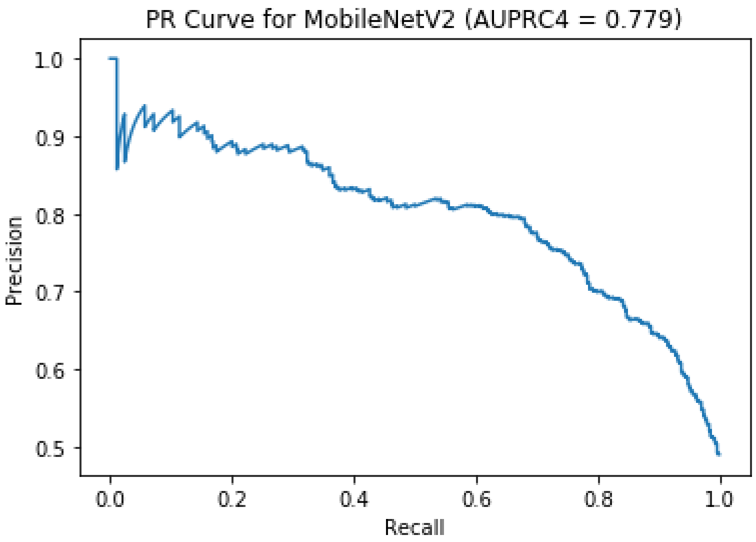
```
35/35 [==============================] - 1s 28ms/step
```





In [ ]:
```
#To plot the PR curve
# Determine the probabilities for positive class
y_scores = model_4.predict(X_test)
precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)

# To calculate area under the PRC
auprc4 = average_precision_score(y_test, y_scores)
print('AUPRC4:', auprc)

# Plot the PR curve
plt.plot(recalls, precisions)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PR Curve for MobileNetV2 (AUPRC4 = {:.3f})'.format(auprc))
plt.show()
```

```
35/35 [==============================] - 1s 28ms/step
AUPRC4: 0.7793772110950611
```



MODEL 5 MULTILAYER PERCEPTRON

In [ ]:
```
#To split the data with 75% for training and 25% for testing
X_train, X_test, y_train, y_test = train_test_split(x_input, y_target, test
_size=0.25, random_state=0)
```

In [ ]:
```
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(4160, 7500)
```

In [ ]:
```
# create model
model_5 = Sequential()
model_5.add(Dense(12, input_dim=7500, activation='relu')) #
model_5.add(Dropout(0.2))
model_5.add(Dense(8, activation='relu'))
model_5.add(Dropout(0.2))
model_5.add(Dense(1, activation='sigmoid'))
```

In [ ]:
```
model_5.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_14 (Dense)            (None, 12)                90012     
                                                                 
 dropout_10 (Dropout)        (None, 12)                0         
                                                                 
 dense_15 (Dense)            (None, 8)                 104       
                                                                 
 dropout_11 (Dropout)        (None, 8)                 0         
                                                                 
 dense_16 (Dense)            (None, 1)                 9         
                                                                 
 dropout_12 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_13 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_14 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_15 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_16 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_17 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_18 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_19 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_20 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_21 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_22 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_23 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_24 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_25 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_26 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_27 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_28 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_29 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_30 (Dropout)        (None, 1)                 0         
                                                                 
 dropout_31 (Dropout)        (None, 1)                 0         
                                                                 
=================================================================
Total params: 90,125
Trainable params: 90,125
Non-trainable params: 0
_________________________________________________________________
```

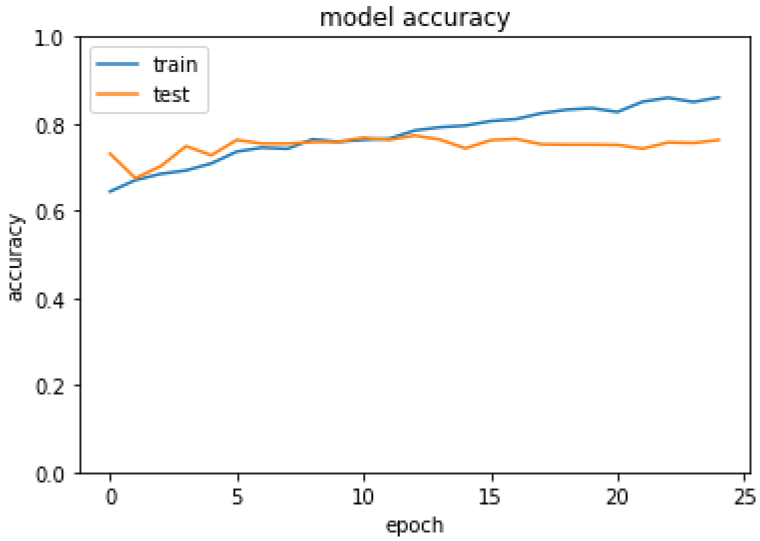
In [ ]:
```
# Compile model
model_5.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc
uracy'])
```

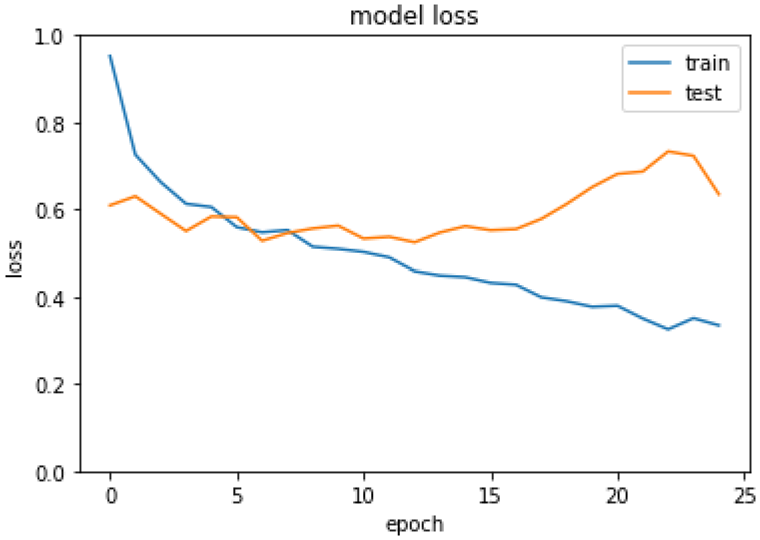
In [ ]:
```
history = model_5.fit(X_train,
          y_train,
          validation_data=(X_test,y_test),
          epochs=25)
```

```
Epoch 1/25
130/130 [==============================] - 1s 3ms/step - loss: 0.9522 - ac
curacy: 0.6440 - val_loss: 0.6092 - val_accuracy: 0.7311
Epoch 2/25
130/130 [==============================] - 0s 2ms/step - loss: 0.7257 - ac
curacy: 0.6702 - val_loss: 0.6301 - val_accuracy: 0.6748
Epoch 3/25
130/130 [==============================] - 0s 2ms/step - loss: 0.6628 - ac
curacy: 0.6846 - val_loss: 0.5900 - val_accuracy: 0.7022
Epoch 4/25
130/130 [==============================] - 0s 2ms/step - loss: 0.6129 - ac
curacy: 0.6925 - val_loss: 0.5503 - val_accuracy: 0.7484
Epoch 5/25
130/130 [==============================] - 0s 2ms/step - loss: 0.6054 - ac
curacy: 0.7087 - val_loss: 0.5837 - val_accuracy: 0.7275
Epoch 6/25
130/130 [==============================] - 0s 2ms/step - loss: 0.5593 - ac
curacy: 0.7358 - val_loss: 0.5820 - val_accuracy: 0.7628
Epoch 7/25
130/130 [==============================] - 0s 2ms/step - loss: 0.5473 - ac
curacy: 0.7450 - val_loss: 0.5278 - val_accuracy: 0.7541
Epoch 8/25
130/130 [==============================] - 0s 2ms/step - loss: 0.5525 - ac
curacy: 0.7423 - val_loss: 0.5457 - val_accuracy: 0.7541
Epoch 9/25
130/130 [==============================] - 0s 2ms/step - loss: 0.5143 - ac
curacy: 0.7642 - val_loss: 0.5562 - val_accuracy: 0.7578
Epoch 10/25
130/130 [==============================] - 0s 2ms/step - loss: 0.5094 - ac
curacy: 0.7584 - val_loss: 0.5627 - val_accuracy: 0.7585
Epoch 11/25
130/130 [==============================] - 0s 2ms/step - loss: 0.5025 - ac
curacy: 0.7630 - val_loss: 0.5330 - val_accuracy: 0.7678
Epoch 12/25
130/130 [==============================] - 0s 2ms/step - loss: 0.4899 - ac
curacy: 0.7656 - val_loss: 0.5371 - val_accuracy: 0.7628
Epoch 13/25
130/130 [==============================] - 0s 2ms/step - loss: 0.4585 - ac
curacy: 0.7844 - val_loss: 0.5245 - val_accuracy: 0.7729
Epoch 14/25
130/130 [==============================] - 0s 2ms/step - loss: 0.4488 - ac
curacy: 0.7918 - val_loss: 0.5469 - val_accuracy: 0.7635
Epoch 15/25
130/130 [==============================] - 0s 2ms/step - loss: 0.4452 - ac
curacy: 0.7957 - val_loss: 0.5616 - val_accuracy: 0.7433
Epoch 16/25
130/130 [==============================] - 0s 2ms/step - loss: 0.4321 - ac
curacy: 0.8060 - val_loss: 0.5522 - val_accuracy: 0.7621
Epoch 17/25
130/130 [==============================] - 0s 2ms/step - loss: 0.4279 - ac
curacy: 0.8106 - val_loss: 0.5548 - val_accuracy: 0.7650
Epoch 18/25
130/130 [==============================] - 0s 2ms/step - loss: 0.3991 - ac
curacy: 0.8240 - val_loss: 0.5783 - val_accuracy: 0.7527
Epoch 19/25
130/130 [==============================] - 0s 2ms/step - loss: 0.3900 - ac
curacy: 0.8322 - val_loss: 0.6125 - val_accuracy: 0.7520
Epoch 20/25
130/130 [==============================] - 0s 2ms/step - loss: 0.3774 - ac
curacy: 0.8358 - val_loss: 0.6511 - val_accuracy: 0.7520
Epoch 21/25
```

```
130/130 [==============================] - 0s 2ms/step - loss: 0.3797 - ac
curacy: 0.8269 - val_loss: 0.6815 - val_accuracy: 0.7513
Epoch 22/25
130/130 [==============================] - 0s 2ms/step - loss: 0.3504 - ac
curacy: 0.8507 - val_loss: 0.6870 - val_accuracy: 0.7426
Epoch 23/25
130/130 [==============================] - 0s 2ms/step - loss: 0.3256 - ac
curacy: 0.8596 - val_loss: 0.7331 - val_accuracy: 0.7570
Epoch 24/25
130/130 [==============================] - 0s 2ms/step - loss: 0.3512 - ac
curacy: 0.8500 - val_loss: 0.7231 - val_accuracy: 0.7556
Epoch 25/25
130/130 [==============================] - 0s 2ms/step - loss: 0.3347 - ac
curacy: 0.8603 - val_loss: 0.6347 - val_accuracy: 0.7628
```

In [ ]:
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim([0,1])
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.ylim([0,1])
plt.show()
```

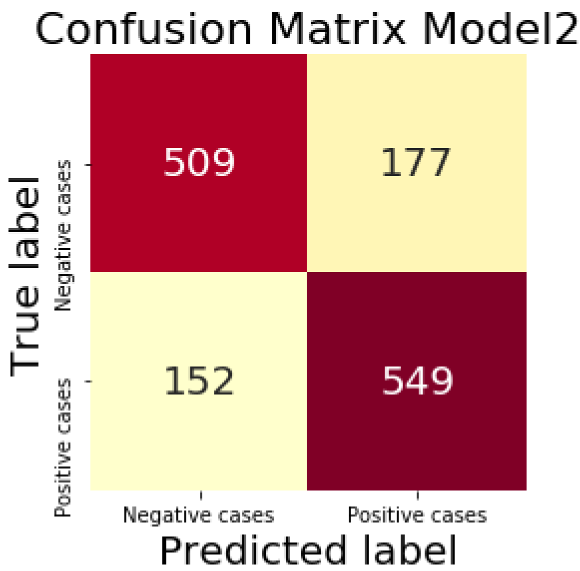




In [ ]:
```
#To evaluate the model accuracy:
score = model_5.evaluate(X_test, y_test, verbose=1)
print('model_5: CNN accuracy:', score[1], '\n')

preds = model_5.predict(X_test)
print(preds.shape) # which means the predictions return in one-hot encoding format
preds1 = np.round(preds)
print(preds1.shape)
print(y_test.shape)
print(classification_report(y_test, preds1))
```

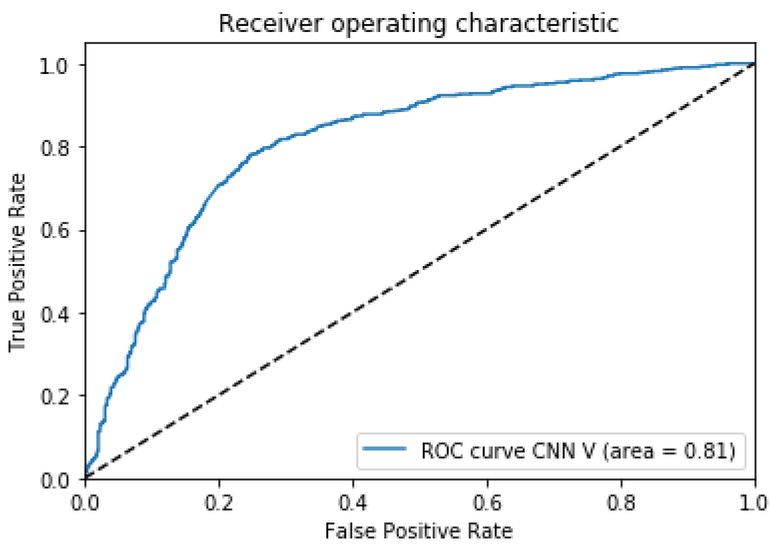
```
44/44 [==============================] - 0s 1ms/step - loss: 0.6347 - accu
racy: 0.7628
model_5: CNN accuracy: 0.7627974152565002

44/44 [==============================] - 0s 951us/step
(1387, 1)
(1387, 1)
(1387,)
              precision    recall  f1-score   support

           0       0.77      0.74      0.76       686
           1       0.76      0.78      0.77       701

    accuracy                           0.76      1387
   macro avg       0.76      0.76      0.76      1387
weighted avg       0.76      0.76      0.76      1387
```

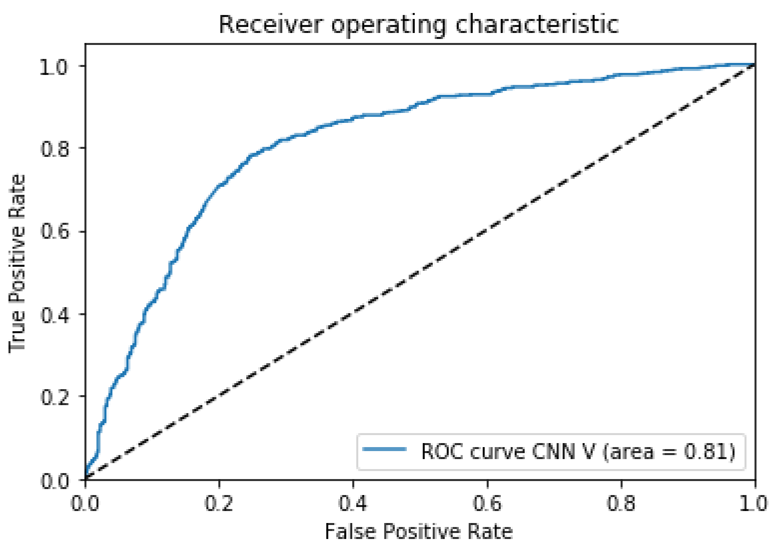
In [ ]:
```
# To plot the confusion matrix
mat = confusion_matrix(y_test, preds1)

sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Negative cases', 'Positive cases'],
            yticklabels=['Negative cases', 'Positive cases'],
            cmap='YlOrRd', annot_kws={"fontsize": 20})

plt.xlabel('Predicted label', fontsize=20)
plt.ylabel('True label', fontsize=20)
plt.title('Confusion Matrix Model2', fontsize=22)

plt.show()
```



In [ ]:
```
# To plot the ROC

y_score = model_5.predict(X_test) # get the prediction probabilities

fpr5 = dict()
tpr5 = dict()
roc_auc5 = dict()
for i in range(num_classes):
    fpr5[i], tpr5[i], _ = roc_curve(y_test, preds)
    roc_auc5[i] = auc(fpr5[i], tpr5[i])
# Plot of a ROC curve for a specific class
for i in range(num_classes):
    plt.figure()
    plt.plot(fpr5[i], tpr5[i],
             label='ROC curve CNN V (area = %0.2f)' % roc_auc5[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show();
```

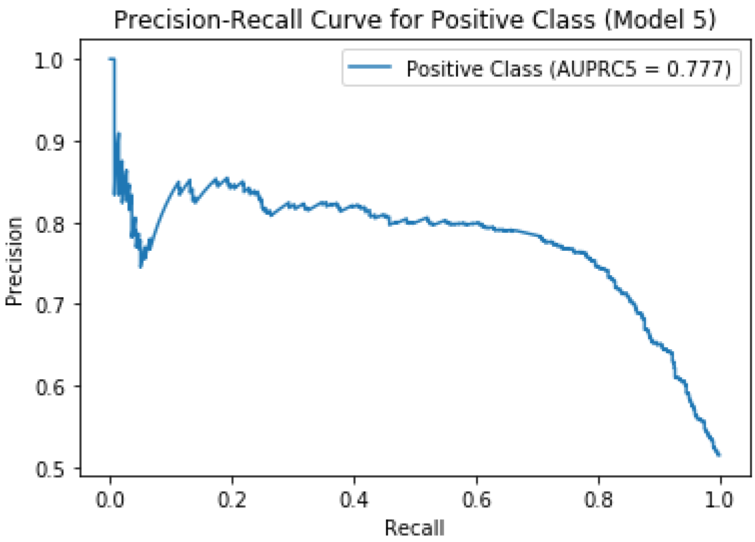
```
44/44 [==============================] - 0s 905us/step
```





In [ ]:
```
y_scores = model_5.predict(X_test)
y_test = y_test.reshape((-1, 1))
precisions_pos, recalls_pos, thresholds_pos = precision_recall_curve(y_tes
t, y_scores)
auprc5b = average_precision_score(y_test, y_scores)
print('AUPRC for Positive Class:', auprc2b)
plt.plot(recalls_pos, precisions_pos, label='Positive Class (AUPRC5 = {:.3
f})'.format(auprc5b))
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Positive Class (Model 5)')
plt.legend()
plt.show()
```

```
44/44 [==============================] - 0s 951us/step
AUPRC for Positive Class: 0.7908419293396383
```



In [ ]:
```
# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed) # n_s
plits: num of folds
cvscores = [] # define this empty list to add the scores in
for train, test in kfold.split(X_train, y_train):
  # create model
    model = Sequential()
    model.add(Dense(12, input_dim=7500, activation='relu'))
    model_5.add(Dropout(0.2))
    model.add(Dense(8, activation='relu'))
    model_5.add(Dropout(0.2))
    model.add(Dense(2, activation='sigmoid'))
    # Compile model
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    # Fit the model
    model.fit(X_train[train], y_train[train], epochs=150, batch_size=10, ve
rbose=0)
    # evaluate the model
    scores = model.evaluate(X_test, y_test, verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

```
accuracy: 71.59%
accuracy: 71.16%
accuracy: 70.51%
accuracy: 75.34%
accuracy: 71.74%
accuracy: 71.88%
accuracy: 72.89%
accuracy: 70.80%
accuracy: 70.73%
accuracy: 72.31%
71.90% (+/- 1.35%)
```

MODEL 6: Support Vector Machines

In [ ]:
```
# To split the data with 75% for training and 25% for testing
X_train, X_test, y_train, y_test = train_test_split(x_input, y_target, test
_size=0.25, random_state=0)
```

In [ ]:
```
#Reshaping the 4D  TO 2D to suit the classical model
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
```

In [ ]:
```
#This prints the shape of the train and test set
X_train.shape, X_test.shape
```

Out[ ]:
```
((4160, 7500), (1387, 7500))
```

In [ ]:
```
X_train[20]
```

Out[ ]:
```
array([0.91505801, 1.30544227, 1.28541194, ..., 0.81530901, 0.88100434,
       0.89142242])
```

In [ ]:
```
# train the SVM classifier:

model = SVC(kernel='rbf')
model.fit(X_train, y_train)
```

Out[ ]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [ ]:
```
# This predict the test set labels
y_preds = clf.predict(X_test)

# Print the classification report
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.71      0.84      0.77       686
           1       0.81      0.67      0.73       701

    accuracy                           0.75      1387
   macro avg       0.76      0.75      0.75      1387
weighted avg       0.76      0.75      0.75      1387
```

In [ ]:
```
#To set the hyperparameter grid
grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [0.1, 0.01, 0.001, 0.0001],
              'kernel': ['linear', 'rbf','poly']}
```

In [ ]:
```
random_search = RandomizedSearchCV(clf, param_distributions=grid, n_iter=5)
```

In [ ]:
```
random_search.fit(X_train, y_train)
```

Out[ ]:
```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                                 class_weight=None, coef0=0.0,
                                 decision_function_shape='ovr', degree=3,
                                 gamma='scale', kernel='linear', max_iter=
-1,
                                 probability=False, random_state=None,
                                 shrinking=True, tol=0.001, verbose=Fals
e),
                   iid='deprecated', n_iter=5, n_jobs=None,
                   param_distributions={'C': [0.1, 1, 10, 100, 1000],
                                        'gamma': [0.1, 0.01, 0.001, 0.000
1],
                                        'kernel': ['linear', 'rbf', 'pol
y']},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [ ]:
```
print("Best hyperparameters: ", random_search.best_params_)
print("Best score: ", random_search.best_score_)
```

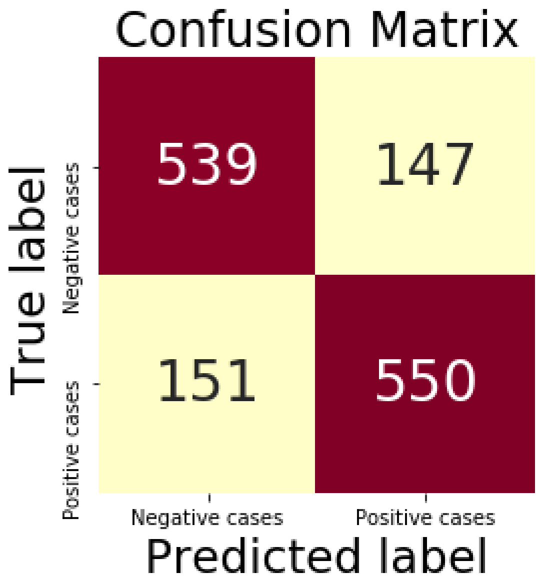
```
Best hyperparameters:  {'kernel': 'rbf', 'gamma': 0.0001, 'C': 1}
Best score:  0.7663461538461538
```

In [ ]:
```
#To choose the best tuned performance
best_svm= svm.SVC(kernel=random_search.best_params_['kernel'],
                   C=random_search.best_params_['C'],
                   gamma=random_search.best_params_['gamma'])
best_svm.fit(X_train, y_train)
y_preds = best_svm.predict(X_test)
```

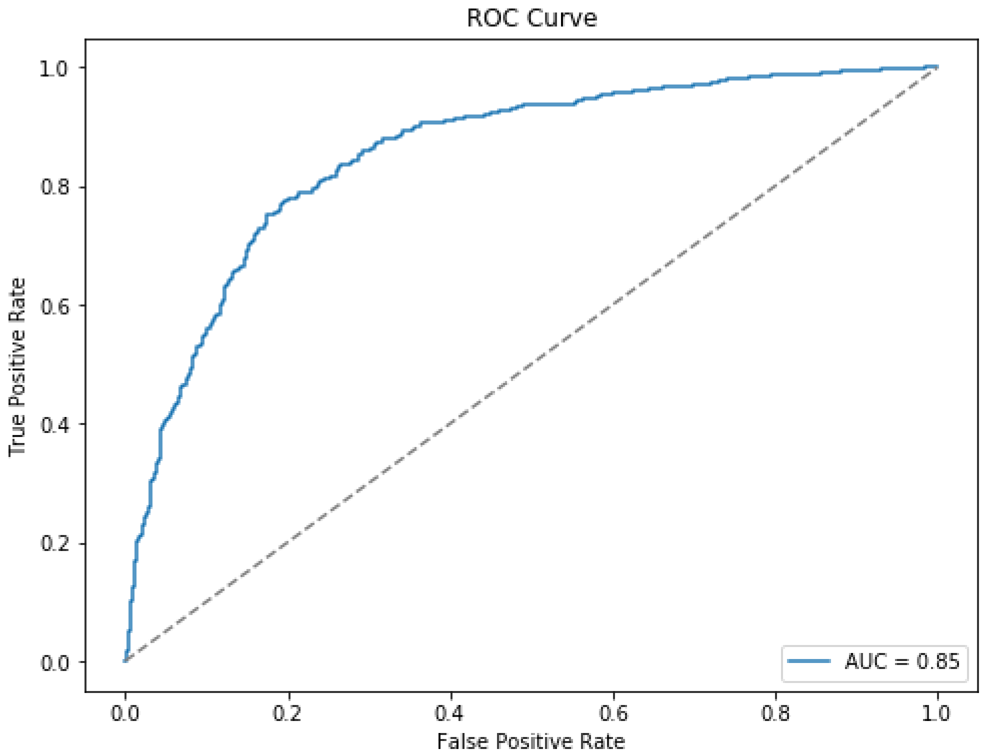
In [ ]:
```
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.78      0.79      0.78       686
           1       0.79      0.78      0.79       701

    accuracy                           0.79      1387
   macro avg       0.79      0.79      0.79      1387
weighted avg       0.79      0.79      0.79      1387
```

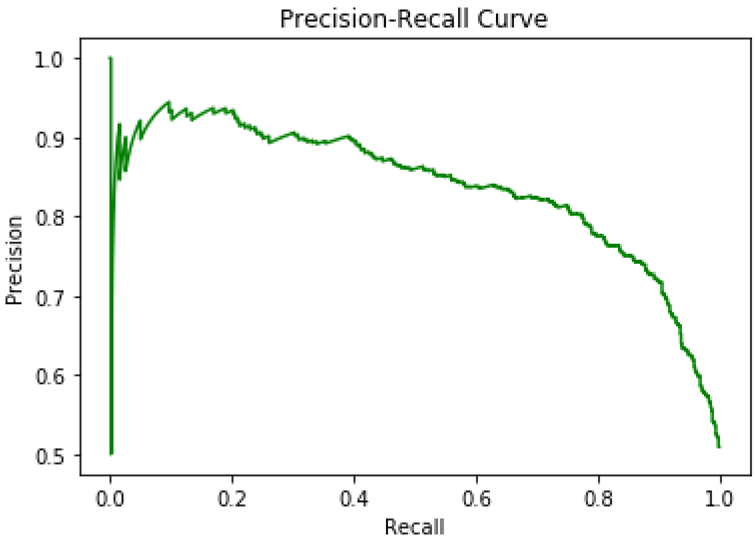
In [ ]:
```
# To plot the confusion matrix
mat = confusion_matrix(y_test, y_preds)

sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['Negative cases', 'Positive cases'],
            yticklabels=['Negative cases', 'Positive cases'],
            cmap='YlOrRd', annot_kws={"fontsize": 28})

plt.xlabel('Predicted label', fontsize=23)
plt.ylabel('True label', fontsize=23)
plt.title('Confusion Matrix', fontsize=24)

plt.show()
```



In [ ]:
```
# To predict probabilities for the test set
y_pred_prob = best_svm.decision_function(X_test)

# To calculate the ROC curve and the AUC score
fpr6, tpr6, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = roc_auc_score(y_test, y_pred_prob)

# To plot the ROC curve
plt.figure(figsize=(8,6))
plt.plot(fpr6, tpr6, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey')
plt.xlabel('False Positive')
plt.ylabel('True Positive')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



In [ ]:
```
# To get the precision and recall values for the test set
precision, recall, _ = precision_recall_curve(y_test, best_clf.decision_function(X_test))

# To plot the PR curve
plt.plot(recall, precision, color='g')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```



Discussion and Evaluation

The summary of the performance of the six models are shown in the table below:
image.png

The performance of the models was tested on the following metric:

Accuracy: This is the proportion or ratio in percentage of the true positive results of cancer cases (true positive and true negative inclusive). Model 3 (CNN3) is apparently the best performing model of the 6 models, with a test accuracy of 81% and a training accuracy of 80%. SVM model came second with a test accuracy of 79%.

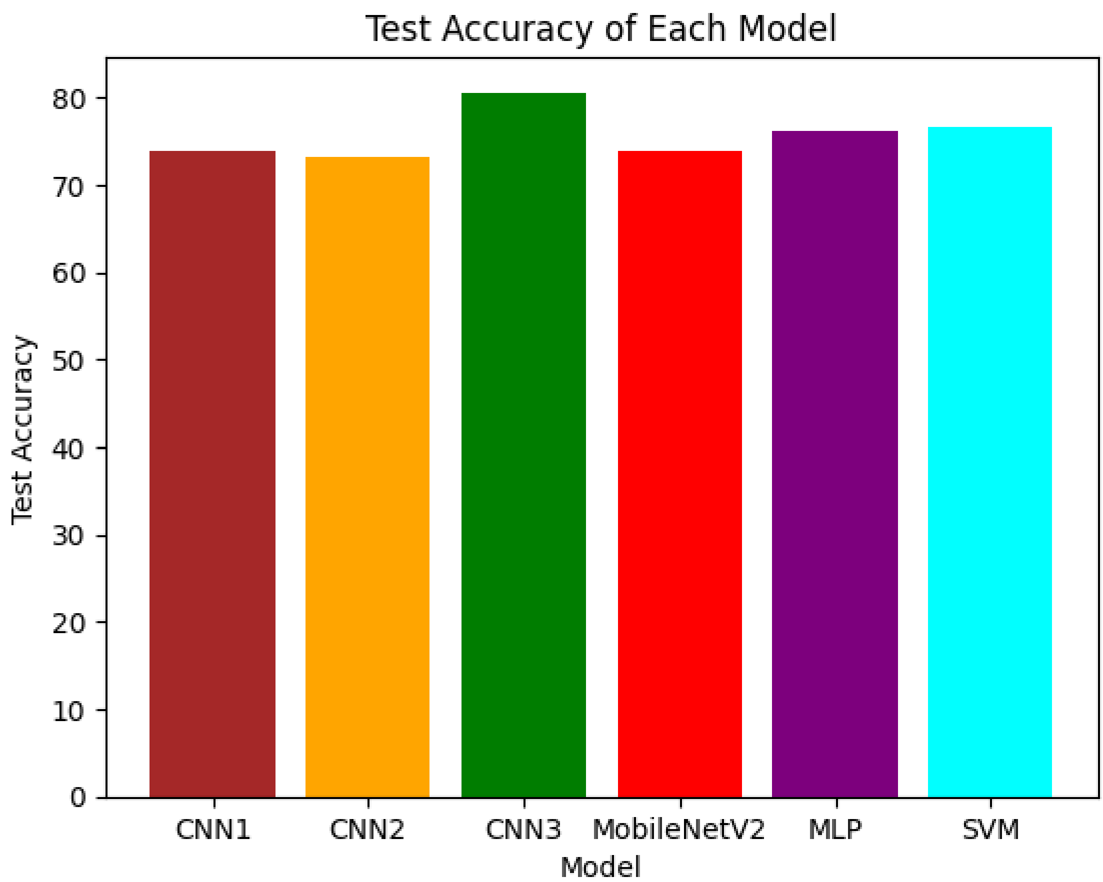
Precision: Precision is the characteristic of the positive precision made by the model. It is mathematically the number of the true positive result divided by the total number of positive predictions (true positives and false positives). The pretrained model (MobileNetV2) had the highest precision score.

Recall: This is the percentage of correct positive results made from all the positive predictions that could have been made. It is mathematically the ratio of the true positive and the sum of the true positives and the false negatives.

F1 Score: It is a machine learning evaluation metric that combines the precision and the recall score of an algorithm. CNN3 has the most impressive F1 score.

Area under the curve: The AUC is the measure of the ability of a model to distinguish between the classes. CNN3 has the most impressive area under the curve score for both ROC and the PRC

In [1]:
```
#To plot the Test Accuracy of each model
import matplotlib.pyplot as plt

models = ['CNN1', 'CNN2', 'CNN3', 'MobileNetV2', 'MLP', 'SVM']
test_accuracy = [73.9, 73.15, 80.63, 73.87, 76.28, 76.63]
colors = ['brown', 'orange',
          'green', 'red', 'purple', 'cyan'] #Defining the colours

plt.bar(models, test_accuracy, color=colors)
plt.xlabel('Model')
plt.ylabel('Test Accuracy')
plt.title('Test Accuracy of Each Model');
```



Conclusion:

This study explored six machine learning models consisting of variants of CNN models, MobileNetV2 which is a based on a pretrained architecture, in the classification of breast cancer. Of all the models CNN(CNN3) exhibited the most promising and impressive performance which is inline with previous research. CNN3 achieved a relatively high test accuracy of 81% and also performed well across all the metrics which it is evaluated on better than the rest of the models. MobileNetV2 AND SVM also had considerable good performance. This study will be of great impact in the field of medicine and will help in the early and accurate detection off breast cancer. There are however some areas of improvement which could help these models perform much better. This include the tuning of more hyperparameters, more augmentation of data. There is also a possibility of high performance with availability of more training data

References

Image data types and what they mean — skimage 0.21.0rc1.dev0 documentation (no date). Available at: https://scikit-image.org/docs/dev/user_guide/data_types.html (https://scikit-image.org/docs/dev/user_guide/data_types.html).

Alkhatib, M.Q., Al-Saad, M., Aburaed, N., Almansoori, S., Zabalza, J., Marshall, S. and Al-Ahmad, H., 2023. Tri-CNN: a three branch model for hyperspectral image classification. Remote Sensing, 15(2), p.316.

Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S. and Miao, Y., 2021. Review of image classification algorithms based on convolutional neural networks. Remote Sensing, 13(22), p.4712.

Hafiz, A.M., Bhat, R.A. and Hassaballah, M., 2023. Image classification using convolutional neural network tree ensembles. Multimedia Tools and Applications, 82(5), pp.6867-6884.

Jin, K. (2022). Handwritten digit recognition based on classical machine learning methods. 2022 3rd International Conference on Electronic Communication and Artificial Intelligence (IWECAI).

Neural Network Diagram Complete Guide | EdrawMax (no date). Available at: https://www.edrawsoft.com/article/neural-network-diagram.html (https://www.edrawsoft.com/article/neural-network-diagram.html).

Wongsuphasawat, K. et al. (2018) “Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow,” IEEE Transactions on Visualization and Computer Graphics, 24(1), pp. 1–12. Available at: https://doi.org/10.1109/tvcg.2017.2744878 (https://doi.org/10.1109/tvcg.2017.2744878).

Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K., 2018. Convolutional neural networks: an overview and application in radiology. Insights into imaging, 9, pp.611-629.

In [ ]: