Jose Tenorio

Coin vs Scrap

EE 5353 Neural Networks

Introduction:

The purpose of this program is to utilize the free cloud service Google Colab. This service allows the user two write and run executable documents. Google Colab connects the user's notebook to a cloud base runtime and execute python code without any setup on the user's machine.

This program executes python code on **Google Colab** that identifies imagines using **convolutional neural nets** (CNN). CNN's can capture spatial and temporal dependencies of an image through the application of filters. Convolutional neural nets reduce the image into a form which is easier to process without losing features which are critical for a good prediction. This process is done with the use of the **Kernel filter**. This filter is also able to extract high-level features such as edges from the input image. The **pooling layer** is responsible for reducing the spatial size of the convolved feature. This helps decrease the computational power required to process the data. This layer is also useful for extracting dominant features which are rotational and positional invariant. Lastly, the **dropout layer** refers to ignoring units during the training phase of certain neurons which are selected at random. By ignoring, I mean they are not considered during a particular forward or backward pass. We do this to prevent over-fitting.

In this program the graduate teaching assistant provided minimum amounts of code. Our task was to write the code for this program based on the past two assignments. In this program we utilized the **model.fit** using the **validation data**; this greatly aided our training process. Additionally, we used **data_generator.fit**. This line of coded added augmentation to the training. The generator is run in parallel to the model, for efficiency. For instance, this allows you to do real-time data augmentation on images on CPU in parallel to training your model on GPU

Procedure:

1. Input the images from the training folder in proper image and label format(use onehot encoding/to_categorical)
2. Divide the data into training (80%) and **validation(20%)**.
3. Resize the images to 200 by 200.
4. Convert the images to black and white
5. Normalize the input data
6. Design a convolutional neural network with he following features:

- Convolutional layer with 64 filters, Size of the filters is 3, 3, Strides is 2 and relu activation
- Pooling layer with pool size 2,2
- Dropout layer with rate 0.5
- Flattening
- Dense layer fully connected with 128 hidden units and relu activations
- Dropout layer with rate 0.5
- Final dense fully connected layer with number of classes and softmax activation.

7. Verify if the number of iterations/nb_epochs = 20.

8. Validation data should be used during training

9. Input the images from the testing folder in proper image and label format as used for training. Shuffle the testing data.

10. Test the images. Print results and testing figure

Part I CNN without augmentation

Training Results:

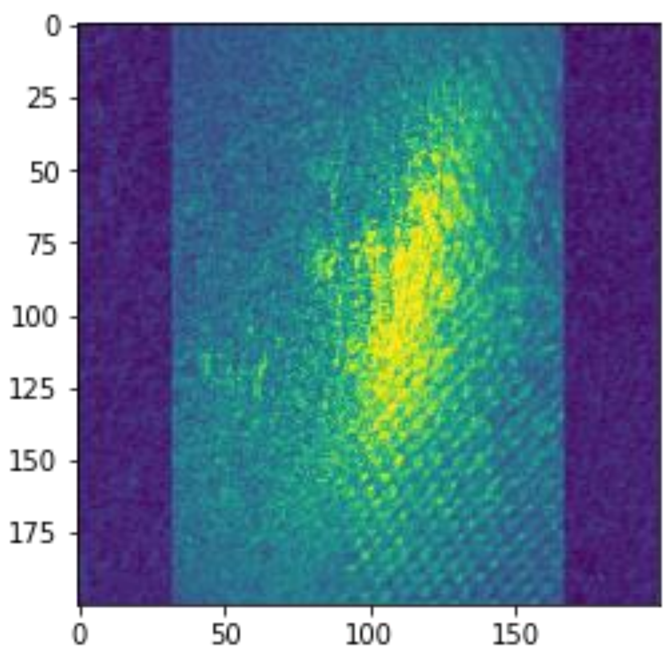
```
###################################################################
Training Data
Data Directory List 1- >  ['COIN', 'SCRAP']
(array([0, 1]), array([400, 921]))
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:151:
UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
Train on 1056 samples, validate on 265 samples

Epoch 1/20
1056/1056 [==============================] - 32s 31ms/step - loss: 0.6469 -
acc: 0.7472 - val_loss: 0.3725 - val_acc: 0.8415
Epoch 2/20
1056/1056 [==============================] - 29s 27ms/step - loss: 0.3794 -
acc: 0.8456 - val_loss: 0.3332 - val_acc: 0.8642
Epoch 3/20
1056/1056 [==============================] - 30s 29ms/step - loss: 0.3483 -
acc: 0.8570 - val_loss: 0.3087 - val_acc: 0.8717
Epoch 4/20
1056/1056 [==============================] - 29s 27ms/step - loss: 0.3189 -
acc: 0.8684 - val_loss: 0.2901 - val_acc: 0.8717
Epoch 5/20
1056/1056 [==============================] - 31s 29ms/step - loss: 0.3114 -
acc: 0.8712 - val_loss: 0.2802 - val_acc: 0.8755
Epoch 6/20
1056/1056 [==============================] - 28s 26ms/step - loss: 0.2918 -
acc: 0.8835 - val_loss: 0.2643 - val_acc: 0.8755
Epoch 7/20
1056/1056 [==============================] - 31s 29ms/step - loss: 0.2848 -
acc: 0.8835 - val_loss: 0.2550 - val_acc: 0.8755
Epoch 8/20
1056/1056 [==============================] - 27s 26ms/step - loss: 0.2630 -
acc: 0.8864 - val_loss: 0.2438 - val_acc: 0.8755
Epoch 9/20
1056/1056 [==============================] - 30s 28ms/step - loss: 0.2399 -
acc: 0.9006 - val_loss: 0.2420 - val_acc: 0.8717
Epoch 10/20
1056/1056 [==============================] - 28s 26ms/step - loss: 0.2207 -
acc: 0.9091 - val_loss: 0.2391 - val_acc: 0.8792
Epoch 11/20
1056/1056 [==============================] - 30s 28ms/step - loss: 0.1931 -
acc: 0.9271 - val_loss: 0.1925 - val_acc: 0.9094
Epoch 12/20
```

```
1056/1056 [==============================] - 28s 27ms/step - loss: 0.1836 -
acc: 0.9261 - val_loss: 0.1846 - val_acc: 0.9094
Epoch 13/20
1056/1056 [==============================] - 29s 28ms/step - loss: 0.1732 -
acc: 0.9356 - val_loss: 0.1762 - val_acc: 0.9132
Epoch 14/20
1056/1056 [==============================] - 30s 28ms/step - loss: 0.1507 -
acc: 0.9470 - val_loss: 0.1568 - val_acc: 0.9321
Epoch 15/20
1056/1056 [==============================] - 30s 28ms/step - loss: 0.1497 -
acc: 0.9403 - val_loss: 0.1613 - val_acc: 0.9245
Epoch 16/20
1056/1056 [==============================] - 28s 27ms/step - loss: 0.1366 -
acc: 0.9489 - val_loss: 0.1594 - val_acc: 0.9283
Epoch 17/20
1056/1056 [==============================] - 29s 27ms/step - loss: 0.1357 -
acc: 0.9479 - val_loss: 0.1660 - val_acc: 0.9321
Epoch 18/20
1056/1056 [==============================] - 28s 27ms/step - loss: 0.1352 -
acc: 0.9470 - val_loss: 0.1704 - val_acc: 0.9132
Epoch 19/20
1056/1056 [==============================] - 29s 28ms/step - loss: 0.1379 -
acc: 0.9498 - val_loss: 0.1606 - val_acc: 0.9208
Epoch 20/20
1056/1056 [==============================] - 30s 28ms/step - loss: 0.1164 -
acc: 0.9602 - val_loss: 0.1433 - val_acc: 0.9208
265/265 [==============================] - 1s 5ms/step
```

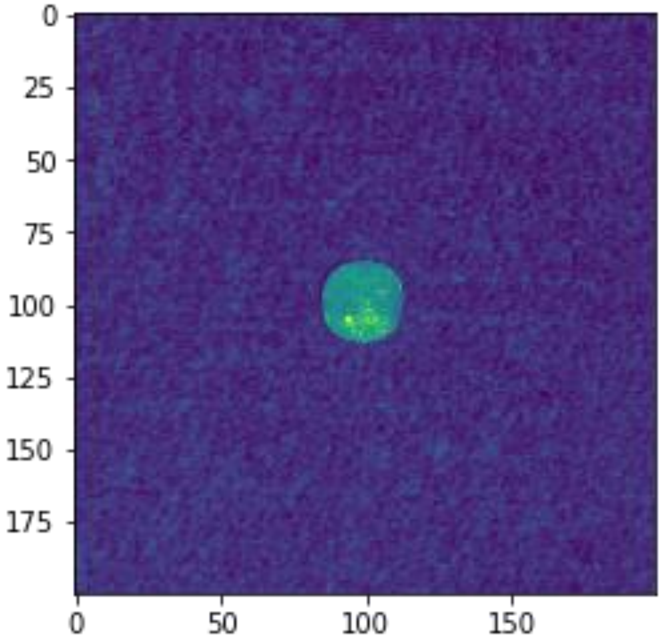
Validation accuracy - > 92.07547174309785

Training Images:

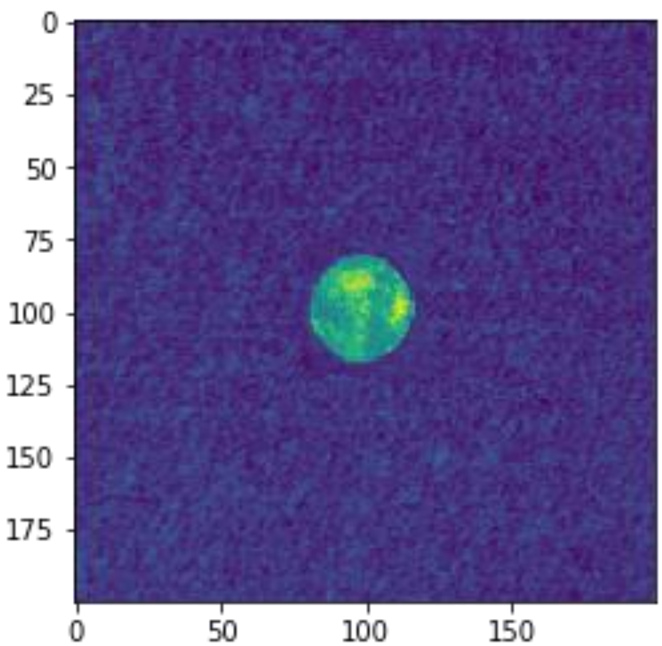
The Predicted Validation image is =SCRAP verify below



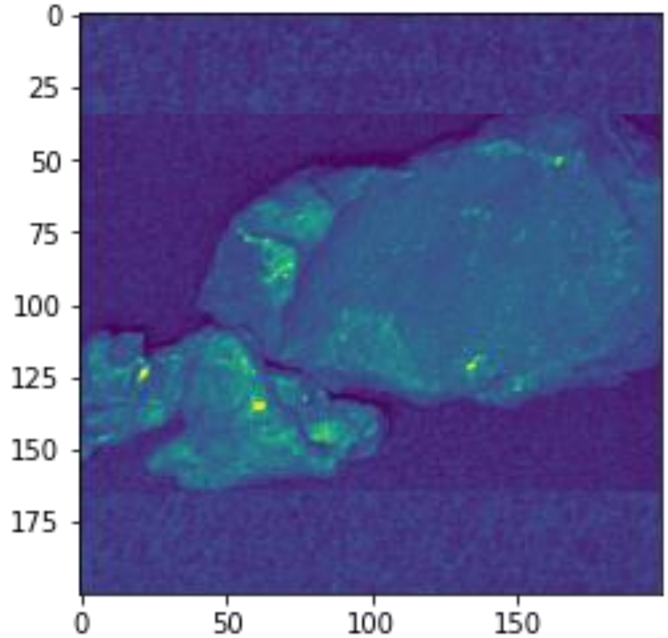
The Predicted Validation image is =COIN verify below



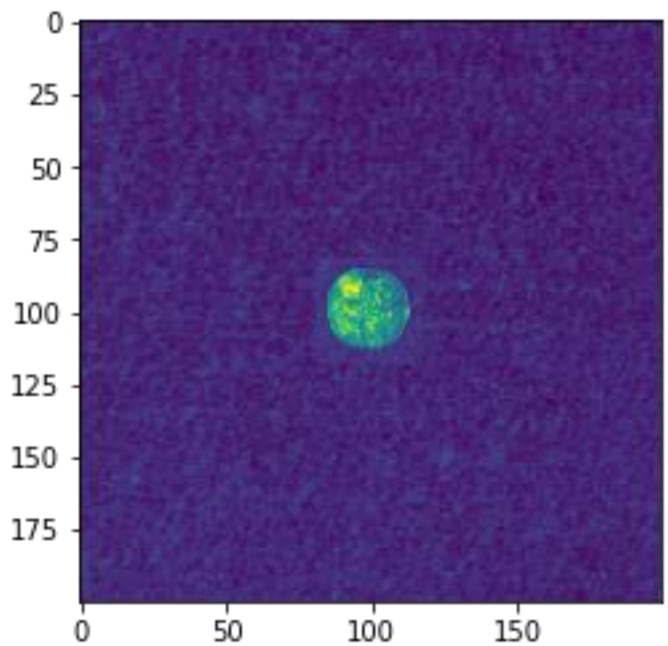
The Predicted Validation image is =COIN verify below



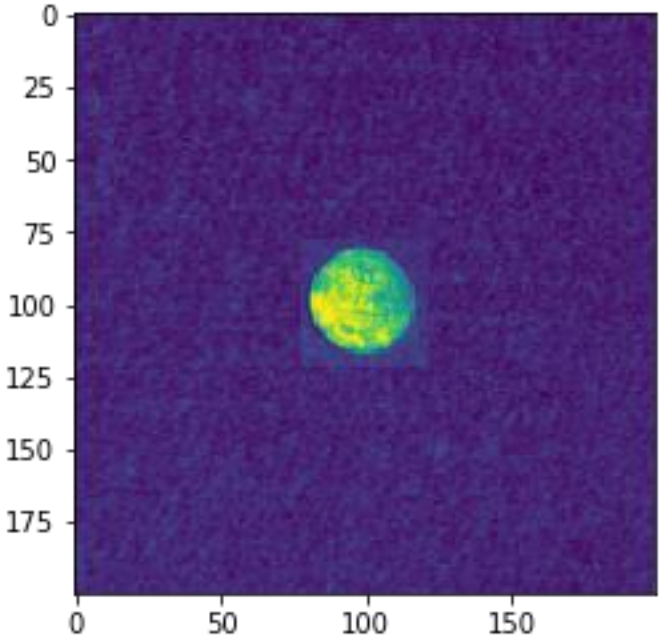
The Predicted Validation image is =SCRAP verify below



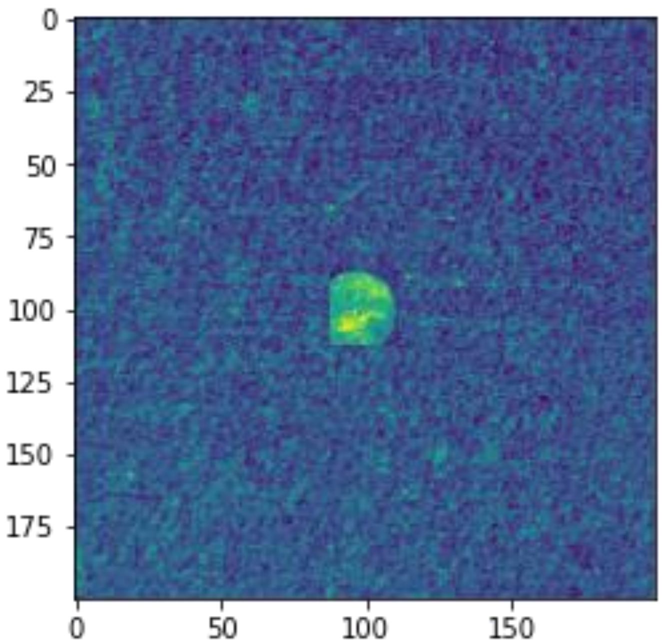
The Predicted Validation image is =COIN verify below



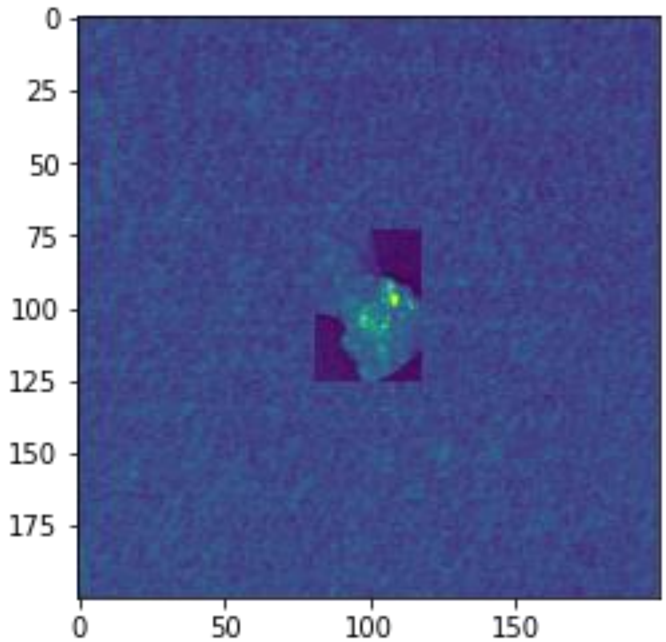
The Predicted Validation image is =COIN verify below



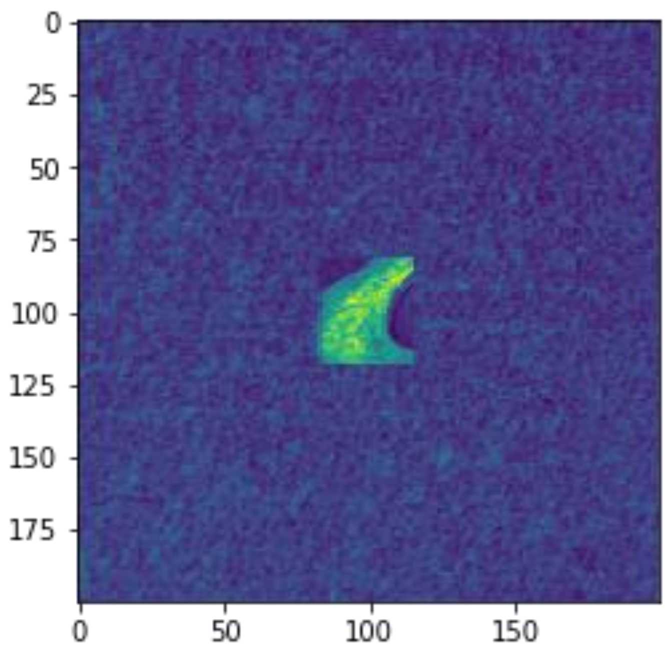
The Predicted Validation image is =COIN verify below



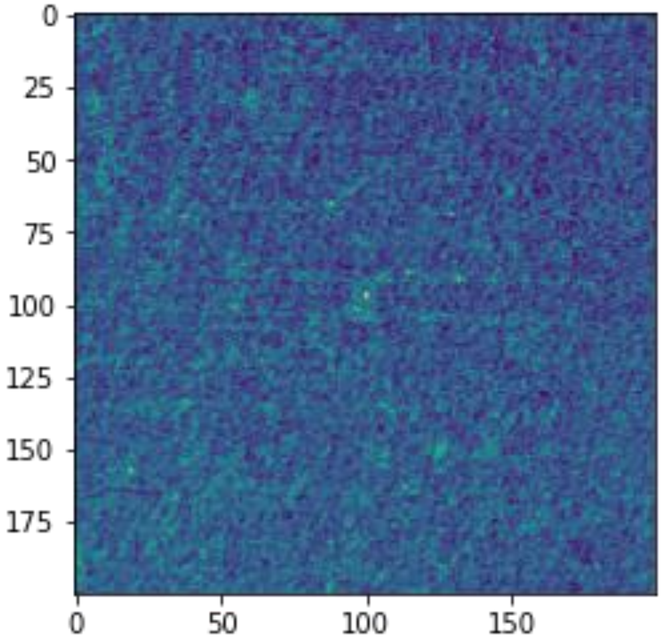
The Predicted Validation image is =COIN verify below



The Predicted Validation image is =SCRAP verify below



The Predicted Validation image is =COIN verify below



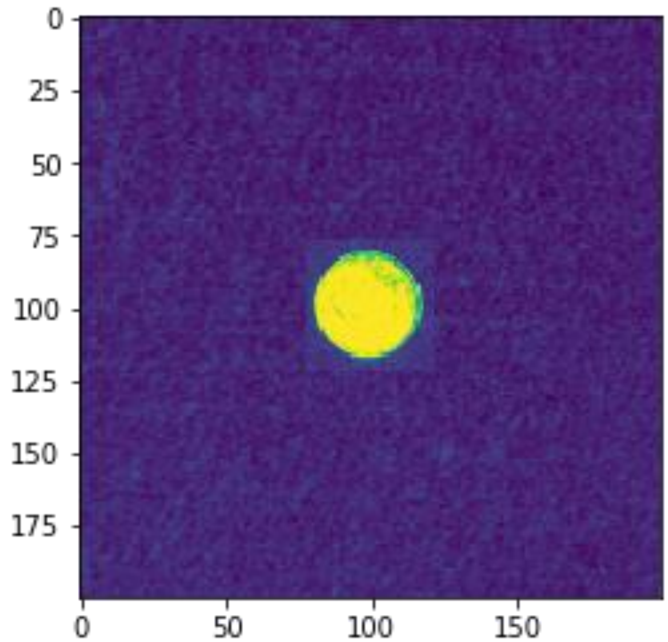
Testing Results:

```
###################################################################
Testing data
Data Directory List 2- >  ['COIN', 'SCRAP']
396/396 [==============================] - 2s 4ms/step
```

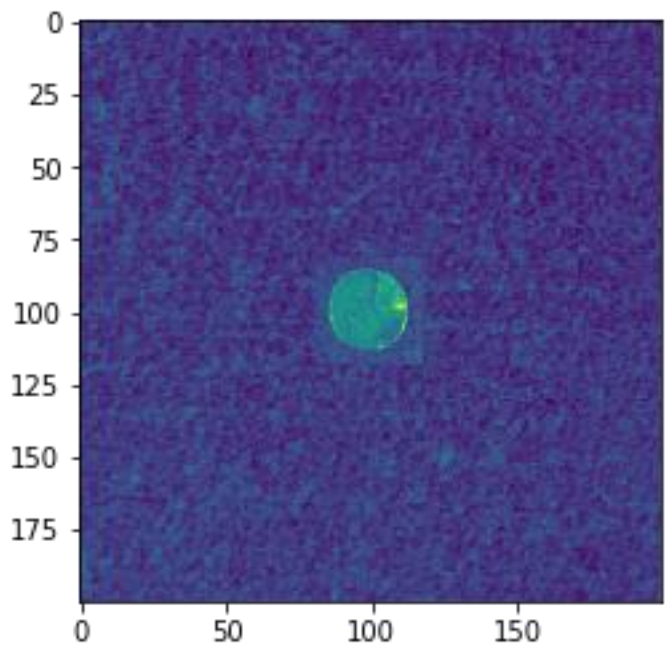
Testing accuracy - > 95.70707070707071

Testing Images:

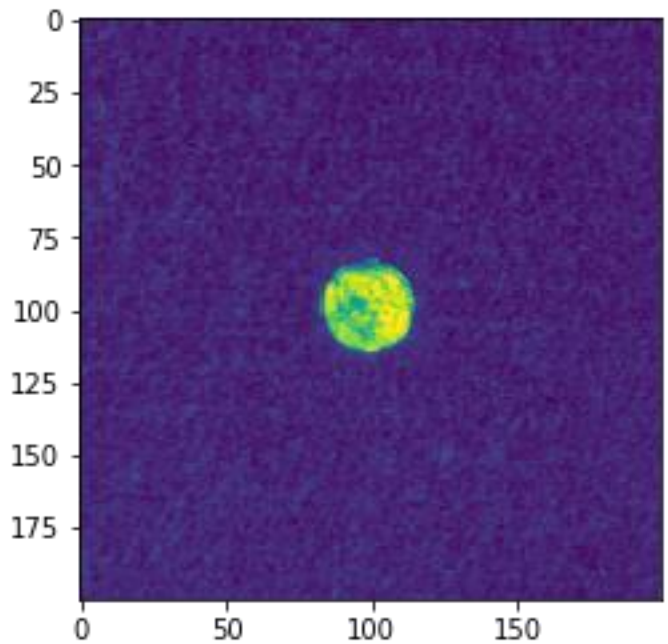
```
The Predicted Testing image is =COIN verify below
```



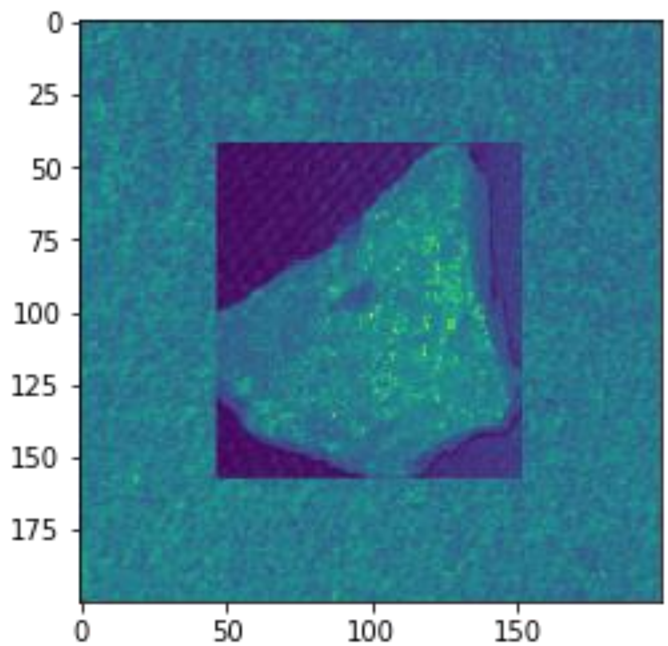
The Predicted Testing image is =COIN verify below



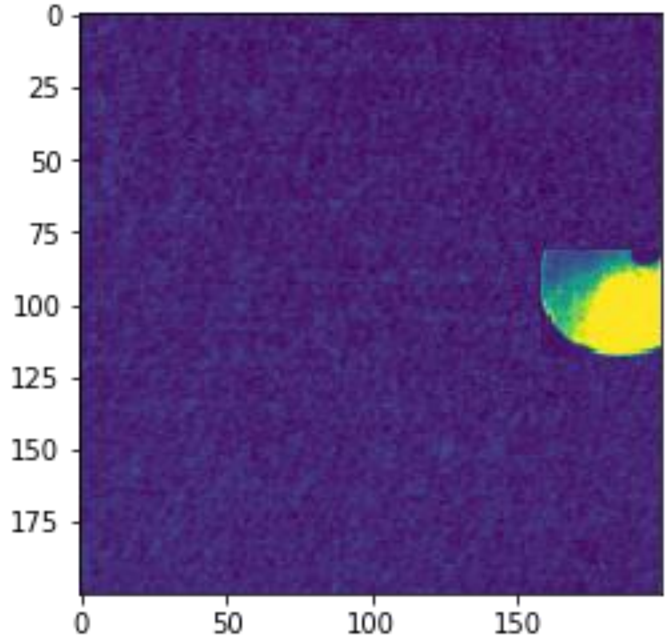
The Predicted Testing image is =COIN verify below



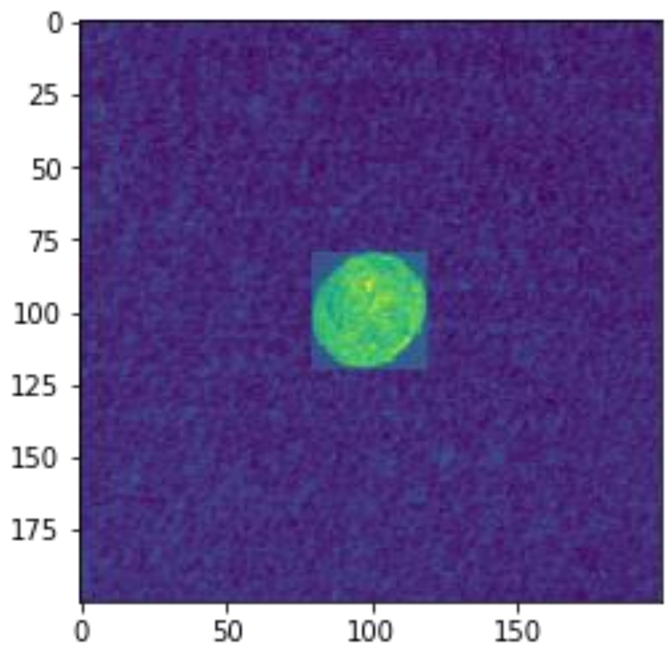
The Predicted Testing image is =SCRAP verify below



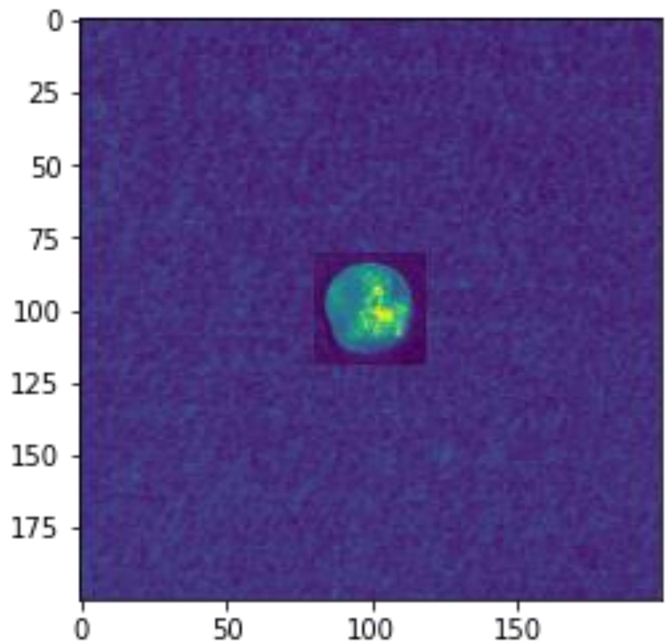
The Predicted Testing image is =SCRAP verify below



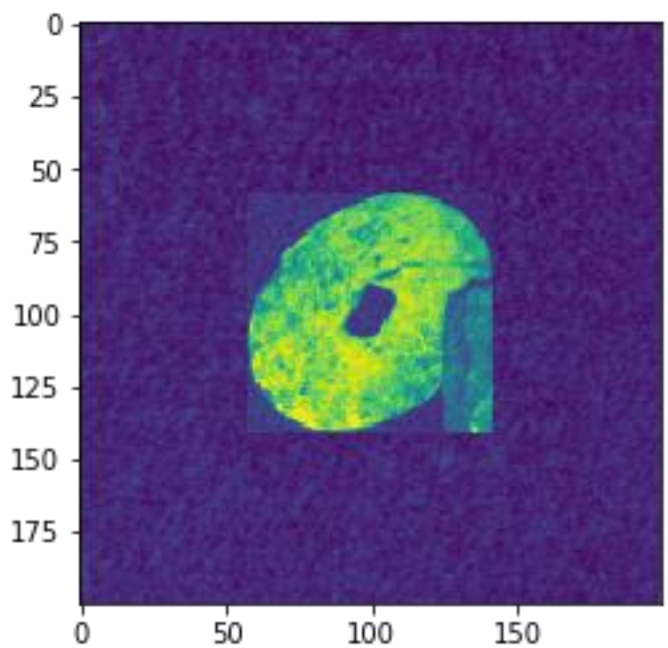
The Predicted Testing image is =COIN verify below



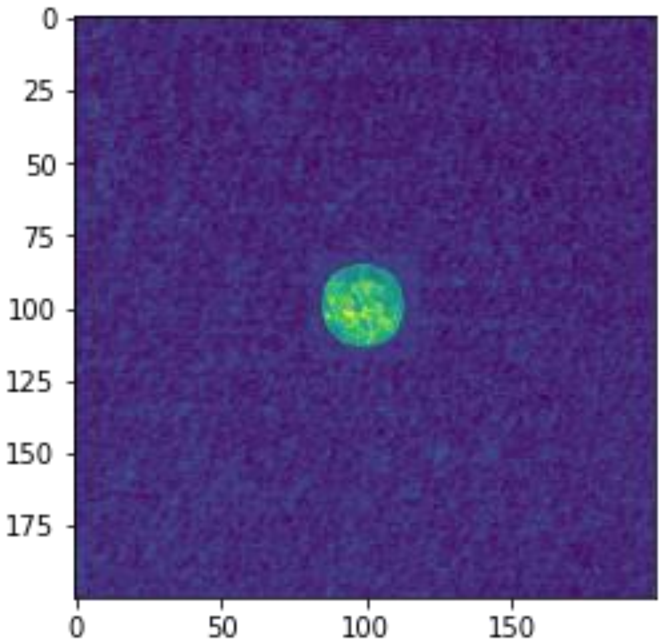
The Predicted Testing image is =COIN verify below



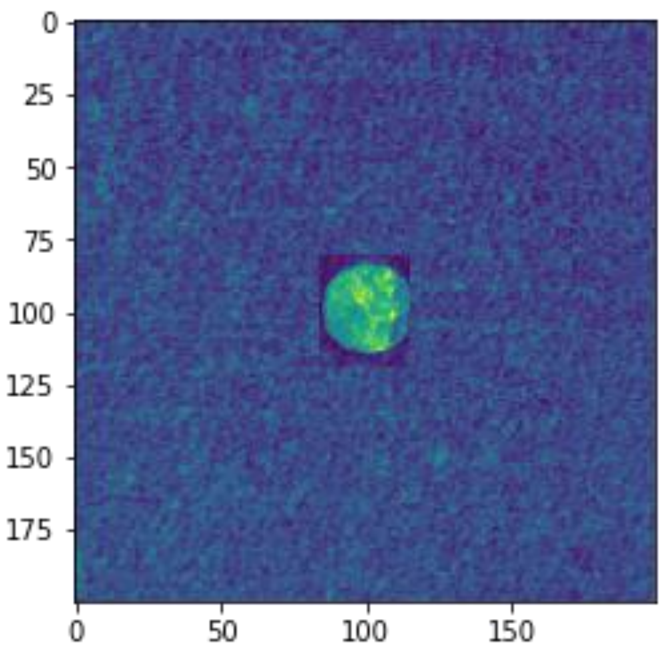
The Predicted Testing image is =SCRAP verify below



The Predicted Testing image is =COIN verify below



The Predicted Testing image is =COIN verify below



Part II CNN with augmentation

Part II Training Results:

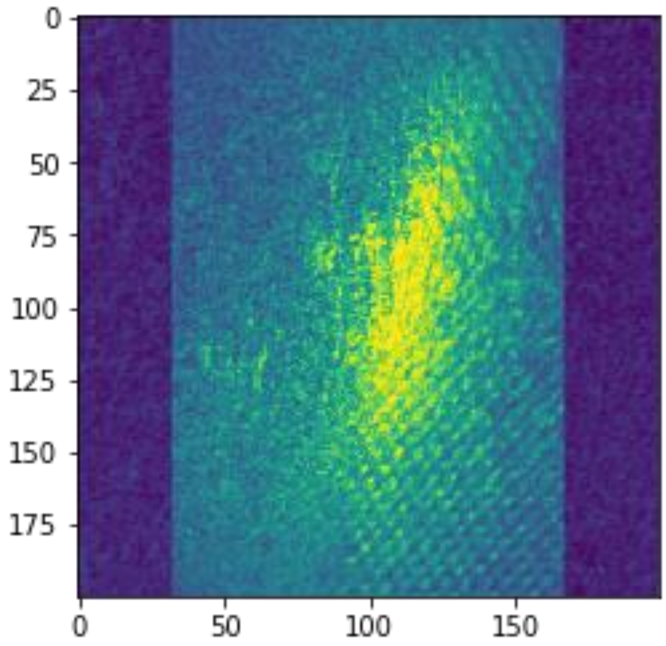
```
#################################################################
Training Data
Data Directory List 1- >  ['COIN', 'SCRAP']
(array([0, 1]), array([400, 921]))
```

```
Epoch 1/20
33/33 [==============================] - 33s 1s/step - loss: 0.7382 - acc:
0.7121 - val_loss: 0.4349 - val_acc: 0.8340
Epoch 2/20
33/33 [==============================] - 31s 952ms/step - loss: 0.4119 - acc:
0.8324 - val_loss: 0.3591 - val_acc: 0.8453
Epoch 3/20
33/33 [==============================] - 30s 917ms/step - loss: 0.3883 - acc:
0.8447 - val_loss: 0.3363 - val_acc: 0.8528
Epoch 4/20
33/33 [==============================] - 31s 933ms/step - loss: 0.3783 - acc:
0.8438 - val_loss: 0.3225 - val_acc: 0.8604
Epoch 5/20
33/33 [==============================] - 31s 938ms/step - loss: 0.3520 - acc:
0.8674 - val_loss: 0.3102 - val_acc: 0.8642
Epoch 6/20
33/33 [==============================] - 33s 985ms/step - loss: 0.3554 - acc:
0.8570 - val_loss: 0.3067 - val_acc: 0.8679
Epoch 7/20
33/33 [==============================] - 31s 953ms/step - loss: 0.3424 - acc:
0.8636 - val_loss: 0.2728 - val_acc: 0.8717
Epoch 8/20
33/33 [==============================] - 33s 1s/step - loss: 0.3310 - acc:
0.8665 - val_loss: 0.2730 - val_acc: 0.8717
Epoch 9/20
33/33 [==============================] - 31s 949ms/step - loss: 0.3346 - acc:
0.8712 - val_loss: 0.2956 - val_acc: 0.8604
Epoch 10/20
33/33 [==============================] - 32s 959ms/step - loss: 0.3720 - acc:
0.8494 - val_loss: 0.2873 - val_acc: 0.8679
Epoch 11/20
33/33 [==============================] - 30s 923ms/step - loss: 0.3132 - acc:
0.8788 - val_loss: 0.2435 - val_acc: 0.8868
Epoch 12/20
33/33 [==============================] - 33s 990ms/step - loss: 0.3021 - acc:
0.8854 - val_loss: 0.2994 - val_acc: 0.8943
Epoch 13/20
33/33 [==============================] - 30s 908ms/step - loss: 0.3031 - acc:
0.8731 - val_loss: 0.2307 - val_acc: 0.8943
Epoch 14/20
33/33 [==============================] - 32s 966ms/step - loss: 0.2891 - acc:
0.8845 - val_loss: 0.2769 - val_acc: 0.8792
Epoch 15/20
33/33 [==============================] - 30s 924ms/step - loss: 0.2941 - acc:
0.8731 - val_loss: 0.2188 - val_acc: 0.8981
Epoch 16/20
33/33 [==============================] - 32s 963ms/step - loss: 0.2630 - acc:
0.8958 - val_loss: 0.1983 - val_acc: 0.9132
Epoch 17/20
33/33 [==============================] - 30s 922ms/step - loss: 0.2504 - acc:
0.8996 - val_loss: 0.2057 - val_acc: 0.9132
Epoch 18/20
33/33 [==============================] - 32s 958ms/step - loss: 0.2537 - acc:
0.8920 - val_loss: 0.2217 - val_acc: 0.9358
Epoch 19/20
33/33 [==============================] - 31s 939ms/step - loss: 0.2508 - acc:
0.9034 - val_loss: 0.2060 - val_acc: 0.9057
```

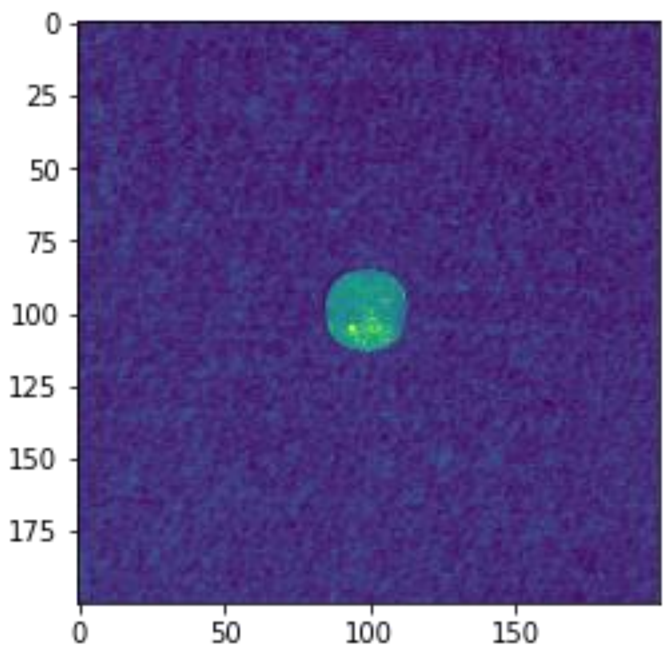
```
Epoch 20/20
33/33 [==============================] - 31s 942ms/step - loss: 0.2457 - acc: 
0.9025 - val_loss: 0.2016 - val_acc: 0.9358
265/265 [==============================] - 1s 4ms/step
Validation accuracy - >  93.58490568286967
```

Part II Training Images:

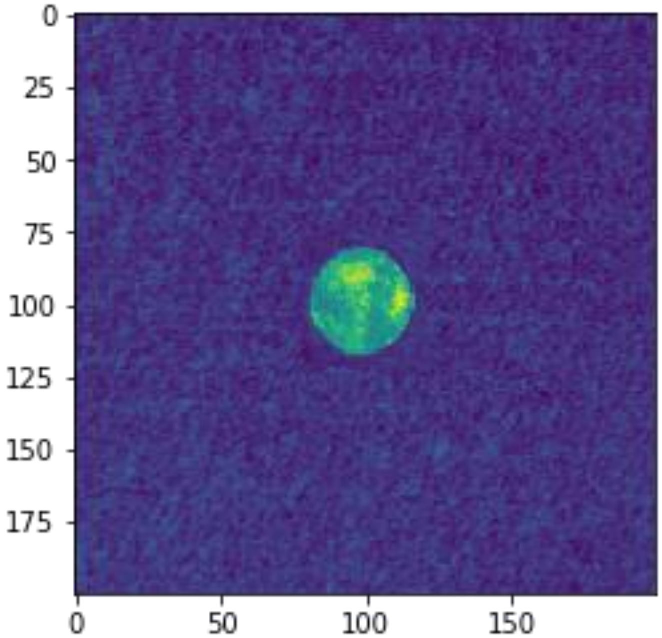
```
The Predicted Validation image is =SCRAP verify below
```



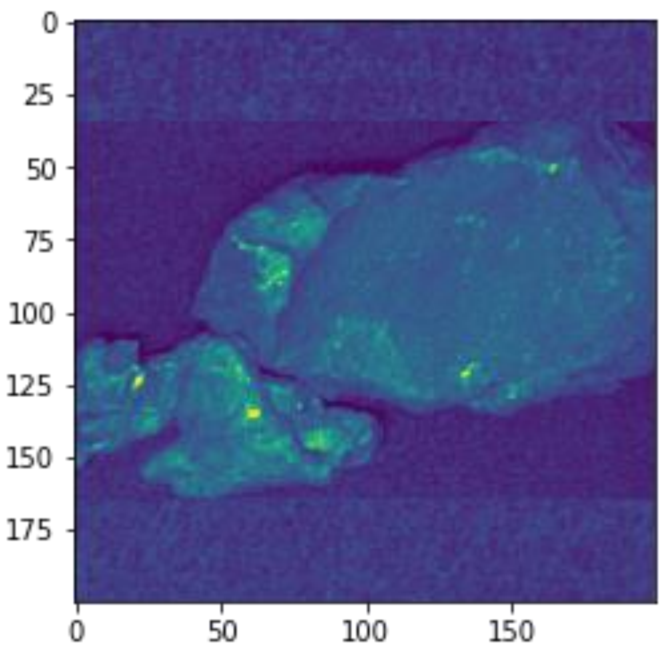

```
The Predicted Validation image is =COIN verify below
```



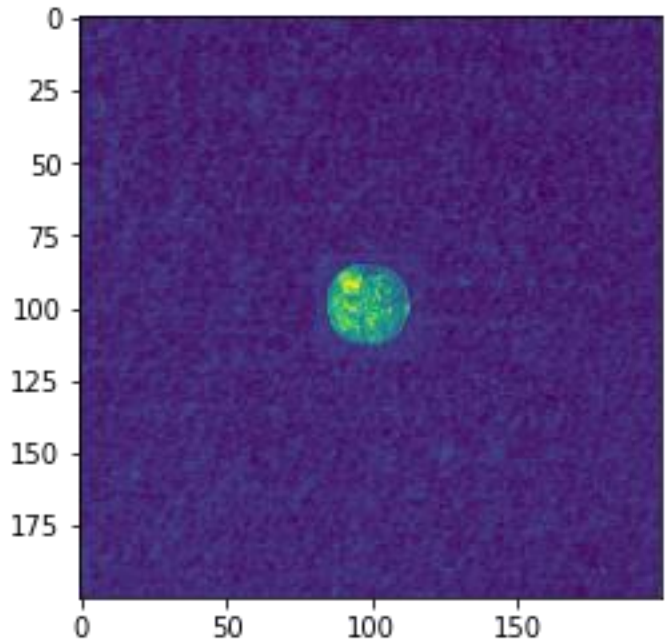

```
The Predicted Validation image is =COIN verify below
```



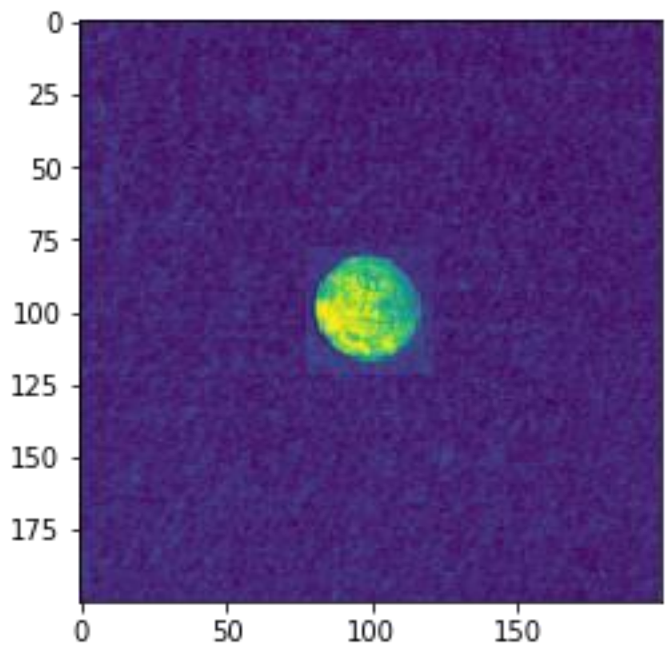
The Predicted Validation image is =SCRAP verify below



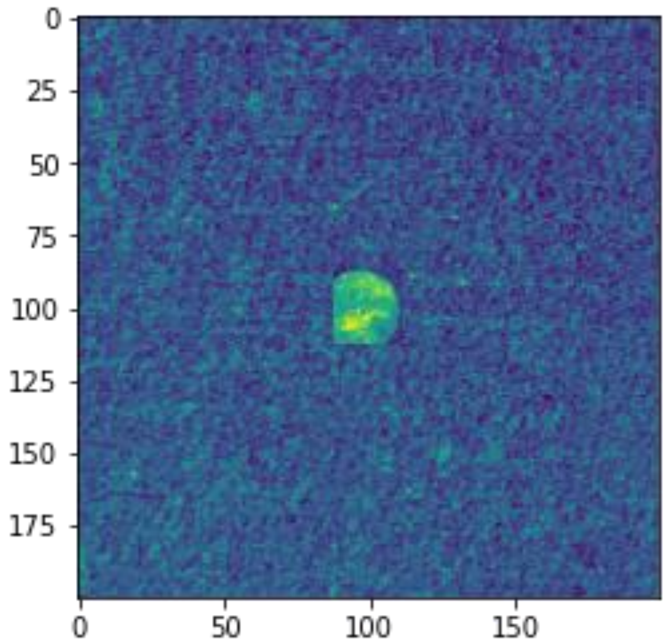
The Predicted Validation image is =COIN verify below



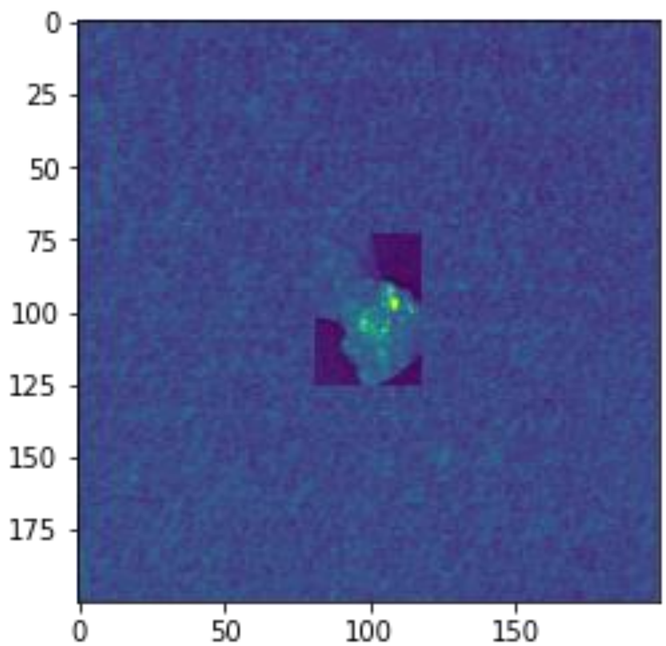
The Predicted Validation image is =COIN verify below



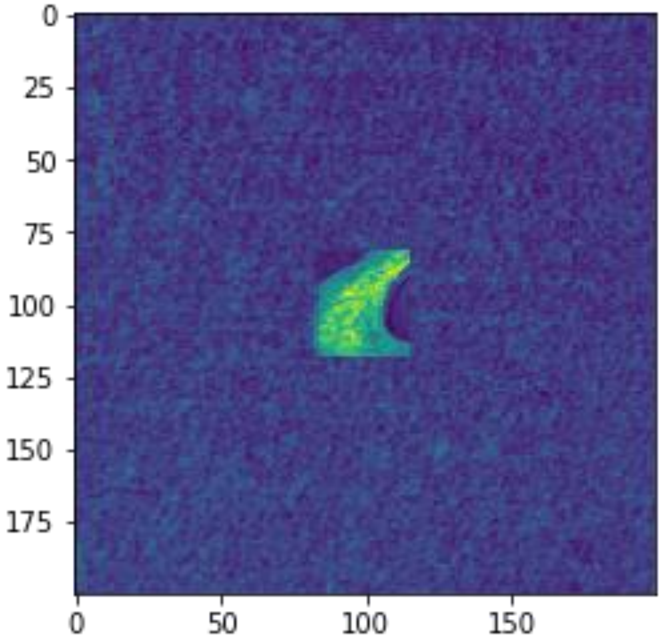
The Predicted Validation image is =COIN verify below



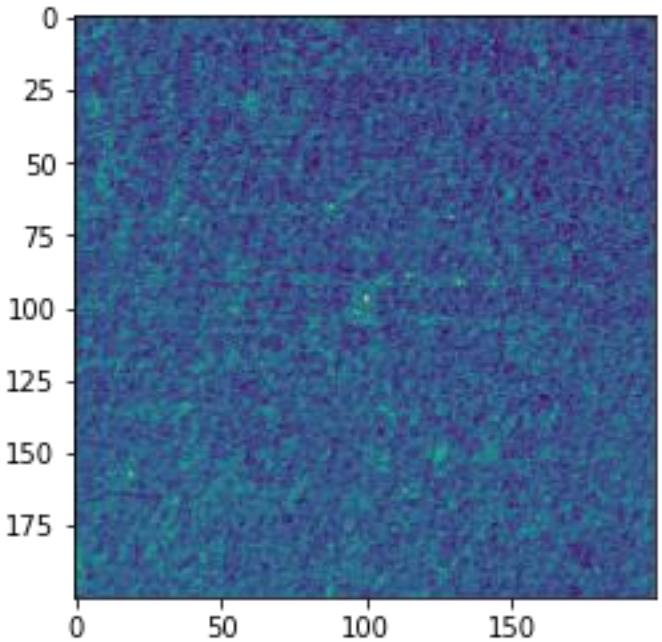
The Predicted Validation image is =SCRAP verify below



The Predicted Validation image is =SCRAP verify below




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



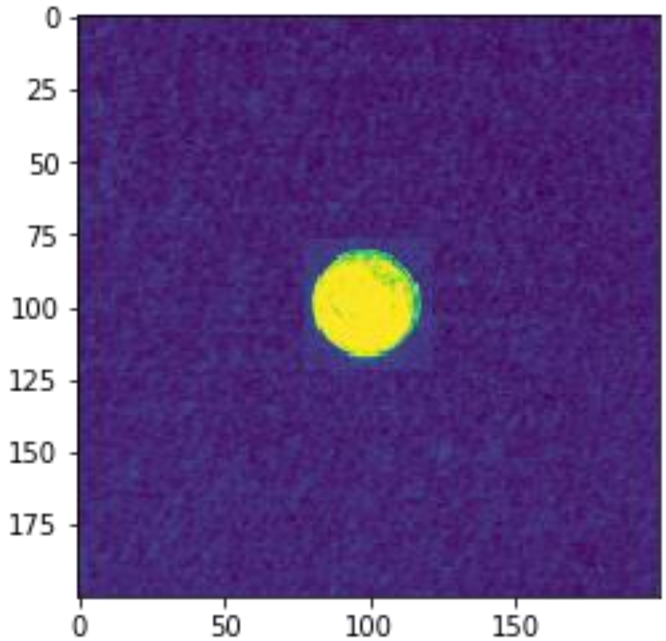
Part II Testing Results:

```
#####################################################################
Testing data
Data Directory List 2- >  ['COIN', 'SCRAP']
(array([0, 1]), array([120, 276]))
396/396 [==============================] - 2s 4ms/step
```

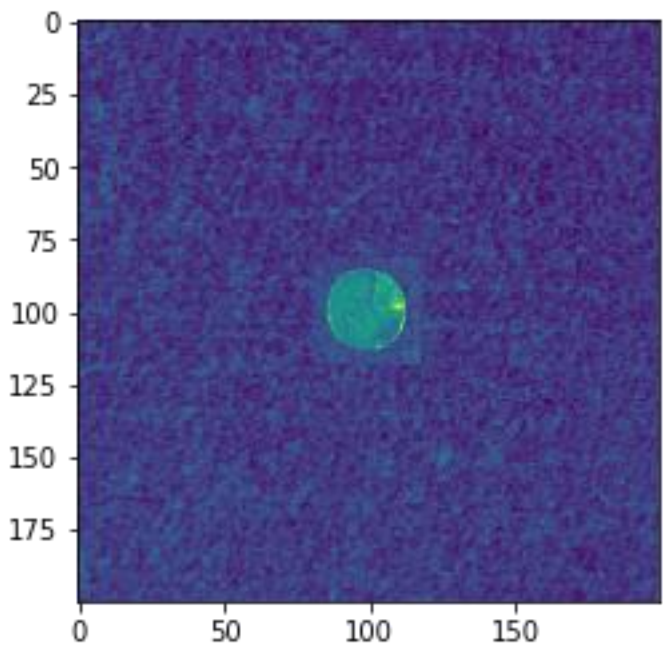
Testing accuracy - > 92.92929292929293

Testing Images:

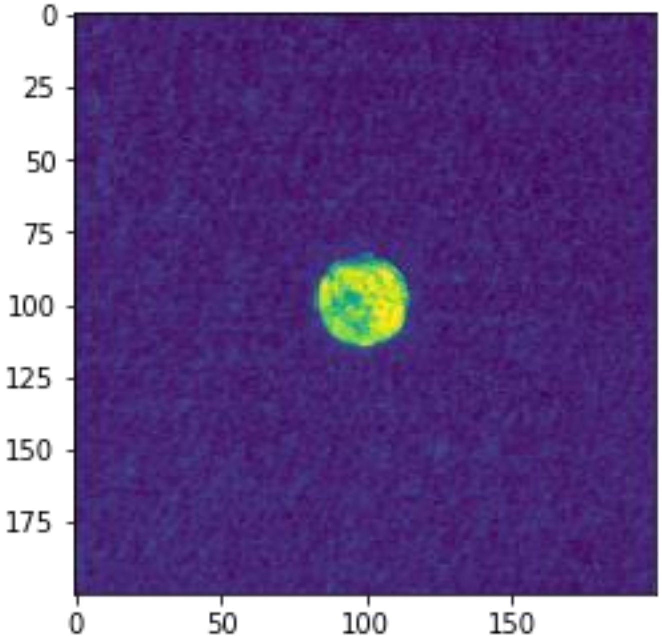
The Predicted Testing image is =COIN verify below



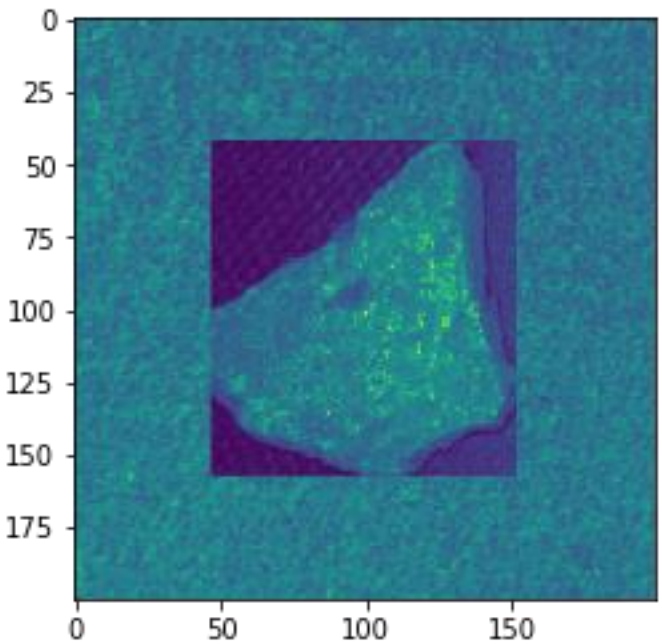
The Predicted Testing image is =COIN verify below



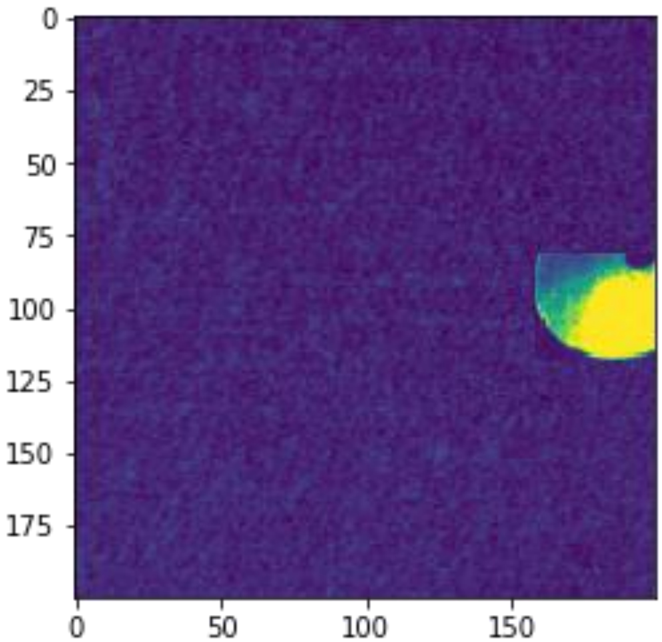
The Predicted Testing image is =COIN verify below



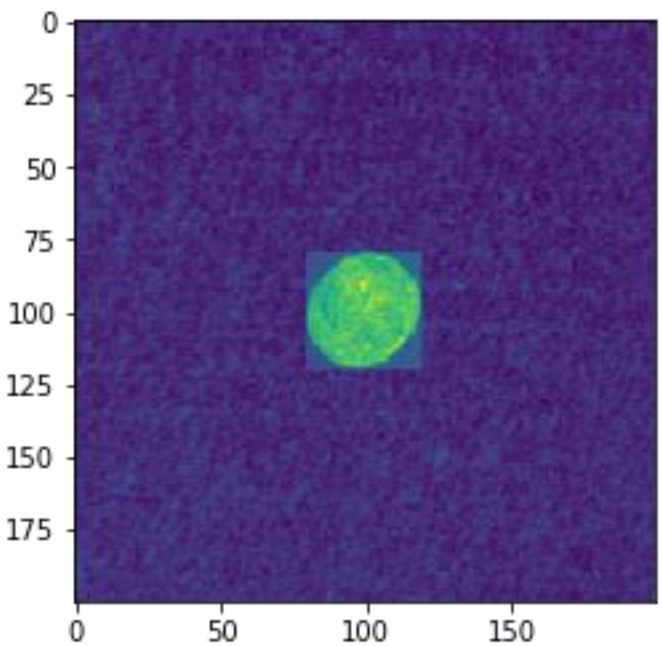
The Predicted Testing image is =SCRAP verify below



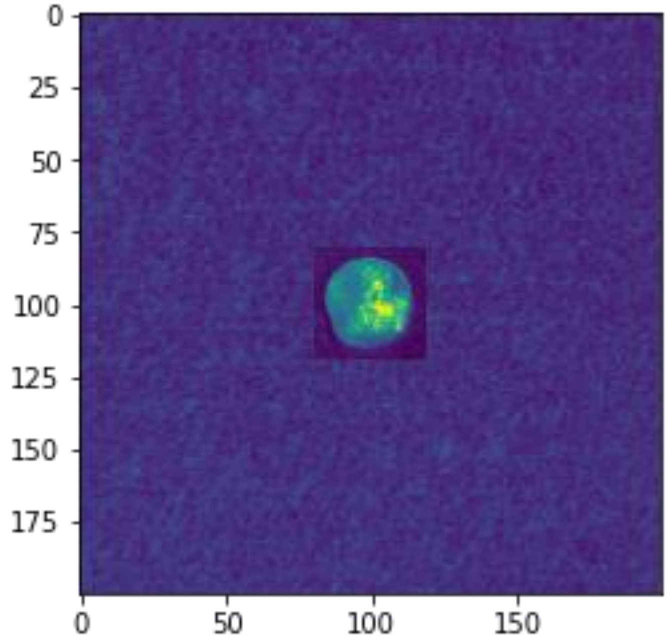
The Predicted Testing image is =COIN verify below



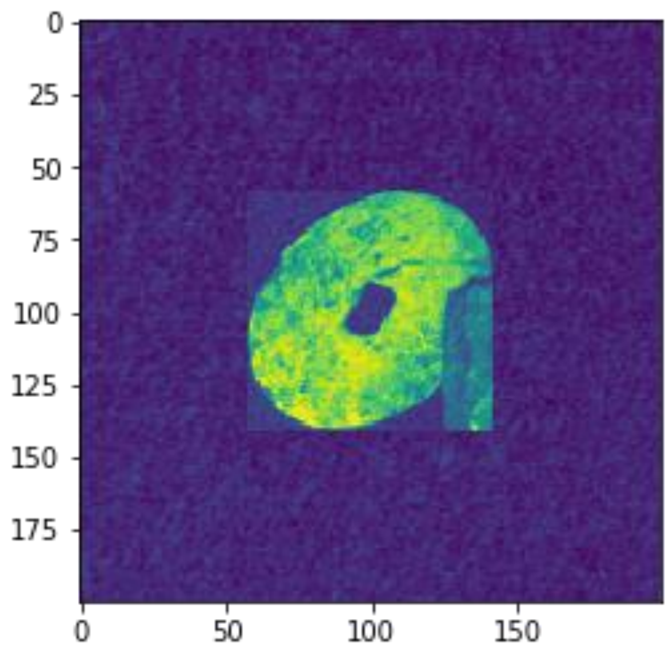
The Predicted Testing image is =COIN verify below



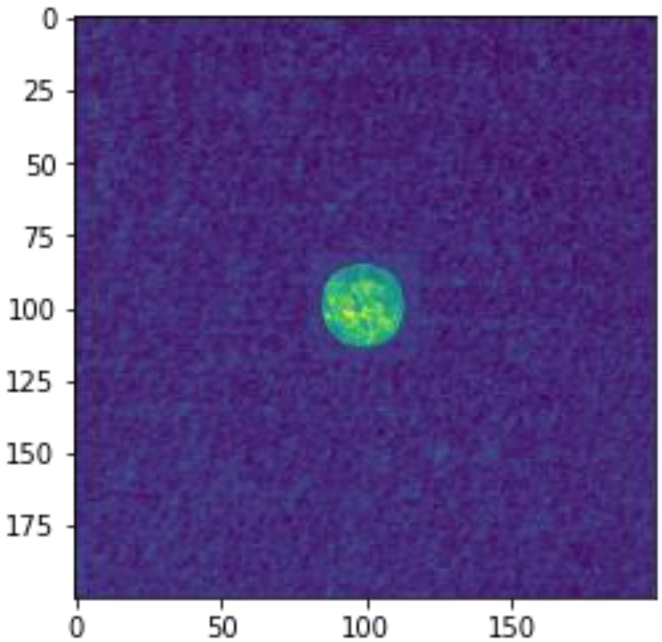
The Predicted Testing image is =COIN verify below



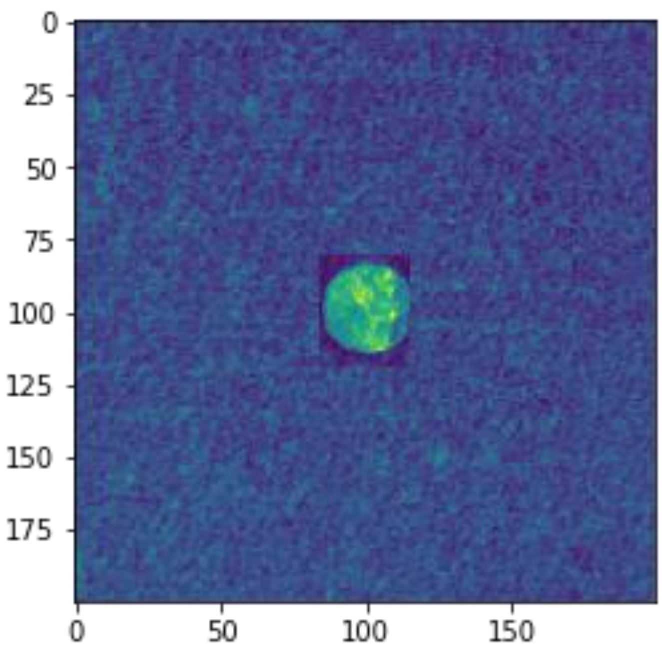
The Predicted Testing image is =SCRAP verify below



The Predicted Testing image is =COIN verify below




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



Input Output Processing Explanation:

Step 1: Import data
Step 2: Define number of classes
Step 3:
For data set in directory

For images in image list
Read image
Change color space
Resize
Append label
End
End
Step 4: Output unique labels
Step 5: Shuffle dataset
Step 6: Divide data into train & test
Step 7: Normalize data
Step 8: Reshape data to fit model
Step 9: Add convolutionary, max pooling, and dropout layers
Step 10: Compile Model
Step 11: Fit model on training Data
Step 12: Evaluate model on test data
Step 13: Test accuracy
Step 14: Print images vs predicted

Code:

```
# -*- coding: utf-8 -*-
"""
Created on Thur Nov 7 14:20:37 2019
Reference from https://github.com/anujshah1003/own_data_cnn_implementation_ke
ras/blob/master/updated_custom_data_cnn.py
"""
import numpy as np
import tensorflow as tf
import random as rn
import os,cv2
import glob
import re
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from keras.utils import np_utils
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
from keras import backend as K
from google.colab import drive

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.random.set_seed(89)
session_conf = tf.compat.v1.ConfigProto(
intra_op_parallelism_threads=1,
inter_op_parallelism_threads=1)
#Force Tensorflow to use a single thread
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=se
ssion_conf)
#K.set_session(sess)
tf.compat.v1.keras.backend.set_session(sess)

# Define Functions
def sorted_aphanumeric(data):
    convert = lambda text: int(text) if text.isdigit() else text.lower()
    alphanum_key = lambda key: [ convert(c) for c in re.split('([0-
9]+)', key) ]
    return sorted(data, key=alphanum_key)

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

# Mount google drive
drive.mount('/content/drive')
PATH = os.getcwd()

print('')
print('###################################################################')
print('Training Data ')

# Define data path
```

```
data_path1 = '/content/drive/My Drive/Colab Notebooks/Program8/Training'
data_path2 = '/content/drive/My Drive/Colab Notebooks/Program8/Testing'

data_dir_list1 = sorted_aphanumeric(os.listdir(data_path1))
data_dir_list2 = sorted_aphanumeric(os.listdir(data_path2))

print('Data Directory List 1- > ',data_dir_list1)
print('Data Directory List 2- > ',data_dir_list2)

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

# Define the number of classes
num_classes = 2
labels_name={'SCRAP':0,'COIN':1}

img_data_list1=[]
labels_list1 = []

img_data_list2=[]
labels_list2 = []

# Read training data
for dataset in data_dir_list1:
    img_list = glob.glob(data_path1+'/'+ 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_list1.append(input_img_resize)
        labels_list1.append(label)

#print(unique(labels_list1))
img_data1 = np.array(img_data_list1)
img_data1 = img_data1.astype('float32')

labels1 = np.array(labels_list1)

#print(unique(labels1))
print(np.unique(labels1,return_counts=True))
Y1 = np_utils.to_categorical(labels1, num_classes)
```

```
#Shuffle the dataset
x,y = shuffle(img_data1,Y1, random_state=2)

X_train, X_validation, y_train, y_validation = train_test_split(x, y, test_si
ze=0.2, random_state=2) # divide data into train and test

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

Nv = X_train.shape[0]
Nv_validation = X_validation.shape[0]

#reshape data to fit model
X_train = X_train.reshape(int(Nv),200,200,1)
X_validation = X_validation.reshape(int(Nv_validation),200,200,1)

model = Sequential()

##################   add model layers   #######################
model.add(Conv2D(64, kernel_size=(3,3), strides = 2, activation='relu',input_
shape=(200,200,1)))
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(num_classes, activation='softmax'))

############################################################################
###########

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

# Fit model on training data
#model.fit(X_train, y_train, batch_size=32, nb_epoch=num_epoch, verbose=1,shu
ffle=False, validation_data = (X_validation, y_validation))

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=len(X_train)//32,
                    epochs=20,validation_data=(X_validation, y_validation),ve
rbose=1)

# Evaluate model on test data
score1 = model.evaluate(X_validation, y_validation, verbose=1)
print('Validation accuracy - > ',score1[1] * 100)

# Print image vs predicated image
ytested1 = model.predict_classes(X_validation)
for i in range(10):
  print("The Predicted Validation image is =%s verify below" % ((list(labels_
name.keys())[list(labels_name.values()).index(ytested1[i])])))
  gen_image(X_validation[i]).show() # printing image vs the predicted image b
elow

##################################################################
# Read testing data
print('')
print('##################################################################')
print('Testing data ')
print('Data Directory List 2- > ',data_dir_list2)

for dataset in data_dir_list2:
    img_list = glob.glob(data_path2+'/'+ 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_list2.append(input_img_resize)
        labels_list2.append(label)

#print(unique(labels_list2))
img_data2 = np.array(img_data_list2)
img_data2 = img_data2.astype('float32')

labels2 = np.array(labels_list2)

#print(unique(labels2))
print(np.unique(labels2,return_counts=True))
```

```
Y2 = np_utils.to_categorical(labels2, num_classes)

#Shuffle the dataset
X_test,Y_test = shuffle(img_data2,Y2, random_state=2)

#Normalization of the data
X_test = X_test/255
Nv = X_test.shape[0]

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

score2 = model.evaluate(X_test, Y_test, verbose=1)
print('Testing accuracy - > ',score2[1] * 100)

 # Input Testing Images
ytested2 = model.predict_classes(X_test)
for i in range(10):
  print("The Predicted Testing image is =%s verify below" % ((list(labels_nam
e.keys())[list(labels_name.values()).index(ytested2[i])])))
  gen_image(X_test[i]).show() # printing image vs the predicted image below
```

Conclusion:

From this programing exercise I have learned that a convolutional neural net can capture spatial and temporal dependencies of an image through the application of filters. One of the goals of convolutional neural nets is to reduce the image into a form that is easier to process without losing features which are critical for a good prediction. This process is done with the use of the Kernel filter. This filter is also able to extract high-level features such as edges from the input image. The pooling layer is responsible for reducing the spatial size of the convolved feature. This helps decrease the computational power required to process the data. This layer is also useful for extracting dominant features which are rotational and positional invariant. The dropout layer refers to ignoring units during the training phase of certain neurons which are selected at random. By ignoring, I mean they are not considered during a particular forward or backward pass. We do this to prevent over-fitting.