NAME : AAKASH MOHAN
ID : 1001656408

SUBJECT CODE : EE5353
SUBJECT : Neural Networks and Deep Learning

PROGRAM ASSIGNMENT 8:

Coin Versus Scrap Recognition using Convolutional Neural Networks using Keras using Google Colab with Data augmentation

CNN with Augmentation:

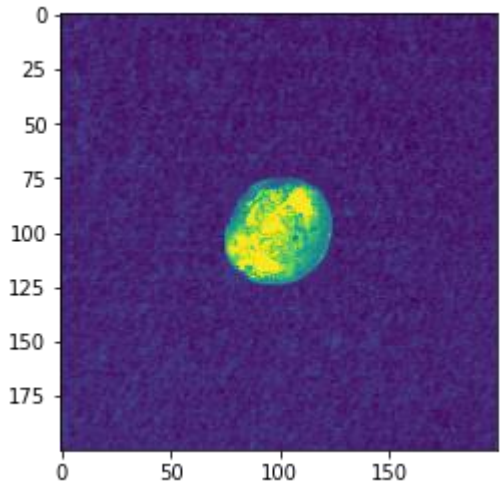
```
import tensorflow as tf
import random as rn
import os,cv2
import numpy as np

os.environ['PYTHONHASHSEED'] = '0'
# Setting the seed for numpy-generated random numbers
np.random.seed(37)
# Setting the seed for python random numbers
rn.seed(1254)
# Setting the graph-level random seed.
tf.set_random_seed(89)
from keras import backend as K
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,inter_op_parallelism_threads=

#Force Tensorflow to use a single thread
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)

import glob

from sklearn.utils import shuffle

from sklearn.model_selection import train_test_split

import re

from keras.utils import np_utils

import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
from keras.preprocessing.image import ImageDataGenerator


def gen_image(arr):
    two_d = (np.reshape(arr, (200, 200)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return  plt

def unique(list1):

    # insert the list to the set
    list_set = set(list1)
    # convert the set to the list
    unique_list = (list(list_set))
    for x in unique_list:
        print(x)

#from sklearn.cross_validation import train_test_split

from google.colab import drive
drive.mount('/content/drive')
```

```
PATH = os.getcwd()
# Define data path
data_path = '/content/drive/My Drive/Colab Notebooks/Coin_Recognition_Assignment_Dataset_f

data_dir_list = (os.listdir(data_path)) # os.listdir(data_path)


img_rows=128
img_cols=128
num_channel=1
num_epoch=20

# Define the number of classes
num_classes = 2

labels_name={'COIN':0, 'SCRAP':1}
img_data_list=[]
labels_list = []

for dataset in data_dir_list:
    img_list = glob.glob(data_path+'/'+ dataset +'/*.jpg')

    label = labels_name[dataset] # label is generated as the library updated above
    for img in img_list:
        input_img=cv2.imread(img,1 )
        input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
        input_img_resize=cv2.resize(input_img,(200,200))
        img_data_list.append(input_img_resize)
        labels_list.append(label)

#print(unique(labels_list))
img_data = np.array(img_data_list)
img_data = img_data.astype('float32')

labels = np.array(labels_list)

#print(unique(labels))
print(np.unique(labels,return_counts=True))
Y = np_utils.to_categorical(labels, num_classes)


#Shuffle the dataset
x,y = shuffle(img_data,Y, random_state=2)


X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2) #

#Normalization of the data
X_train = X_train / 255
X_test = X_test / 255

Nv = X_train.shape[0]
Nv_test = X_test.shape[0]
```

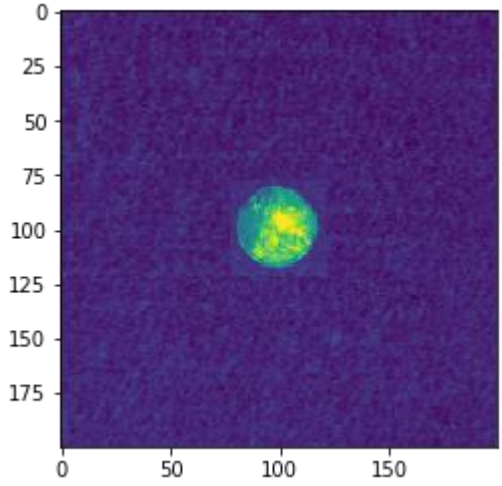
```
#reshape data to fit model
X_train = X_train.reshape(int(Nv),200,200,1)
X_test = X_test.reshape(int(Nv_test),200,200,1)

#create model
model = Sequential()
#add model layers

model.add(Conv2D(64, kernel_size=3,strides=(2,2), activation='relu', input_shape=(200,200,
        # 64 are the number of filters, kernel size is the size of the filters example 3*3
#model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

# 8. Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# 9. Fit model on training data
#model.fit(X_train, y_train, batch_size=32, nb_epoch=20, verbose=1, shuffle=False, validat
data_generator = ImageDataGenerator(vertical_flip=True,horizontal_flip=True)
data_generator.fit(X_train)
model.fit_generator(data_generator.flow(X_train, y_train, batch_size=32),steps_per_epoch=l


#TESTING

# Define data path
data_path = '/content/drive/My Drive/Colab Notebooks/Coin_Recognition_Assignment_Dataset_f
data_dir_list = (os.listdir(data_path)) # os.listdir(data_path)

# Define the number of classes
num_classes = 2

labels_name={'COIN':0, 'SCRAP':1}
img_data_list=[]
labels_list = []

for dataset in data_dir_list:
    img_list = glob.glob(data_path+'/'+ dataset +'/*.jpg')

    label = labels_name[dataset] # label is generated as the library updated above
    for img in img_list:
        input_img=cv2.imread(img,1 )
        input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
        input_img_resize=cv2.resize(input_img,(200,200))
        img_data_list.append(input_img_resize)
        labels_list.append(label)

        #print(unique(labels_list))
img_data = np.array(img_data_list)
img data = img data.astype('float32')
```

```
labels = np.array(labels_list)

#print(unique(labels))
print(np.unique(labels,return_counts=True))
Y = np_utils.to_categorical(labels, num_classes)


#Shuffle the dataset
x,y = shuffle(img_data,Y, random_state=2)



#Normalization of the data
X_t = x / 255
y_t=y

Nv_test = X_t.shape[0]

#reshape data to fit model
X_t = X_t.reshape(int(Nv_test),200,200,1)




# 10. Evaluate model on test data
score = model.evaluate(X_t, y_t, verbose=1)

print('Testing accuracy - > ',score[1] * 100)

ytested = model.predict_classes(X_t)
for i in range(10):
  print("The Predicted Testing image is =%s verify below" % ((list(labels_name.keys())[lis
  gen_image(X_t[i]).show() # printing image vs the predicted image below
```

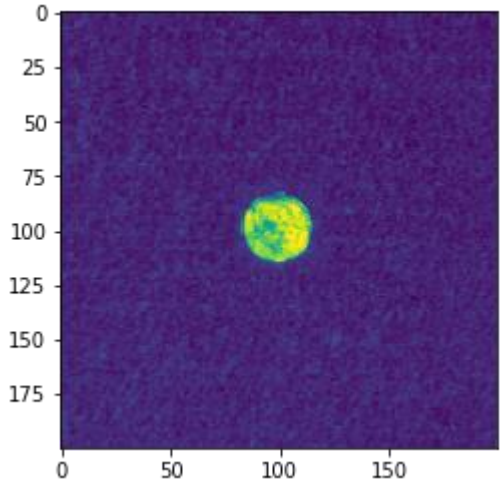
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
(array([0, 1]), array([921, 400]))
Epoch 1/20
33/33 [==============================] - 5s 149ms/step - loss: 0.6834 - acc: 0.7216 -
Epoch 2/20
33/33 [==============================] - 1s 20ms/step - loss: 0.4329 - acc: 0.8210 -
Epoch 3/20
33/33 [==============================] - 1s 21ms/step - loss: 0.3839 - acc: 0.8542 -
Epoch 4/20
33/33 [==============================] - 1s 20ms/step - loss: 0.3713 - acc: 0.8542 -
Epoch 5/20
33/33 [==============================] - 1s 20ms/step - loss: 0.3409 - acc: 0.8646 -
Epoch 6/20
33/33 [==============================] - 1s 20ms/step - loss: 0.3347 - acc: 0.8759 -
Epoch 7/20
33/33 [==============================] - 1s 19ms/step - loss: 0.3119 - acc: 0.8807 -
Epoch 8/20
33/33 [==============================] - 1s 19ms/step - loss: 0.3116 - acc: 0.8731 -
Epoch 9/20
33/33 [==============================] - 1s 20ms/step - loss: 0.3009 - acc: 0.8722 -
Epoch 10/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2943 - acc: 0.8845 -
Epoch 11/20
33/33 [==============================] - 1s 19ms/step - loss: 0.2869 - acc: 0.8816 -
Epoch 12/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2650 - acc: 0.8826 -
Epoch 13/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2681 - acc: 0.8902 -
Epoch 14/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2657 - acc: 0.8996 -
Epoch 15/20
33/33 [==============================] - 1s 21ms/step - loss: 0.2537 - acc: 0.8920 -
Epoch 16/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2515 - acc: 0.8977 -
Epoch 17/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2490 - acc: 0.8892 -
Epoch 18/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2188 - acc: 0.9053 -
Epoch 19/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2256 - acc: 0.9044 -
Epoch 20/20
33/33 [==============================] - 1s 20ms/step - loss: 0.2267 - acc: 0.9015 -
(array([0, 1]), array([276, 120]))
396/396 [==============================] - 0s 213us/step
Testing accuracy - >  94.6969697571764
The Predicted Testing image is =COIN verify below
```



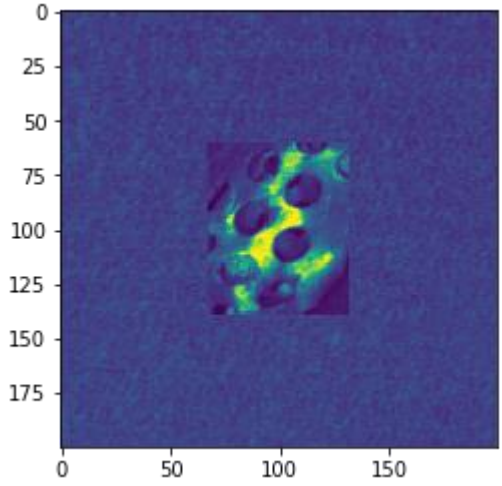
```
The Predicted Testing image is =COIN verify below
```



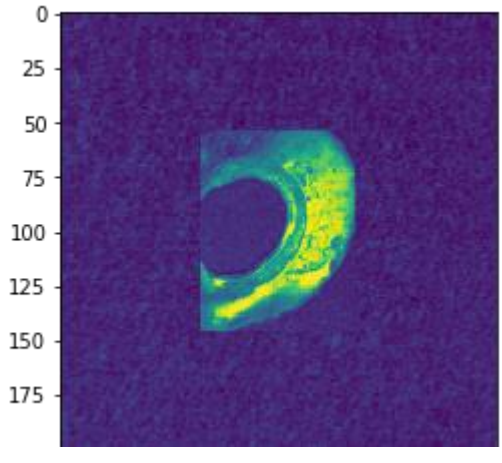

```
The Predicted Testing image is =COIN verify below
```

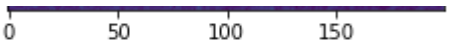



```
The Predicted Testing image is =SCRAP verify below
```

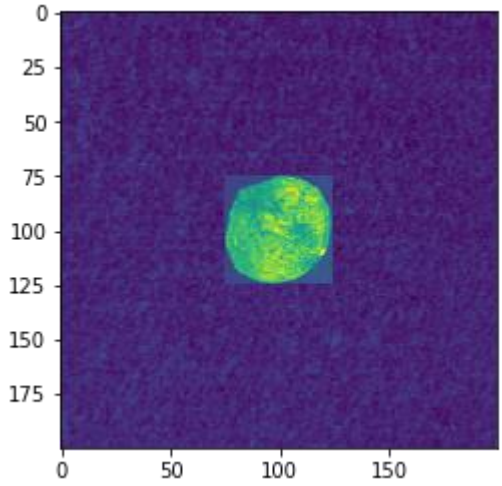



```
The Predicted Testing image is =SCRAP verify below
```

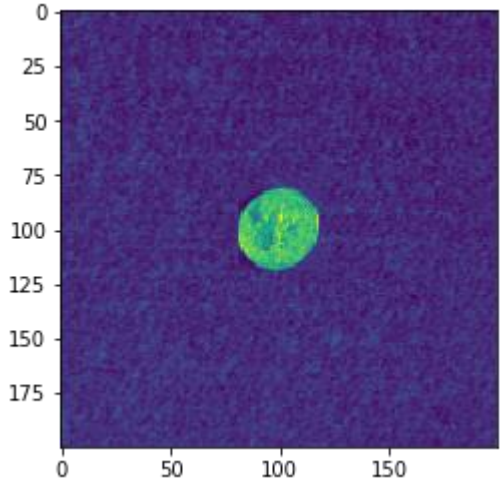




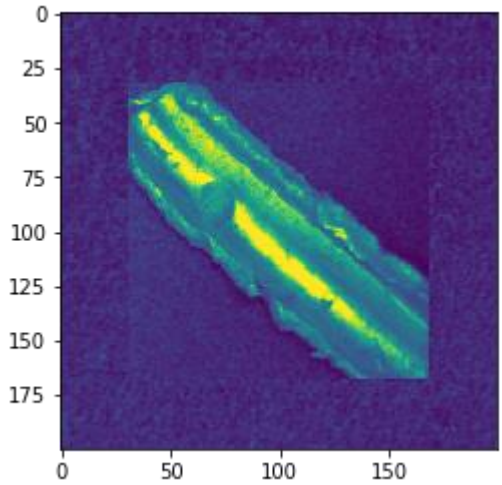

```
The Predicted Testing image is =COIN verify below
```



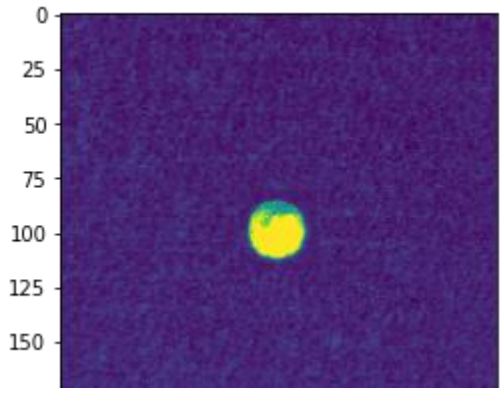

```
The Predicted Testing image is =COIN verify below
```

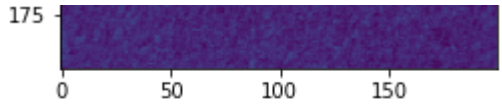



```
The Predicted Testing image is =SCRAP verify below
```

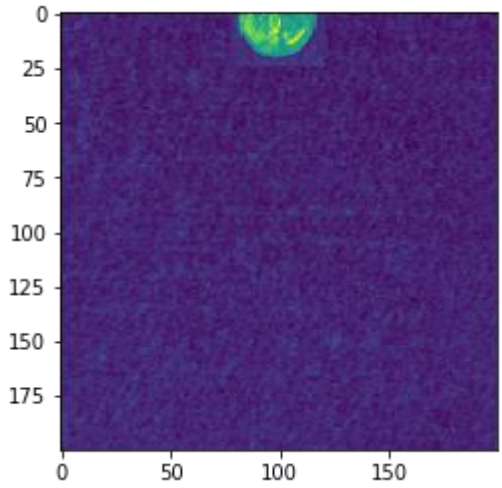



```
The Predicted Testing image is =COIN verify below
```






```
The Predicted Testing image is =COIN verify below
```



CNN without Augmentation:

```
import tensorflow as tf
import random as rn
import os,cv2
import numpy as np
```

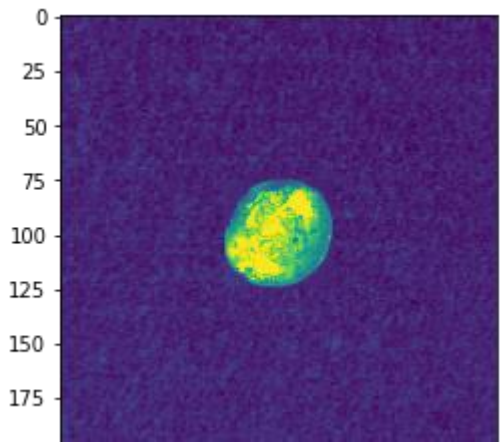
```
os.environ['PYTHONHASHSEED'] = '0'
# Setting the seed for numpy-generated random numbers
np.random.seed(37)
# Setting the seed for python random numbers
rn.seed(1254)
# Setting the graph-level random seed.
tf.set_random_seed(89)
from keras import backend as K
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,inter_op_parallelism_threads=

#Force Tensorflow to use a single thread
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)

import glob

from sklearn.utils import shuffle

from sklearn.model_selection import train_test_split

import re

from keras.utils import np_utils

import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
from keras.preprocessing.image import ImageDataGenerator


def gen_image(arr):
    two_d = (np.reshape(arr, (200, 200)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return  plt

def unique(list1):

    # insert the list to the set
    list_set = set(list1)
    # convert the set to the list
    unique_list = (list(list_set))
    for x in unique_list:
        print(x)

#from sklearn.cross_validation import train_test_split

from google.colab import drive
drive.mount('/content/drive')
```

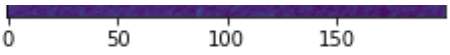
```
PATH = os.getcwd() #
Define data path
data_path = '/content/drive/My Drive/Colab Notebooks/Coin_Recognition_Assignment_Dataset_f

data_dir_list = (os.listdir(data_path)) # os.listdir(data_path)


img_rows=128
img_cols=128
num_channel=1
num_epoch=20

# Define the number of classes
num_classes = 2

labels_name={'COIN':0, 'SCRAP':1}
img_data_list=[]
labels_list = []

for dataset in data_dir_list:
        img_list = glob.glob(data_path+'/'+ dataset +'/*.jpg')

        label = labels_name[dataset] # label is generated as the library updated above
        for img in img_list:
            input_img=cv2.imread(img,1 )
            input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
            input_img_resize=cv2.resize(input_img,(200,200))
            img_data_list.append(input_img_resize)
            labels_list.append(label)

#print(unique(labels_list)) img_data =
np.array(img_data_list)
img_data = img_data.astype('float32')

labels = np.array(labels_list)

#print(unique(labels))
print(np.unique(labels,return_counts=True))
Y = np_utils.to_categorical(labels, num_classes)


#Shuffle the dataset
x,y = shuffle(img_data,Y, random_state=2)


X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2) #

#Normalization of the data
X_train = X_train / 255
X_test = X_test / 255

Nv = X_train.shape[0] Nv_test
= X_test.shape[0]
```

```
#reshape data to fit model
X_train = X_train.reshape(int(Nv),200,200,1) X_test
= X_test.reshape(int(Nv_test),200,200,1)

#create model
model = Sequential()
#add model layers

    model.add(Conv2D(64, kernel_size=3,strides=(2,2), activation='relu', input_shape=(200,200,
            # 64 are the number of filters, kernel size is the size of the filters example 3*3
#model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
model.add(Flatten())  model.add(Dense(128,
activation='relu')) model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

# 8. Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# 9. Fit model on training data
model.fit(X_train, y_train, batch_size=32, nb_epoch=20, verbose=1, shuffle=False, validati
'''data_generator = ImageDataGenerator(vertical_flip=True,horizontal_flip=True)
data_generator.fit(X_train)
model.fit_generator(data_generator.flow(X_train, y_train, batch_size=32),steps_per_epoch=l


#TESTING

# Define data path
data_path = '/content/drive/My Drive/Colab Notebooks/Coin_Recognition_Assignment_Dataset_f
data_dir_list = (os.listdir(data_path)) # os.listdir(data_path)

# Define the number of classes
num_classes = 2

labels_name={'COIN':0, 'SCRAP':1}
img_data_list=[]
labels_list = []

for dataset in data_dir_list:
        img_list = glob.glob(data_path+'/'+ dataset +'/*.jpg')

        label = labels_name[dataset] # label is generated as the library updated above
        for img in img_list:
            input_img=cv2.imread(img,1 )
            input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
            input_img_resize=cv2.resize(input_img,(200,200))
            img_data_list.append(input_img_resize)
            labels_list.append(label)

        #print(unique(labels_list))
img_data = np.array(img_data_list) img
data = img data.astype('float32')
```

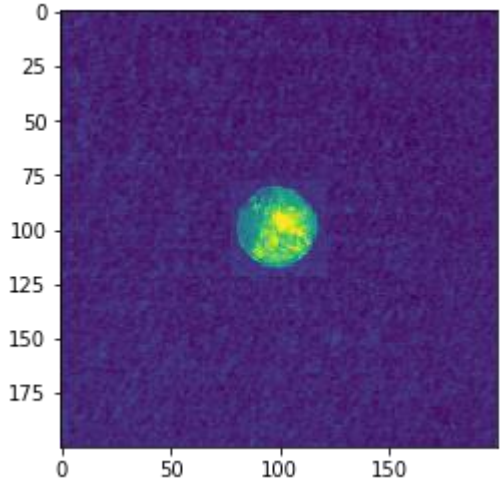
```
labels = np.array(labels_list)

#print(unique(labels))
print(np.unique(labels,return_counts=True))
Y = np_utils.to_categorical(labels, num_classes)


#Shuffle the dataset
x,y = shuffle(img_data,Y, random_state=2)



#Normalization of the data X_t
= x / 255
y_t=y

Nv_test = X_t.shape[0]

#reshape data to fit model
X_t = X_t.reshape(int(Nv_test),200,200,1)




# 10. Evaluate model on test data
score = model.evaluate(X_t, y_t, verbose=1)

print('Testing accuracy - > ',score[1] * 100)

ytested = model.predict_classes(X_t)
for i in range(10):
      print("The Predicted Testing image is =%s verify below" % ((list(labels_name.keys())[lis
      gen_image(X_t[i]).show() # printing image vs the predicted image below
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
(array([0, 1]), array([921, 400]))
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:132: UserWarning: The `n
Train on 1056 samples, validate on 265 samples
Epoch 1/20
1056/1056 [==============================] - 5s 5ms/step - loss: 0.8547 - acc: 0.7140
Epoch 2/20
1056/1056 [==============================] - 1s 608us/step - loss: 0.3931 - acc: 0.84
Epoch 3/20
1056/1056 [==============================] - 1s 605us/step - loss: 0.3614 - acc: 0.86
Epoch 4/20
1056/1056 [==============================] - 1s 599us/step - loss: 0.3330 - acc: 0.86
Epoch 5/20
1056/1056 [==============================] - 1s 615us/step - loss: 0.3235 - acc: 0.87
Epoch 6/20
1056/1056 [==============================] - 1s 599us/step - loss: 0.3002 - acc: 0.88
Epoch 7/20
1056/1056 [==============================] - 1s 602us/step - loss: 0.2998 - acc: 0.88
Epoch 8/20
1056/1056 [==============================] - 1s 611us/step - loss: 0.2970 - acc: 0.88
Epoch 9/20
1056/1056 [==============================] - 1s 593us/step - loss: 0.2738 - acc: 0.89
Epoch 10/20
1056/1056 [==============================] - 1s 620us/step - loss: 0.2597 - acc: 0.89
Epoch 11/20
1056/1056 [==============================] - 1s 603us/step - loss: 0.2497 - acc: 0.90
Epoch 12/20
1056/1056 [==============================] - 1s 602us/step - loss: 0.2287 - acc: 0.91
Epoch 13/20
1056/1056 [==============================] - 1s 592us/step - loss: 0.2058 - acc: 0.91
Epoch 14/20
1056/1056 [==============================] - 1s 623us/step - loss: 0.1909 - acc: 0.91
Epoch 15/20
1056/1056 [==============================] - 1s 590us/step - loss: 0.1890 - acc: 0.92
Epoch 16/20
1056/1056 [==============================] - 1s 604us/step - loss: 0.1746 - acc: 0.92
Epoch 17/20
1056/1056 [==============================] - 1s 617us/step - loss: 0.1584 - acc: 0.93
Epoch 18/20
1056/1056 [==============================] - 1s 611us/step - loss: 0.1475 - acc: 0.94
Epoch 19/20
1056/1056 [==============================] - 1s 611us/step - loss: 0.1476 - acc: 0.94
Epoch 20/20
1056/1056 [==============================] - 1s 592us/step - loss: 0.1338 - acc: 0.95
(array([0, 1]), array([276, 120]))
396/396 [==============================] - 0s 199us/step
Testing accuracy - >  97.72727278747944
The Predicted Testing image is =COIN verify below
```

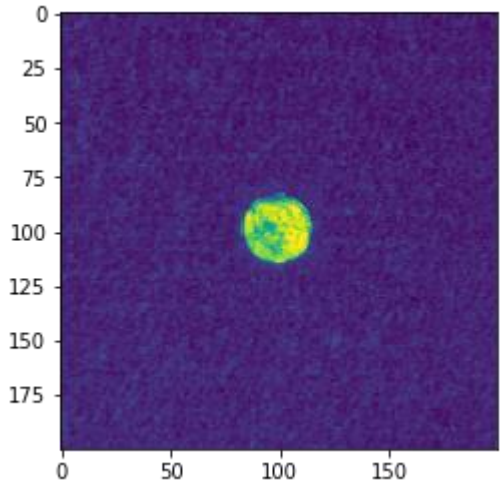




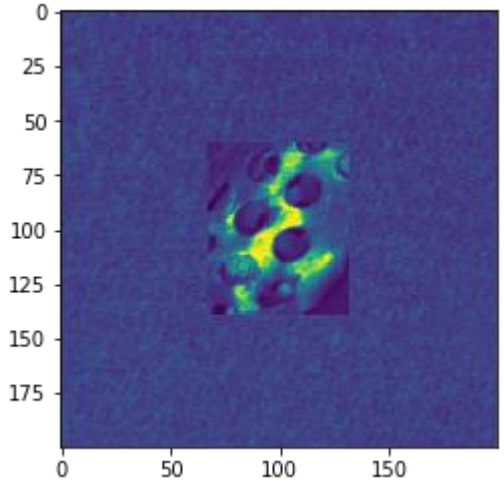
The Predicted Testing image is =COIN verify below



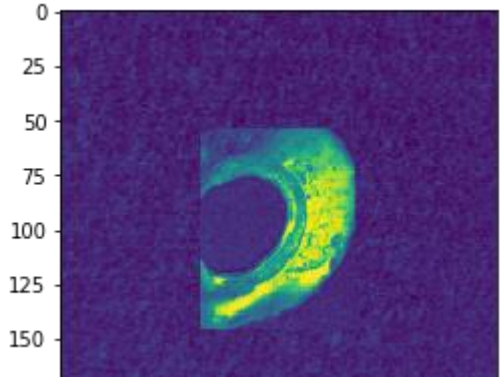
The Predicted Testing image is =COIN verify below

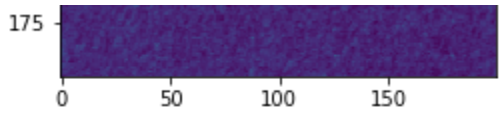


The Predicted Testing image is =SCRAP verify below

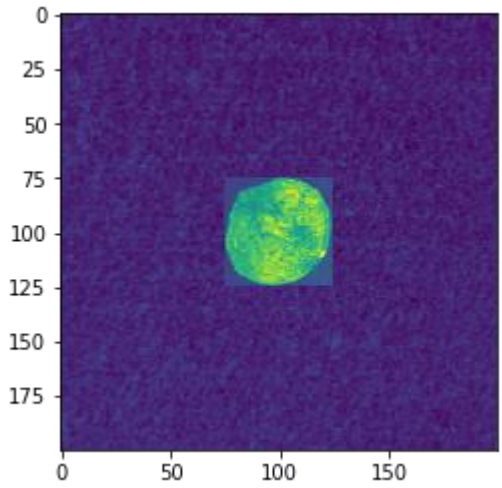


The Predicted Testing image is =SCRAP verify below

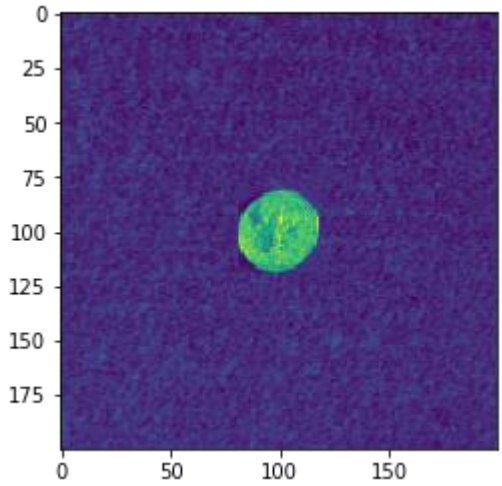




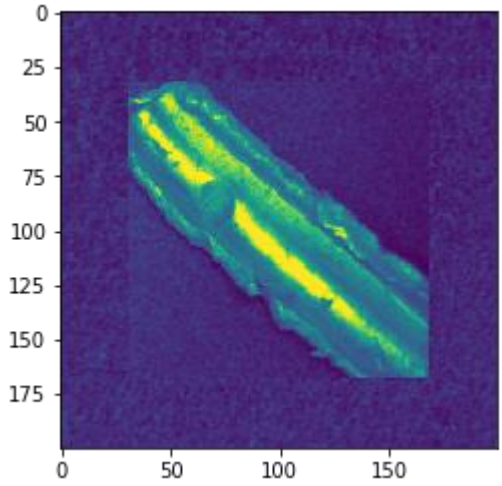

```
The Predicted Testing image is =COIN verify below
```



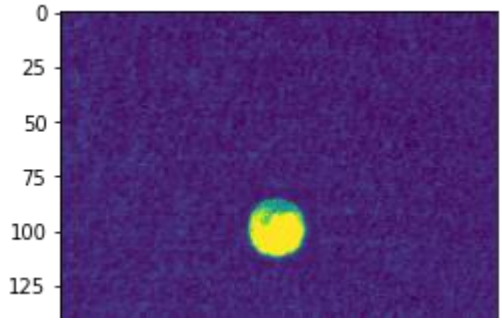

```
The Predicted Testing image is =COIN verify below
```

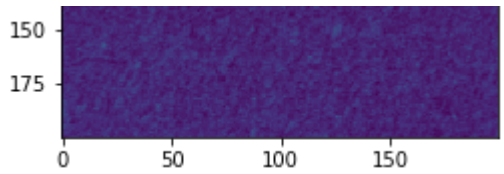



```
The Predicted Testing image is =SCRAP verify below
```

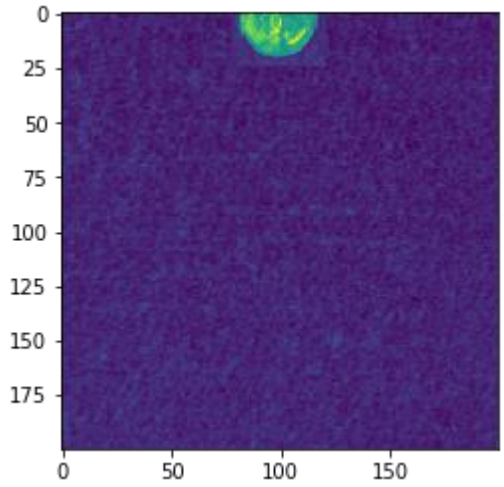



```
The Predicted Testing image is =COIN verify below
```






```
The Predicted Testing image is =COIN verify below
```



EXPLANATION:

- At the beginning we are including few libraries and lines of code to generate reproduceable result, (i,e) to get same accuracy no matter how many times we run the code.
- The environ['PYTHONHASHSEED'] is used to maintain the precision which helps us to get same results.
- This seed is set by means of numpy a library which generates random numbers at python level.
- Then we are importing some basics keras library files which is required for our code, the only new library file that we are importing is the ImageDataGenerator.
- Then we use the "Lambda func" to sort the image and generate image. The lambda function is used for higher order functions.
- We use the tensor flow for all the functions in this program.
- We use two different data path in this program one for the training file and another for the testing
- We first select the data path for the training file and use python data structures to create the label list as coin and scarp.
- Since we have to identify if it's coin or scrap the total classes will be 2 and change the file from png file to jpg file.

- Then we convert the image from color to black and white and then resize the image and then shuffle the dataset.
- Then we split the dataset into training and validation and normalize the values. Then we add the layers to support our program like, convolution layer , pooling layer, dropout, flatten, dropout and dense layer.
- Then we compile and train the data to get the results, next we set the data path for the testing file.
- Repeat the same procedure as training for testing part but we don't have to compile and fit the layer, instead we evaluate the testing file and get the results.
- We repeat the same procedure with data augmentation to increase the accuracy of the program, we include the ImageDataGenerator part in this data augmentation part.
- The data augmentation is to achieve better result with trained with the data.

CONCLUSION:

In this assignment we code in order to differentiate between coin and scrap. We provide separate data paths for training and testing file and provide with data augmentation to achieve better results. We import the necessary files and try to train and test the images. We provide the model layers and reshape the images and evaluate the testing file. The training, validation and testing have accuracies of 90% and above.