In [1]:

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [2]:

```
import cv2
import numpy as np

def find_largest_contour(image):
    """
    This function finds all the contours in an image and return the largest
    contour area.
    :param image: a binary image
    """
    image = image.astype(np.uint8)
    contours, hierarchy = cv2.findContours(
        image,
        cv2.RETR_TREE,
        cv2.CHAIN_APPROX_SIMPLE
    )
    largest_contour = max(contours, key=cv2.contourArea)
    return largest_contour

def show(name, image):
    """
    A simple function to visualize OpenCV images on screen.
    :param name: a string signifying the imshow() window name
    :param image: NumPy image to show
    """
    cv2.imshow(name, image)
    cv2.waitKey(0)

def apply_new_background(mask3d, foreground, save_name):
    """
    This function applies a new background to the extracted foreground image
    if `--new-background` flag is `True` while executing the file.
    :param mask3d: mask3d mask containing the foreground binary pixels
    :param foreground: mask containg the extracted foreground image
    :param save_name: name of the input image file
    """
    # normalization of mask3d mask, keeping values between 0 and 1
    mask3d = mask3d / 255.0
    # get the scaled product by multiplying
    foreground = cv2.multiply(mask3d, foreground)
    # read the new background image
    background = cv2.imread('input/background.jpg')
    # resize it according to the foreground image
    background = cv2.resize(background, (foreground.shape[1], foreground.shape[0]))
    background = background.astype(np.float)
    # get the scaled product by multiplying
    background = cv2.multiply(1.0 - mask3d, background)
    # add the foreground and new background image
    new_image = cv2.add(foreground, background)
    show('New image', new_image.astype(np.uint8))
    cv2.imwrite(f"outputs/{save_name}_new_background.jpg", new_image)
```

In [3]:

```
import numpy as np
import cv2
import argparse
from utils import*
#from utils import show
#from utils import apply_new_background
#from utils import find_largest_contour
 #define the argument parser
parser = argparse.ArgumentParser()
parser.add_argument('-i', '--input', help='path to the input image',required=True)
parser.add_argument('-n', '--new-background', dest='new_background',action='store_true'
args = vars(parser.parse_args())

image = cv2.imread('Users/karth/Desktop/Data/original/damaged/side/0101069901524_side.p
# show('Input image', image)
# blur the image to smmooth out the edges a bit, also reduces a bit of noise
blurred = cv2.GaussianBlur(image, (5, 5), 0)
# convert the image to grayscale
gray = cv2.cvtColor(blurred, cv2.COLOR_BGR2GRAY)
# apply thresholding to conver the image to binary format
# after this operation all the pixels below 200 value will be 0...
# and all th pixels above 200 will be 255
ret, gray = cv2.threshold(gray, 200 , 255, cv2.CHAIN_APPROX_NONE)

# find the largest contour area in the image
contour = find_largest_contour(gray)
image_contour = np.copy(image)
cv2.drawContours(image_contour, [contour], 0, (0, 255, 0), 2, cv2.LINE_AA, maxLevel=1)
show('Contour', image_contour)

# create a black `mask` the same size as the original grayscale image
mask = np.zeros_like(gray)
# fill the new mask with the shape of the largest contour
# all the pixels inside that area will be white
cv2.fillPoly(mask, [contour], 255)
# create a copy of the current mask
res_mask = np.copy(mask)
res_mask[mask == 0] = cv2.GC_BGD # obvious background pixels
res_mask[mask == 255] = cv2.GC_PR_BGD # probable background pixels
res_mask[mask == 255] = cv2.GC_FGD # obvious foreground pixels

# create a mask for obvious and probable foreground pixels
# all the obvious foreground pixels will be white and...
# ... all the probable foreground pixels will be black
mask2 = np.where(
    (res_mask == cv2.GC_FGD) | (res_mask == cv2.GC_PR_FGD),255,0).astype('uint8')

# create `new_mask3d` from `mask2` but with 3 dimensions instead of 2
new_mask3d = np.repeat(mask2[:, :, np.newaxis], 3, axis=2)
mask3d = new_mask3d
mask3d[new_mask3d > 0] = 255.0
mask3d[mask3d > 255] = 255.0
# apply Gaussian blurring to smoothen out the edges a bit
# `mask3d` is the final foreground mask (not extracted foreground image)
mask3d = cv2.GaussianBlur(mask3d, (5, 5), 0)
show('Foreground mask', mask3d)

# create the foreground image by zeroing out the pixels where `mask2`...
```

```
# ... has black pixels
foreground = np.copy(image).astype(float)
foreground[mask2 == 0] = 0
show('Foreground', foreground.astype(np.uint8))

# save the images to disk
save_name = args['input'].split('/')[-1].split('.')[0]
cv2.imwrite(f"outputs/{save_name}_foreground.png", foreground)
cv2.imwrite(f"outputs/{save_name}_foreground_mask.png", mask3d)
cv2.imwrite(f"outputs/{save_name}_contour.png", image_contour)

# the `--new-background` flag is `True`, then apply the new background...
# ... to the extracted foreground image
if args['new_background']:
    apply_new_background(mask3d, foreground, save_name)
```

```
usage: ipykernel_launcher.py [-h] -i INPUT [-n]
ipykernel_launcher.py: error: the following arguments are required: -i/--input
An exception has occurred, use %tb to see the full traceback.

SystemExit: 2

C:\Users\karth\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3452: UserWa
rning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

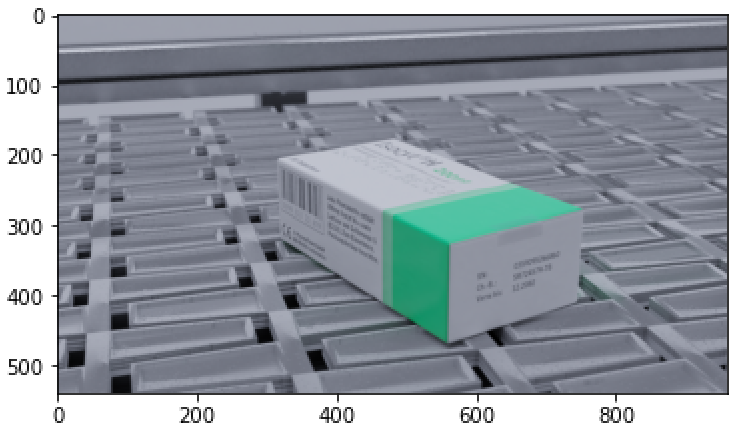
In [44]:

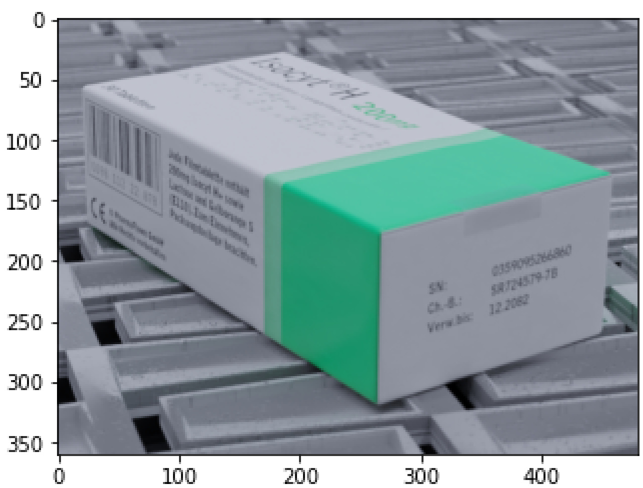
```
import numpy as np
import cv
import time
import os
from IPython.display import clear_output
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib import colors
from utils import *
#from utils import show
#from utils import apply_new_background
#from utils import find_largest_contour
# Crop img
filename = '0359095266860_side'
filename = filename + '.png'
img_path = '/Users/karth/Desktop/Data/original/intact/side/' + filename
img= cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
x = 295; y = 150
height= 360; width = 480
crop_img = img[y:y+height, x: x+width]
plt.imshow(crop_img)
plt.show()
path = '/Users/karth/Desktop/Data/Processed/side_only/intact/' + filename
cv2.imwrite(path, cv2.cvtColor(crop_img, cv2.COLOR_RGB2BGR))

img_path = '/Users/karth/Desktop/Data/Processed/side_only/damaged/0122691608037_side.pn
img= cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
```

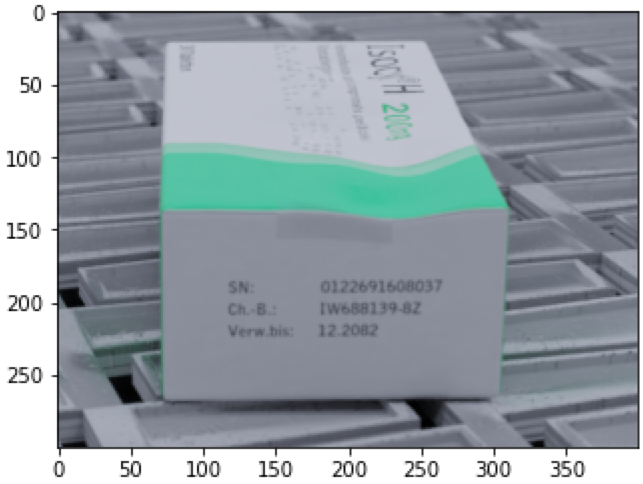
```
blurred = cv2.GaussianBlur(img, (5, 5), 0)
gray = cv2.cvtColor(blurred, cv2.COLOR_BGR2GRAY)
ret, gray = cv2.threshold(gray, 200 , 255, cv2.CHAIN_APPROX_NONE)
#contour = find_largest_contour(gray)
image_contour = np.copy(img)
#cv2.drawContours(image_contour, [contour], 0, (0, 255, 0), 2, cv2.LINE_AA, maxLevel=1)
#show('Contour', image_contour)

mask = np.zeros(img.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
rect = (0,0,300,400)
cv2.grabCut(img,mask,rect,bgdModel,fgdModel,10,cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = img*mask2[:,:,np.newaxis]
plt.imshow(img),plt.colorbar(),plt.show()
path='/Users/karth/Desktop/Data/data_sets/Processed/' + 'side_only/damaged/' + filename
cv2.imwrite(path, img)

r, g, b = cv2.split(img)
fig = plt.figure()
axis = fig.add_subplot(1, 1, 1, projection="3d")
pixel_colors = img.reshape((np.shape(img)[0]*np.shape(img)[1], 3))
norm = colors.Normalize(vmin=-1.,vmax=1.)
norm.autoscale(pixel_colors)
pixel_colors = norm(pixel_colors).tolist()
#
hsv_img = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
h, s, v = cv2.split(hsv_img)
fig = plt.figure()
axis = fig.add_subplot(1, 1, 1, projection="3d")
#
axis.scatter(h.flatten(), s.flatten(), v.flatten(), facecolors=pixel_colors, marker="."
axis.set_xlabel("Hue")
axis.set_ylabel("Saturation")
axis.set_zlabel("Value")
plt.show()
#
#
axis.scatter(r.flatten(), g.flatten(), b.flatten(), facecolors=pixel_colors, marker="."
axis.set_xlabel("Red")
axis.set_ylabel("Green")
axis.set_zlabel("Blue")
plt.show()
#
light_green= (20, 30, 40)
dark_green = (100, 200, 180)
#
from matplotlib.colors import hsv_to_rgb
lo_square = np.full((10, 10, 3), light_green, dtype=np.uint8) / 255.0
do_square = np.full((10, 10, 3), dark_green, dtype=np.uint8) / 255.0
plt.subplot(1, 2, 1)
plt.imshow(hsv_to_rgb(lo_square))
plt.subplot(1, 2, 2)
plt.imshow(hsv_to_rgb(do_square))
plt.show()
#
mask = cv2.inRange(img,  light_green, dark_green)
result = cv2.bitwise_and(img, img, mask=mask)
plt.subplot(1, 2, 1)
```

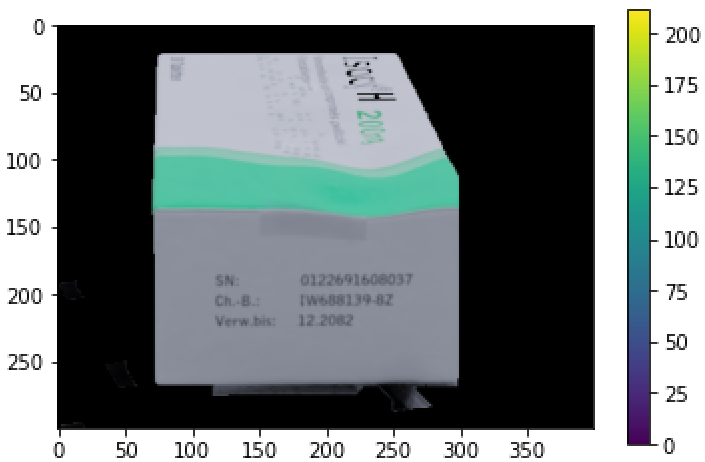
```
plt.imshow(mask, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(result)
plt.show()
#
#
light_white = (140, 140, 155)
dark_white = (150, 150, 165)
mask_white = cv2.inRange(img, light_white, dark_white)
result_white = cv2.bitwise_and(img, img, mask=mask_white)
lw_square = np.full((10, 10, 3), light_white, dtype=np.uint8) / 255.0
dw_square = np.full((10, 10, 3), dark_white, dtype=np.uint8) / 255.0
#
plt.subplot(1, 2, 1)
plt.imshow(hsv_to_rgb(lw_square))
plt.subplot(1, 2, 2)
plt.imshow(hsv_to_rgb(dw_square))
plt.show()
#
# #
plt.subplot(1, 2, 1)
plt.imshow(mask_white, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(result_white)
plt.show()

#crop_image = image[140:420, 335:755]


path='/Users/karth/Desktop/Data/data_sets/Processed/' + 'template.png'
#cv.imwrite(path, crop_image)



#template = cv.imread('Users/karth/Desktop/Data/Processed/template.png')
#height, width = template.shape[::]
#plt.imshow(template, cmap='gray')
#
files = ['damaged', 'intact']
adress = '/Users/karth/Desktop/Data/Processed/side_only/{}'
data_surface = {}
for f in files:
    data_surface[f]=[]
    for col in files:
        os.chdir(adress.format(col))
        for i in os.listdir(os.getcwd()):
            if i.endswith('.png'):
                data_surface[col].append(i)
#
#
#
start = time.time()
for title in files:
    os.chdir('/Users/karth/Desktop/Data/Processed/side_only/{}'.format(title))
    counter = 0
    for i in data_surface[title]:
        img_rgb = cv.imread(i)
        img_gray = cv.cvtColor(img_rgb, cv.COLOR_BGR2GRAY)
        res = cv.matchTemplate(img_gray, template, cv.TM_SQDIFF)
        min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
```

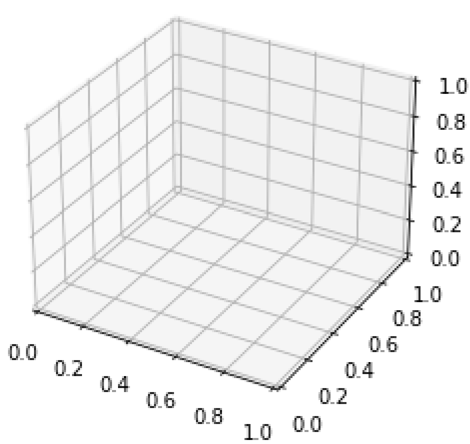
```
        top_left = min_loc
        w0 = top_left[0]
        h0 = top_left[1]
        crop_image = img_rgb[w0:w0 + width, h0:top_left[1] + height]
#
mask = np.zeros(img.shape[:2],np.uint8)
cv.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = img*mask2[:,:,np.newaxis]
path = '/Users/karth/Desktop/Data/Processed/side_only/' + title + '/' + i
cv.imwrite(path, crop_image)
clear_output(wait=True)
print("Processed Class",title)
calculate_time = time.time() - start
print("Process Time",round(calculate_time,3))
```

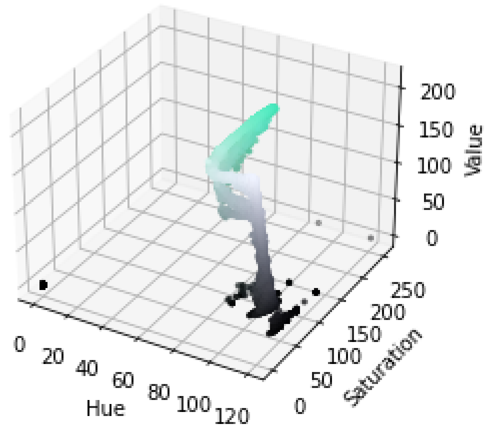


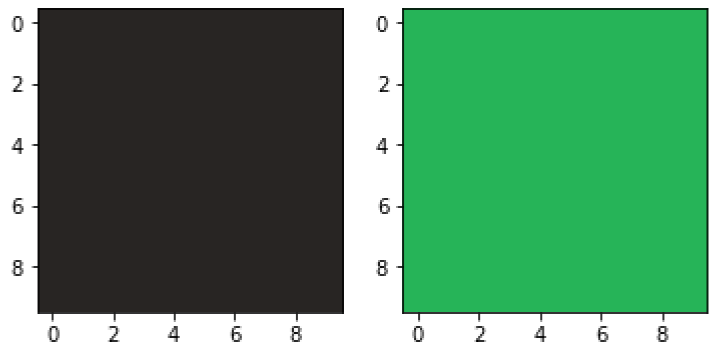


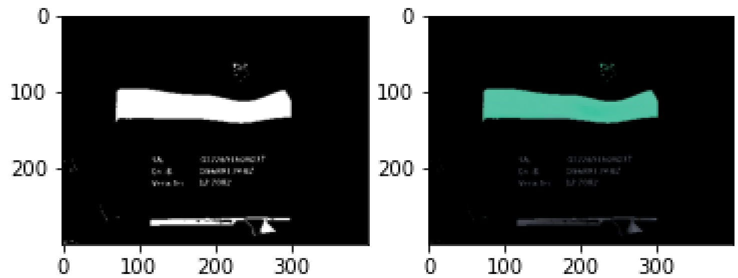

SN: 0122691608037
Ch.-B.: IW688139-8Z
Verw.bis: 12.2082

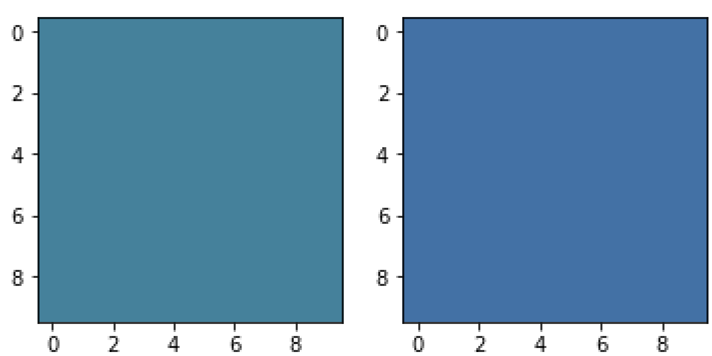

SN: 0122691608037
Ch.-B.: IW688139-8Z
Verw.bis: 12.2082

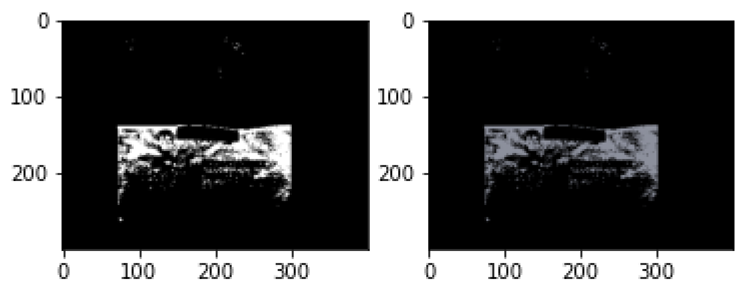












```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15208/3627630238.py in <module>
```

```
    140         for i in os.listdir(os.getcwd()):
    141             if i.endswith('.png'):
--> 142                 data_surface[col].append(i)
    143 #
    144 #

KeyError: 'intact'
```

In [46]:

```
import time
import os
from IPython.display import clear_output
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, l

datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.05,
    height_shift_range=0.05,
    shear_range=0.01,
    zoom_range=0.2,
    brightness_range=[0.2,1.0],
    horizontal_flip=True,
    fill_mode='nearest')


adress = '/Users/karth/Desktop/Data/Processed/side_only/{}'
files = ['damaged', 'intact']
data_surface = {}
for f in files:
    data_surface[f]=[]
for col in files:
    os.chdir(adress.format(col))
    for i in os.listdir(os.getcwd()):
        if i.endswith('.png'):
            data_surface[col].append(i)

start = time.time()
aug_dir='/Users/karth/Desktop/Data/Processed/aug_data'
for title in files:
    os.chdir('/Users/karth/Desktop/Data/Processed/side_only/{}'.format(title))
    counter = 0
    for i in data_surface[title]:
        img = load_img(i)
        x = img_to_array(img)
        x = x.reshape((1,) + x.shape)
        k = 0
        for batch in datagen.flow(x, batch_size=1,
                                  save_to_dir=aug_dir+'/'+title, save_prefix=i[:-4], save_forma
            k += 1
            if k > 19:
                break  # otherwise the generator would loop indefinitely
    clear_output(wait=True)
    print("Augmented class:",title)
calculate_time = time.time() - start
print("Augment Time:",round(calculate_time,3))
```

```
Augmented class: intact
Augment Time: 293.507
```

In [47]:

```
import pandas as pd
```

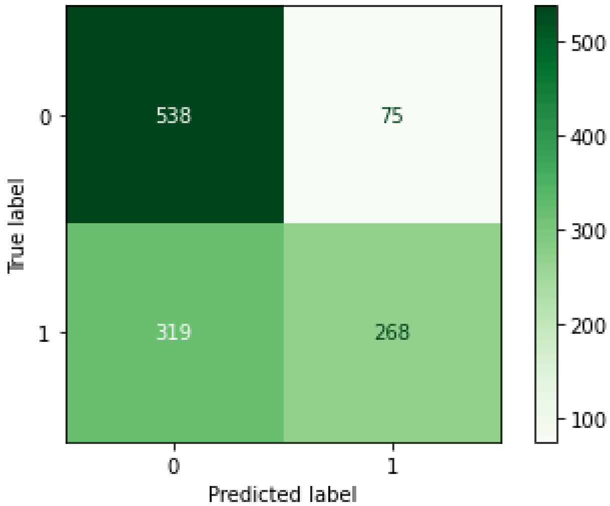
```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import time
import os
from IPython.display import clear_output
from sklearn.decomposition import PCA
import pickle as pk
import cv2

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, plot_confusion_matrix, plot_roc_curve
from keras.models import Sequential
from keras.utils import np_utils
from keras.layers import Dense, Dropout, GaussianNoise, Conv1D
from keras import regularizers
from sklearn.utils import shuffle
from keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

sample_size = 3000
width = 200
height = 150

files = ['damaged', 'intact']
adress = '/Users/karth/Desktop/Data/Processed/aug_data/{}'
data_surface = {}
for f in files:
    data_surface[f]=[]
for col in files:
    os.chdir(adress.format(col))
    for i in os.listdir(os.getcwd()):
        if i.endswith('.png'):
            data_surface[col].append(i)


start = time.time()
image_data = []
image_target = []

for title in files:
    os.chdir('/Users/karth/Desktop/Data/Processed/aug_data/{}'.format(title))
    counter = 0
    for i in data_surface[title]:
        img = cv2.imread(i,0)
        image_data.append(cv2.resize(img,(width, height)).flatten())
        image_target.append(title)
        counter += 1
        if counter == sample_size:
            break
    clear_output(wait=True)
    print("Compiled Class",title)
calculate_time = time.time() - start
```

```
print("Load Img Time",round(calculate_time,3))


image_data_array= np.array(image_data)
labels = LabelEncoder()
labels.fit(image_target)

# Normalization
image_data_norm =image_data_array

# PCA process
start = time.time()
pca = PCA()
pca.fit(image_data_norm)
#print('Time of PCA1: ',(time()-start))
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.5) + 1
print('PCA dimension: ',d)
# start = time()
pca = PCA(n_components=d)
image_data_PCA=pca.fit_transform(image_data_norm)
print('Time of PCA: ',round((time.time()-start),3))


y_labels = labels.transform(image_target)
#y = to_categorical(y_labels)
image_data_train, image_data_test, y_train, y_test = train_test_split(image_data_PCA, y
                                                                      test_size=0.2,
                                                                      random_state=42,s
# KNeighborsClassifier
model = KNeighborsClassifier(2)
model.fit(image_data_train, y_train)
y_pred = model.predict(image_data_test)
print("Acc:",round(accuracy_score(y_test,y_pred),2))
plot_confusion_matrix(model,image_data_test, y_test, cmap='Greens')
plt.show()

# SVC model
model = SVC()
model.C = 100
model.fit(image_data_train, y_train)
y_pred = model.predict(image_data_test)
print("Acc:",round(accuracy_score(y_test,y_pred),3))
plot_confusion_matrix(model,image_data_test, y_test, cmap='Greens')
plt.show()


#NN model
X_train = image_data_train
X_test = image_data_test
print('Train size: ', X_train.shape)
print('Test size: ', X_test.shape)

# NN model
model = Sequential()
model.add(Dense(32, kernel_regularizer=regularizers.l2(0.001), activation='relu', input
model.add(Dropout(rate=0.5))
model.add(Dense(8, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(Dropout(rate=0.4))
model.add(Dense(16, activation='relu'))
```

```
model.add(Dropout(rate=0.3))
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# build the model
start = time.time()
n_epochs=200
es_callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=50)
model_history = model.fit(X_train, y_train, batch_size=32, epochs=n_epochs, shuffle=Tru
                          validation_split = 0.2, callbacks=[es_callback])
print('Time per epoch: ',(time.time()-start))



pred_train= model.predict(X_train)
scores = model.evaluate(X_train, y_train, verbose=0)
print('Accuracy on training data: {}'.format(scores[1]))

pred_test= model.predict(X_test)
scores2 = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy on test data: {}'.format(scores2[1]))

pred_total= model.predict(image_data_PCA)
scores = model.evaluate(image_data_PCA, y_labels, verbose=0)
print('Accuracy on whole data: {}'.format(scores[1]))

plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
```

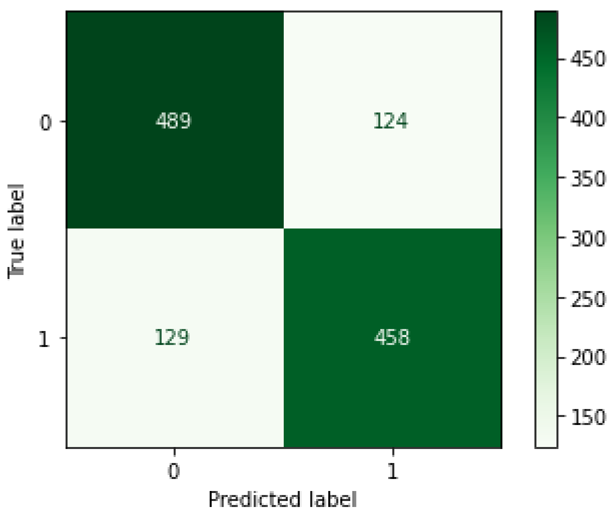
```
Compiled Class intact
Load Img Time 29.293
PCA dimension:  62
Time of PCA:  515.008
Acc: 0.67
```

```
C:\Users\karth\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is dep
recated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrix
Display.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
Acc: 0.789
```

```
C:\Users\karth\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is dep
recated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrix
Display.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
Train size:  (4800, 62)
Test size:  (1200, 62)
Epoch 1/200
120/120 [==============================] - 1s 3ms/step - loss: 88.8300 - accuracy: 0.503
6 - val_loss: 12.6010 - val_accuracy: 0.5052
Epoch 2/200
120/120 [==============================] - 0s 2ms/step - loss: 36.6425 - accuracy: 0.485
7 - val_loss: 2.8165 - val_accuracy: 0.5135
Epoch 3/200
120/120 [==============================] - 0s 2ms/step - loss: 15.8642 - accuracy: 0.502
1 - val_loss: 1.2362 - val_accuracy: 0.4958
Epoch 4/200
120/120 [==============================] - 0s 1ms/step - loss: 9.3355 - accuracy: 0.4974
- val_loss: 0.9227 - val_accuracy: 0.5052
Epoch 5/200
120/120 [==============================] - 0s 2ms/step - loss: 5.3835 - accuracy: 0.5146
- val_loss: 0.7886 - val_accuracy: 0.4917
Epoch 6/200
120/120 [==============================] - 0s 1ms/step - loss: 3.9344 - accuracy: 0.5143
- val_loss: 0.7527 - val_accuracy: 0.4875
Epoch 7/200
120/120 [==============================] - 0s 2ms/step - loss: 2.7722 - accuracy: 0.4987
- val_loss: 0.7498 - val_accuracy: 0.4875
```

```
Epoch 8/200
120/120 [==============================] - 0s 1ms/step - loss: 2.3445 - accuracy: 0.5141
- val_loss: 0.7423 - val_accuracy: 0.5125
Epoch 9/200
120/120 [==============================] - 0s 1ms/step - loss: 1.9944 - accuracy: 0.4930
- val_loss: 0.7406 - val_accuracy: 0.5104
Epoch 10/200
120/120 [==============================] - 0s 2ms/step - loss: 1.5015 - accuracy: 0.5005
- val_loss: 0.7400 - val_accuracy: 0.5063
Epoch 11/200
120/120 [==============================] - 0s 2ms/step - loss: 1.3952 - accuracy: 0.4911
- val_loss: 0.7394 - val_accuracy: 0.5115
Epoch 12/200
120/120 [==============================] - 0s 2ms/step - loss: 1.2762 - accuracy: 0.5060
- val_loss: 0.7394 - val_accuracy: 0.5115
Epoch 13/200
120/120 [==============================] - 0s 1ms/step - loss: 1.2018 - accuracy: 0.5000
- val_loss: 0.7393 - val_accuracy: 0.5146
Epoch 14/200
120/120 [==============================] - 0s 2ms/step - loss: 1.0737 - accuracy: 0.4924
- val_loss: 0.7395 - val_accuracy: 0.4885
Epoch 15/200
120/120 [==============================] - 0s 1ms/step - loss: 1.0459 - accuracy: 0.4987
- val_loss: 0.7392 - val_accuracy: 0.4885
Epoch 16/200
120/120 [==============================] - 0s 1ms/step - loss: 0.9487 - accuracy: 0.5039
- val_loss: 0.7392 - val_accuracy: 0.4875
Epoch 17/200
120/120 [==============================] - 0s 2ms/step - loss: 0.8705 - accuracy: 0.5029
- val_loss: 0.7390 - val_accuracy: 0.4865
Epoch 18/200
120/120 [==============================] - 0s 2ms/step - loss: 0.8651 - accuracy: 0.4948
- val_loss: 0.7389 - val_accuracy: 0.4865
Epoch 19/200
120/120 [==============================] - 0s 1ms/step - loss: 0.8514 - accuracy: 0.5039
- val_loss: 0.7391 - val_accuracy: 0.4875
Epoch 20/200
120/120 [==============================] - 0s 1ms/step - loss: 0.8531 - accuracy: 0.5036
- val_loss: 0.7392 - val_accuracy: 0.4875
Epoch 21/200
120/120 [==============================] - 0s 1ms/step - loss: 0.8348 - accuracy: 0.5039
- val_loss: 0.7390 - val_accuracy: 0.4854
Epoch 22/200
120/120 [==============================] - 0s 2ms/step - loss: 0.8303 - accuracy: 0.5063
- val_loss: 0.7390 - val_accuracy: 0.4854
Epoch 23/200
120/120 [==============================] - 0s 2ms/step - loss: 0.8083 - accuracy: 0.5086
- val_loss: 0.7388 - val_accuracy: 0.4854
Epoch 24/200
120/120 [==============================] - 0s 1ms/step - loss: 0.8157 - accuracy: 0.5010
- val_loss: 0.7387 - val_accuracy: 0.4854
Epoch 25/200
120/120 [==============================] - 0s 1ms/step - loss: 0.8168 - accuracy: 0.4979
- val_loss: 0.7387 - val_accuracy: 0.4865
Epoch 26/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7845 - accuracy: 0.5057
- val_loss: 0.7385 - val_accuracy: 0.4865
Epoch 27/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7761 - accuracy: 0.5018
- val_loss: 0.7386 - val_accuracy: 0.4865
Epoch 28/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7636 - accuracy: 0.5005
- val_loss: 0.7384 - val_accuracy: 0.4854
Epoch 29/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7905 - accuracy: 0.5047
```

```
- val_loss: 0.7381 - val_accuracy: 0.4865
Epoch 30/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7600 - accuracy: 0.5065
- val_loss: 0.7384 - val_accuracy: 0.4854
Epoch 31/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7537 - accuracy: 0.5063
- val_loss: 0.7384 - val_accuracy: 0.4854
Epoch 32/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7595 - accuracy: 0.4987
- val_loss: 0.7382 - val_accuracy: 0.4854
Epoch 33/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7454 - accuracy: 0.5018
- val_loss: 0.7379 - val_accuracy: 0.4865
Epoch 34/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7550 - accuracy: 0.5010
- val_loss: 0.7380 - val_accuracy: 0.4854
Epoch 35/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7556 - accuracy: 0.5026
- val_loss: 0.7377 - val_accuracy: 0.4865
Epoch 36/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7489 - accuracy: 0.4924
- val_loss: 0.7377 - val_accuracy: 0.4865
Epoch 37/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7484 - accuracy: 0.4995
- val_loss: 0.7376 - val_accuracy: 0.4865
Epoch 38/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7434 - accuracy: 0.4982
- val_loss: 0.7376 - val_accuracy: 0.4854
Epoch 39/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7536 - accuracy: 0.4971
- val_loss: 0.7375 - val_accuracy: 0.4854
Epoch 40/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7423 - accuracy: 0.4995
- val_loss: 0.7373 - val_accuracy: 0.4865
Epoch 41/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7391 - accuracy: 0.5010
- val_loss: 0.7372 - val_accuracy: 0.4854
Epoch 42/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7474 - accuracy: 0.5031
- val_loss: 0.7372 - val_accuracy: 0.4854
Epoch 43/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7452 - accuracy: 0.5057
- val_loss: 0.7370 - val_accuracy: 0.4854
Epoch 44/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7386 - accuracy: 0.5010
- val_loss: 0.7369 - val_accuracy: 0.4854
Epoch 45/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7386 - accuracy: 0.4945
- val_loss: 0.7367 - val_accuracy: 0.4854
Epoch 46/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7414 - accuracy: 0.4924
- val_loss: 0.7365 - val_accuracy: 0.4865
Epoch 47/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7422 - accuracy: 0.4872
- val_loss: 0.7364 - val_accuracy: 0.4865
Epoch 48/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7339 - accuracy: 0.5026
- val_loss: 0.7365 - val_accuracy: 0.4854
Epoch 49/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7392 - accuracy: 0.4977
- val_loss: 0.7362 - val_accuracy: 0.4854
Epoch 50/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7381 - accuracy: 0.5091
- val_loss: 0.7361 - val_accuracy: 0.4854
Epoch 51/200
```

```
120/120 [==============================] - 0s 1ms/step - loss: 0.7383 - accuracy: 0.4995
- val_loss: 0.7358 - val_accuracy: 0.4854
Epoch 52/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7355 - accuracy: 0.5049
- val_loss: 0.7356 - val_accuracy: 0.4865
Epoch 53/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7396 - accuracy: 0.5034
- val_loss: 0.7354 - val_accuracy: 0.4854
Epoch 54/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7356 - accuracy: 0.5036
- val_loss: 0.7352 - val_accuracy: 0.4854
Epoch 55/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7364 - accuracy: 0.4997
- val_loss: 0.7350 - val_accuracy: 0.4854
Epoch 56/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7348 - accuracy: 0.5057
- val_loss: 0.7348 - val_accuracy: 0.4854
Epoch 57/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7351 - accuracy: 0.4992
- val_loss: 0.7344 - val_accuracy: 0.4865
Epoch 58/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7366 - accuracy: 0.4935
- val_loss: 0.7343 - val_accuracy: 0.4854
Epoch 59/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7353 - accuracy: 0.4953
- val_loss: 0.7340 - val_accuracy: 0.4854
Epoch 60/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7341 - accuracy: 0.4943
- val_loss: 0.7338 - val_accuracy: 0.4854
Epoch 61/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7373 - accuracy: 0.5013
- val_loss: 0.7335 - val_accuracy: 0.4854
Epoch 62/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7337 - accuracy: 0.5026
- val_loss: 0.7331 - val_accuracy: 0.4854
Epoch 63/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7323 - accuracy: 0.5026
- val_loss: 0.7328 - val_accuracy: 0.4854
Epoch 64/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7333 - accuracy: 0.5042
- val_loss: 0.7326 - val_accuracy: 0.4854
Epoch 65/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7354 - accuracy: 0.4990
- val_loss: 0.7320 - val_accuracy: 0.4865
Epoch 66/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7325 - accuracy: 0.4878
- val_loss: 0.7317 - val_accuracy: 0.4865
Epoch 67/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7325 - accuracy: 0.4943
- val_loss: 0.7313 - val_accuracy: 0.4854
Epoch 68/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7321 - accuracy: 0.4875
- val_loss: 0.7309 - val_accuracy: 0.4854
Epoch 69/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7308 - accuracy: 0.5042
- val_loss: 0.7305 - val_accuracy: 0.4854
Epoch 70/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7296 - accuracy: 0.4846
- val_loss: 0.7301 - val_accuracy: 0.4854
Epoch 71/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7296 - accuracy: 0.5049
- val_loss: 0.7296 - val_accuracy: 0.4854
Epoch 72/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7304 - accuracy: 0.4935
- val_loss: 0.7290 - val_accuracy: 0.4865
```

```
Epoch 73/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7290 - accuracy: 0.5000
- val_loss: 0.7285 - val_accuracy: 0.4865
Epoch 74/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7287 - accuracy: 0.5018
- val_loss: 0.7280 - val_accuracy: 0.4854
Epoch 75/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7281 - accuracy: 0.4846
- val_loss: 0.7275 - val_accuracy: 0.4854
Epoch 76/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7275 - accuracy: 0.4922
- val_loss: 0.7269 - val_accuracy: 0.4854
Epoch 77/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7267 - accuracy: 0.5036
- val_loss: 0.7263 - val_accuracy: 0.4854
Epoch 78/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7262 - accuracy: 0.4914
- val_loss: 0.7257 - val_accuracy: 0.4865
Epoch 79/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7253 - accuracy: 0.4932
- val_loss: 0.7251 - val_accuracy: 0.4854
Epoch 80/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7244 - accuracy: 0.5031
- val_loss: 0.7244 - val_accuracy: 0.4854
Epoch 81/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7243 - accuracy: 0.4852
- val_loss: 0.7237 - val_accuracy: 0.4865
Epoch 82/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7229 - accuracy: 0.5039
- val_loss: 0.7230 - val_accuracy: 0.4865
Epoch 83/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7229 - accuracy: 0.5052
- val_loss: 0.7223 - val_accuracy: 0.4854
Epoch 84/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7210 - accuracy: 0.4930
- val_loss: 0.7216 - val_accuracy: 0.4854
Epoch 85/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7215 - accuracy: 0.5016
- val_loss: 0.7208 - val_accuracy: 0.4865
Epoch 86/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7201 - accuracy: 0.5023
- val_loss: 0.7200 - val_accuracy: 0.4865
Epoch 87/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7199 - accuracy: 0.5078
- val_loss: 0.7192 - val_accuracy: 0.4854
Epoch 88/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7175 - accuracy: 0.5076
- val_loss: 0.7184 - val_accuracy: 0.4875
Epoch 89/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7179 - accuracy: 0.5047
- val_loss: 0.7177 - val_accuracy: 0.4875
Epoch 90/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7164 - accuracy: 0.5065
- val_loss: 0.7167 - val_accuracy: 0.4885
Epoch 91/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7158 - accuracy: 0.5065
- val_loss: 0.7157 - val_accuracy: 0.4885
Epoch 92/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7134 - accuracy: 0.5065
- val_loss: 0.7140 - val_accuracy: 0.4906
Epoch 93/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7141 - accuracy: 0.5083
- val_loss: 0.7138 - val_accuracy: 0.4885
Epoch 94/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7134 - accuracy: 0.5057
```

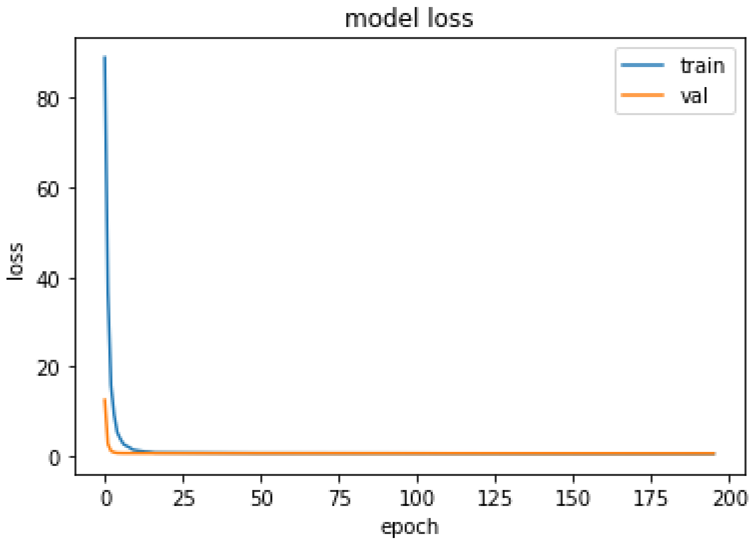
```
- val_loss: 0.7130 - val_accuracy: 0.4885
Epoch 95/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7121 - accuracy: 0.5089
- val_loss: 0.7118 - val_accuracy: 0.4896
Epoch 96/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7091 - accuracy: 0.5156
- val_loss: 0.7090 - val_accuracy: 0.5021
Epoch 97/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7073 - accuracy: 0.5164
- val_loss: 0.7055 - val_accuracy: 0.5073
Epoch 98/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7078 - accuracy: 0.5125
- val_loss: 0.7069 - val_accuracy: 0.5063
Epoch 99/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7048 - accuracy: 0.5271
- val_loss: 0.7059 - val_accuracy: 0.5052
Epoch 100/200
120/120 [==============================] - 0s 2ms/step - loss: 0.7056 - accuracy: 0.5271
- val_loss: 0.7073 - val_accuracy: 0.5083
Epoch 101/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6998 - accuracy: 0.5333
- val_loss: 0.7012 - val_accuracy: 0.5229
Epoch 102/200
120/120 [==============================] - 0s 1ms/step - loss: 0.7028 - accuracy: 0.5255
- val_loss: 0.7023 - val_accuracy: 0.5188
Epoch 103/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6961 - accuracy: 0.5437
- val_loss: 0.6987 - val_accuracy: 0.5396
Epoch 104/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6978 - accuracy: 0.5409
- val_loss: 0.6942 - val_accuracy: 0.5490
Epoch 105/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6987 - accuracy: 0.5466
- val_loss: 0.7008 - val_accuracy: 0.5354
Epoch 106/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6931 - accuracy: 0.5477
- val_loss: 0.6937 - val_accuracy: 0.5510
Epoch 107/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6994 - accuracy: 0.5461
- val_loss: 0.6940 - val_accuracy: 0.5594
Epoch 108/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6862 - accuracy: 0.5708
- val_loss: 0.6814 - val_accuracy: 0.5906
Epoch 109/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6870 - accuracy: 0.5599
- val_loss: 0.6908 - val_accuracy: 0.5708
Epoch 110/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6910 - accuracy: 0.5529
- val_loss: 0.6894 - val_accuracy: 0.5771
Epoch 111/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6897 - accuracy: 0.5625
- val_loss: 0.6844 - val_accuracy: 0.5813
Epoch 112/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6850 - accuracy: 0.5693
- val_loss: 0.6783 - val_accuracy: 0.6000
Epoch 113/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6852 - accuracy: 0.5695
- val_loss: 0.6808 - val_accuracy: 0.5958
Epoch 114/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6832 - accuracy: 0.5661
- val_loss: 0.6725 - val_accuracy: 0.6052
Epoch 115/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6824 - accuracy: 0.5706
- val_loss: 0.6768 - val_accuracy: 0.5990
Epoch 116/200
```

```
120/120 [==============================] - 0s 1ms/step - loss: 0.6771 - accuracy: 0.5852
- val_loss: 0.6705 - val_accuracy: 0.6031
Epoch 117/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6862 - accuracy: 0.5654
- val_loss: 0.6813 - val_accuracy: 0.6000
Epoch 118/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6782 - accuracy: 0.5802
- val_loss: 0.6760 - val_accuracy: 0.6125
Epoch 119/200
120/120 [==============================] - 0s 2ms/step - loss: 0.6758 - accuracy: 0.5854
- val_loss: 0.6672 - val_accuracy: 0.6229
Epoch 120/200
120/120 [==============================] - 0s 2ms/step - loss: 0.6755 - accuracy: 0.5776
- val_loss: 0.6740 - val_accuracy: 0.6094
Epoch 121/200
120/120 [==============================] - 0s 2ms/step - loss: 0.6791 - accuracy: 0.5766
- val_loss: 0.6698 - val_accuracy: 0.6125
Epoch 122/200
120/120 [==============================] - 0s 2ms/step - loss: 0.6775 - accuracy: 0.5781
- val_loss: 0.6696 - val_accuracy: 0.6115
Epoch 123/200
120/120 [==============================] - 0s 2ms/step - loss: 0.6808 - accuracy: 0.5786
- val_loss: 0.6717 - val_accuracy: 0.6094
Epoch 124/200
120/120 [==============================] - 0s 2ms/step - loss: 0.6761 - accuracy: 0.5844
- val_loss: 0.6681 - val_accuracy: 0.6021
Epoch 125/200
120/120 [==============================] - 0s 2ms/step - loss: 0.6775 - accuracy: 0.5815
- val_loss: 0.6693 - val_accuracy: 0.6042
Epoch 126/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6755 - accuracy: 0.5880
- val_loss: 0.6688 - val_accuracy: 0.6125
Epoch 127/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6679 - accuracy: 0.5911
- val_loss: 0.6651 - val_accuracy: 0.6271
Epoch 128/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6729 - accuracy: 0.5852
- val_loss: 0.6680 - val_accuracy: 0.6219
Epoch 129/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6703 - accuracy: 0.5930
- val_loss: 0.6656 - val_accuracy: 0.6198
Epoch 130/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6761 - accuracy: 0.5880
- val_loss: 0.6735 - val_accuracy: 0.6021
Epoch 131/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6730 - accuracy: 0.5870
- val_loss: 0.6792 - val_accuracy: 0.5885
Epoch 132/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6681 - accuracy: 0.5872
- val_loss: 0.6659 - val_accuracy: 0.6135
Epoch 133/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6672 - accuracy: 0.5977
- val_loss: 0.6768 - val_accuracy: 0.5917
Epoch 134/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6704 - accuracy: 0.5898
- val_loss: 0.6646 - val_accuracy: 0.6083
Epoch 135/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6652 - accuracy: 0.5964
- val_loss: 0.6707 - val_accuracy: 0.6010
Epoch 136/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6676 - accuracy: 0.5987
- val_loss: 0.6780 - val_accuracy: 0.6031
Epoch 137/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6632 - accuracy: 0.6023
- val_loss: 0.6630 - val_accuracy: 0.6250
```

```
Epoch 138/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6729 - accuracy: 0.5943
- val_loss: 0.6695 - val_accuracy: 0.6219
Epoch 139/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6561 - accuracy: 0.6187
- val_loss: 0.6679 - val_accuracy: 0.6073
Epoch 140/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6597 - accuracy: 0.6138
- val_loss: 0.6689 - val_accuracy: 0.6073
Epoch 141/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6647 - accuracy: 0.6177
- val_loss: 0.6700 - val_accuracy: 0.6010
Epoch 142/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6582 - accuracy: 0.6146
- val_loss: 0.6720 - val_accuracy: 0.5990
Epoch 143/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6605 - accuracy: 0.6198
- val_loss: 0.6696 - val_accuracy: 0.6010
Epoch 144/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6540 - accuracy: 0.6302
- val_loss: 0.6689 - val_accuracy: 0.6073
Epoch 145/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6564 - accuracy: 0.6260
- val_loss: 0.6673 - val_accuracy: 0.6187
Epoch 146/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6455 - accuracy: 0.6375
- val_loss: 0.6625 - val_accuracy: 0.6177
Epoch 147/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6415 - accuracy: 0.6406
- val_loss: 0.6669 - val_accuracy: 0.6010
Epoch 148/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6487 - accuracy: 0.6346
- val_loss: 0.6724 - val_accuracy: 0.6000
Epoch 149/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6427 - accuracy: 0.6380
- val_loss: 0.6705 - val_accuracy: 0.6125
Epoch 150/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6464 - accuracy: 0.6372
- val_loss: 0.6760 - val_accuracy: 0.5969
Epoch 151/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6424 - accuracy: 0.6367
- val_loss: 0.6689 - val_accuracy: 0.6042
Epoch 152/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6409 - accuracy: 0.6346
- val_loss: 0.6667 - val_accuracy: 0.6042
Epoch 153/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6465 - accuracy: 0.6393
- val_loss: 0.6646 - val_accuracy: 0.6083
Epoch 154/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6403 - accuracy: 0.6448
- val_loss: 0.6690 - val_accuracy: 0.5969
Epoch 155/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6366 - accuracy: 0.6401
- val_loss: 0.6666 - val_accuracy: 0.6010
Epoch 156/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6414 - accuracy: 0.6430
- val_loss: 0.6695 - val_accuracy: 0.6010
Epoch 157/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6341 - accuracy: 0.6443
- val_loss: 0.6714 - val_accuracy: 0.6000
Epoch 158/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6356 - accuracy: 0.6471
- val_loss: 0.6700 - val_accuracy: 0.6031
Epoch 159/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6266 - accuracy: 0.6531
```

```
- val_loss: 0.6706 - val_accuracy: 0.6000
Epoch 160/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6399 - accuracy: 0.6471
- val_loss: 0.6703 - val_accuracy: 0.6052
Epoch 161/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6389 - accuracy: 0.6461
- val_loss: 0.6744 - val_accuracy: 0.5979
Epoch 162/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6404 - accuracy: 0.6430
- val_loss: 0.6754 - val_accuracy: 0.5917
Epoch 163/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6353 - accuracy: 0.6555
- val_loss: 0.6682 - val_accuracy: 0.6031
Epoch 164/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6332 - accuracy: 0.6487
- val_loss: 0.6701 - val_accuracy: 0.6062
Epoch 165/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6355 - accuracy: 0.6547
- val_loss: 0.6689 - val_accuracy: 0.6031
Epoch 166/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6294 - accuracy: 0.6534
- val_loss: 0.6637 - val_accuracy: 0.6094
Epoch 167/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6377 - accuracy: 0.6513
- val_loss: 0.6663 - val_accuracy: 0.6042
Epoch 168/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6340 - accuracy: 0.6529
- val_loss: 0.6677 - val_accuracy: 0.6031
Epoch 169/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6363 - accuracy: 0.6471
- val_loss: 0.6714 - val_accuracy: 0.5969
Epoch 170/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6355 - accuracy: 0.6526
- val_loss: 0.6652 - val_accuracy: 0.6073
Epoch 171/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6354 - accuracy: 0.6516
- val_loss: 0.6675 - val_accuracy: 0.6125
Epoch 172/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6344 - accuracy: 0.6469
- val_loss: 0.6704 - val_accuracy: 0.6021
Epoch 173/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6348 - accuracy: 0.6578
- val_loss: 0.6710 - val_accuracy: 0.6052
Epoch 174/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6327 - accuracy: 0.6529
- val_loss: 0.6717 - val_accuracy: 0.5948
Epoch 175/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6231 - accuracy: 0.6664
- val_loss: 0.6677 - val_accuracy: 0.6021
Epoch 176/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6289 - accuracy: 0.6589
- val_loss: 0.6706 - val_accuracy: 0.6021
Epoch 177/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6230 - accuracy: 0.6591
- val_loss: 0.6655 - val_accuracy: 0.6115
Epoch 178/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6228 - accuracy: 0.6589
- val_loss: 0.6672 - val_accuracy: 0.6062
Epoch 179/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6325 - accuracy: 0.6602
- val_loss: 0.6676 - val_accuracy: 0.6073
Epoch 180/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6187 - accuracy: 0.6680
- val_loss: 0.6648 - val_accuracy: 0.6125
Epoch 181/200
```

```
120/120 [==============================] - 0s 1ms/step - loss: 0.6229 - accuracy: 0.6583
- val_loss: 0.6696 - val_accuracy: 0.6198
Epoch 182/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6289 - accuracy: 0.6568
- val_loss: 0.6637 - val_accuracy: 0.6208
Epoch 183/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6240 - accuracy: 0.6643
- val_loss: 0.6642 - val_accuracy: 0.6167
Epoch 184/200
120/120 [==============================] - ETA: 0s - loss: 0.6306 - accuracy: 0.66 - 0s
1ms/step - loss: 0.6338 - accuracy: 0.6536 - val_loss: 0.6691 - val_accuracy: 0.6062
Epoch 185/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6264 - accuracy: 0.6607
- val_loss: 0.6666 - val_accuracy: 0.6083
Epoch 186/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6231 - accuracy: 0.6643
- val_loss: 0.6649 - val_accuracy: 0.6104
Epoch 187/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6296 - accuracy: 0.6568
- val_loss: 0.6668 - val_accuracy: 0.6104
Epoch 188/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6279 - accuracy: 0.6581
- val_loss: 0.6669 - val_accuracy: 0.6073
Epoch 189/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6267 - accuracy: 0.6612
- val_loss: 0.6662 - val_accuracy: 0.6073
Epoch 190/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6145 - accuracy: 0.6750
- val_loss: 0.6668 - val_accuracy: 0.6062
Epoch 191/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6248 - accuracy: 0.6667
- val_loss: 0.6736 - val_accuracy: 0.6031
Epoch 192/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6256 - accuracy: 0.6565
- val_loss: 0.6721 - val_accuracy: 0.5969
Epoch 193/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6193 - accuracy: 0.6638
- val_loss: 0.6720 - val_accuracy: 0.6021
Epoch 194/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6214 - accuracy: 0.6677
- val_loss: 0.6626 - val_accuracy: 0.6177
Epoch 195/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6225 - accuracy: 0.6659
- val_loss: 0.6719 - val_accuracy: 0.6125
Epoch 196/200
120/120 [==============================] - 0s 1ms/step - loss: 0.6197 - accuracy: 0.6672
- val_loss: 0.6731 - val_accuracy: 0.6031
Time per epoch:  33.15892052650452
Accuracy on training data: 0.690416693687439
Accuracy on test data: 0.6000000238418579
Accuracy on whole data: 0.6723333597183228
```



In [48]:

```
import os
import keras.losses
import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.image import imread
import cv2
import time
from IPython.display import clear_output
# %matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import holoviews as hv
from holoviews import opts
hv.extension('bokeh')
import json

from tensorflow import keras
from keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_a
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense, Conv2D, MaxPoo
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.utils import plot_model
from tensorflow.keras import backend
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, plot_confusion_matrix, plot_roc_curve
from keras.utils import np_utils




sample_size = 3000
```

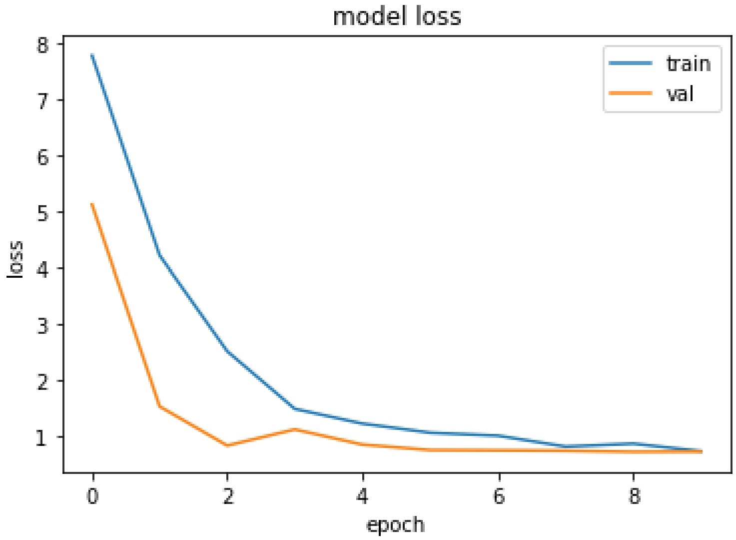
```
height = 90
width = 160


files = ['damaged', 'intact']
adress = '/Users/karth/Desktop/Data/Processed/side_only/{}'
data_surface = {}
for f in files:
    data_surface[f]=[]
for col in files:
    os.chdir(adress.format(col))
    for i in os.listdir(os.getcwd()):
        if i.endswith('.png'):
            data_surface[col].append(i)


start = time.time()
image_data = []
image_target = []

for title in files:
    os.chdir('/Users/karth/Desktop/Data/Processed/side_only/{}'.format(title))
    counter = 0
    for i in data_surface[title]:
        img = cv2.imread(i,0)
        image_data.append(cv2.resize(img,(width, height)))
        image_target.append(title)
        counter += 1
        if counter == sample_size:
            break
    clear_output(wait=True)
    print("Compiled Class",title)
calculate_time = time.time() - start
print("Load Img Time",round(calculate_time,3))


image_data_array= np.array(image_data)
print(image_data_array.shape)
labels = LabelEncoder()
labels.fit(image_target)
y_labels = labels.transform(image_target)

image_data_train, X_test, y_train_total, y_test = train_test_split(image_data, y_labels
                                                                              test_size=0
                                                                              shuffle=Tru
X_train, X_val, y_train, y_val = train_test_split(image_data_train, y_train_total,
                                                  test_size=0.2, random_state=42,
                                                  shuffle=True)

image_data_train= np.expand_dims(image_data_train, axis=-1)
X_train = np.expand_dims(X_train, axis=-1)
X_val = np.expand_dims(X_val, axis=-1)
X_test= np.expand_dims(X_test, axis=-1)

image_gen = ImageDataGenerator(rescale=None)
train_set = image_gen.flow(X_train,y_train)
val_set = image_gen.flow(X_val,y_val)
test_set = image_gen.flow(X_test,y_test)
image_data_train_set = image_gen.flow(image_data_train,y_train_total)
```

```
# ConV Net
image_shape = (height,width,1)
batch_size = 32
backend.clear_session()
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=(4,4), strides=2, input_shape=image_shape, act
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(filters=16, kernel_size=(2,2), strides=1, input_shape=image_shape, act
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(filters=32, kernel_size=(2,2), strides=1, input_shape=image_shape, act
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Flatten())
model.add(Dense(units=64,kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(Dense(112, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(units=1, activation='sigmoid'))


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

from time import time
n_epochs = 10
start = time()
es_callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=50)
model_history = model.fit_generator(train_set, epochs=n_epochs,
                                    shuffle=True, validation_data=val_set,
                                    callbacks=[es_callback])
print('Time per epoch',(time()-start)/n_epochs)

pred_train= model.predict(image_data_train_set)
scores = model.evaluate(image_data_train_set, verbose=0)
print('Accuracy on training data: {}'.format(scores[1]))

pred_val= model.predict(val_set)
scores2 = model.evaluate(val_set, verbose=0)
print('Accuracy on validation data: {}'.format(scores2[1]))

pred_test= model.predict(test_set)
scores2 = model.evaluate(test_set, verbose=0)
print('Accuracy on test data: {}'.format(scores2[1]))

pred_total= model.predict(image_data_train_set)
scores = model.evaluate(image_data_train_set, verbose=0)
print('Accuracy on whole data: {}'.format(scores[1]))


plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
```

```
Compiled Class intact
Load Img Time 0.728
(200, 90, 160)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 45, 80, 16)        272       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 22, 40, 16)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 22, 40, 16)        1040      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 11, 20, 16)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 11, 20, 32)        2080      
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 5, 10, 32)        0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 1600)              0         
                                                                 
 dense (Dense)               (None, 64)                102464    
                                                                 
 dense_1 (Dense)             (None, 112)               7280      
                                                                 
 dropout (Dropout)           (None, 112)               0         
                                                                 
 dense_2 (Dense)             (None, 1)                 113       
                                                                 
=================================================================
Total params: 113,249
Trainable params: 113,249
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 1s 79ms/step - loss: 7.7735 - accuracy: 0.5391 - val_loss: 5.1210 - val_accuracy: 0.5312
Epoch 2/10
4/4 [==============================] - 0s 43ms/step - loss: 4.2160 - accuracy: 0.5234 - val_loss: 1.5318 - val_accuracy: 0.5312
Epoch 3/10
4/4 [==============================] - 0s 41ms/step - loss: 2.5146 - accuracy: 0.4922 - val_loss: 0.8347 - val_accuracy: 0.5312
Epoch 4/10
4/4 [==============================] - 0s 41ms/step - loss: 1.4883 - accuracy: 0.4688 - val_loss: 1.1226 - val_accuracy: 0.4688
Epoch 5/10
4/4 [==============================] - 0s 36ms/step - loss: 1.2259 - accuracy: 0.5312 - val_loss: 0.8523 - val_accuracy: 0.5312
Epoch 6/10
4/4 [==============================] - 0s 39ms/step - loss: 1.0628 - accuracy: 0.5156 - val_loss: 0.7550 - val_accuracy: 0.5625
Epoch 7/10
4/4 [==============================] - 0s 36ms/step - loss: 1.0105 - accuracy: 0.4688 - val_loss: 0.7493 - val_accuracy: 0.5312
Epoch 8/10
4/4 [==============================] - 0s 37ms/step - loss: 0.8163 - accuracy: 0.5625 - val_loss: 0.7429 - val_accuracy: 0.5312
Epoch 9/10
4/4 [==============================] - 0s 36ms/step - loss: 0.8689 - accuracy: 0.4141 - val_loss: 0.7240 - val_accuracy: 0.5625
Epoch 10/10
4/4 [==============================] - 0s 35ms/step - loss: 0.7357 - accuracy: 0.5938 - val_loss: 0.7284 - val_accuracy: 0.5000
Time per epoch 0.2228483200073242
```

```
Accuracy on training data: 0.5375000238418579
Accuracy on validation data: 0.5
Accuracy on test data: 0.5
Accuracy on whole data: 0.5375000238418579
```




```
In [ ]:
```