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]))

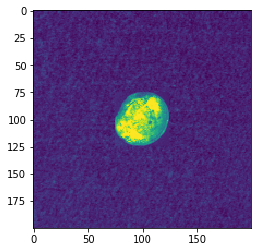
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch | 1/20 |  | | | | | | | |
| 33/33 | [==============================] | - | 5s | 149ms/step - loss: 0.6834 - acc: 0.7216 - | | | | | |
| Epoch | 2/20 |  |  |  | | | | | |
| 33/33 | [==============================] | - 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]))

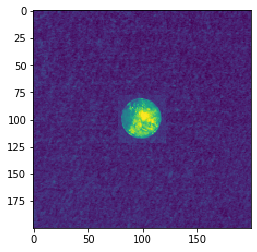
396/396 [==============================] - 0s 213us/step

Testing accuracy - > 94.6969697571764

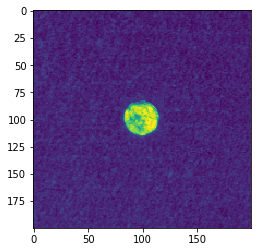
The Predicted Testing image is =COIN verify below



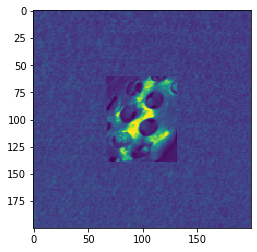
The Predicted Testing image is =COIN verify below



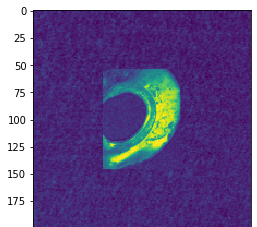
The Predicted Testing image is =COIN verify below



The Predicted Testing image is =SCRAP verify below

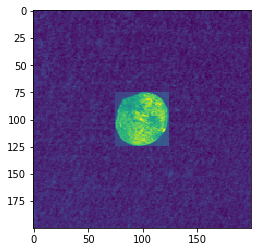


The Predicted Testing image is =SCRAP verify below

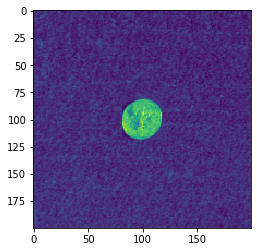




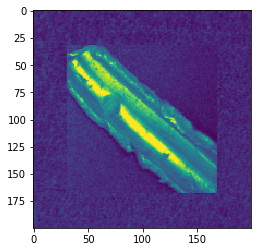
The Predicted Testing image is =COIN verify below



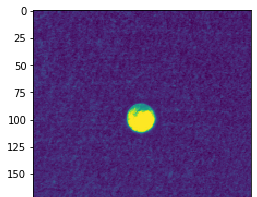
The Predicted Testing image is =COIN verify below



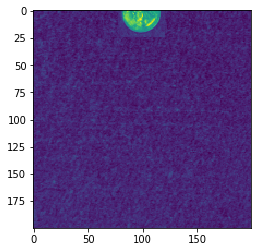
The Predicted Testing image is =SCRAP verify below



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

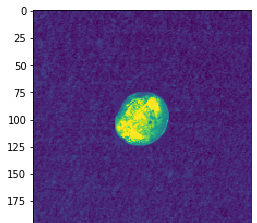
Epoch 9/20

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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

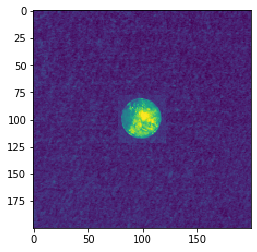
Testing accuracy - > 97.72727278747944

The Predicted Testing image is =COIN verify below

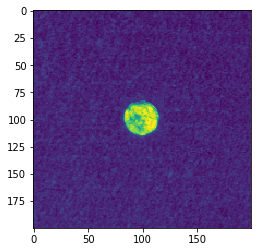




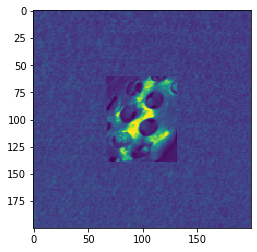
The Predicted Testing image is =COIN verify below



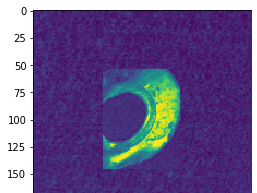
The Predicted Testing image is =COIN verify below



The Predicted Testing image is =SCRAP verify below

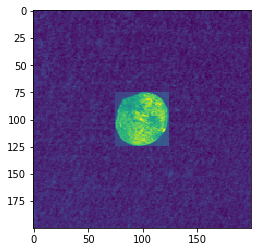


The Predicted Testing image is =SCRAP verify below

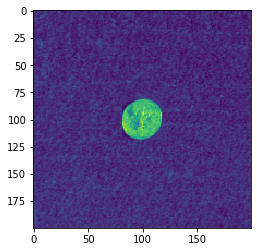




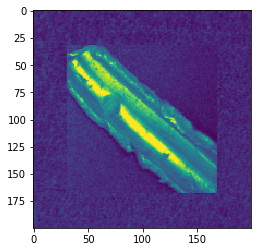
The Predicted Testing image is =COIN verify below



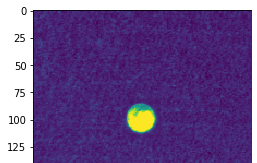
The Predicted Testing image is =COIN verify below



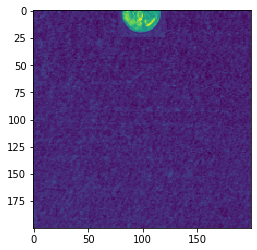
The Predicted Testing image is =SCRAP verify below



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.