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.

Procedure:

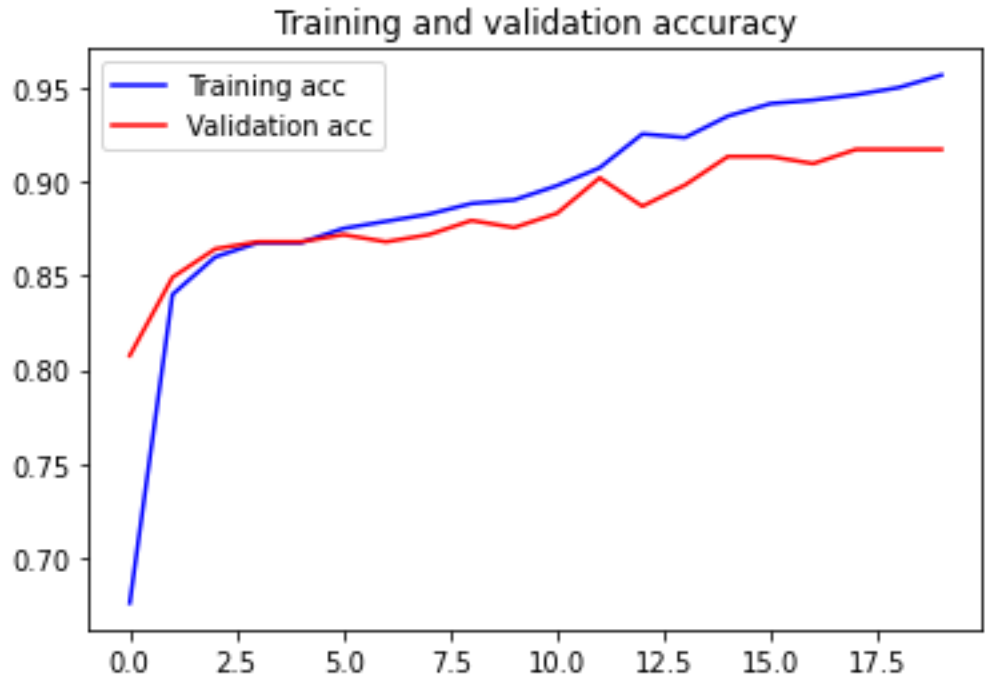
1. Input the images from the training folder in proper image and label format(use onehot encoding/to_categorical)
2. Divide the data into training (80%) and validation(20%).
3. Resize the images to 200 by 200.
4. Convert the images to black and white
5. Normalize the input data
6. Design a convolutional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

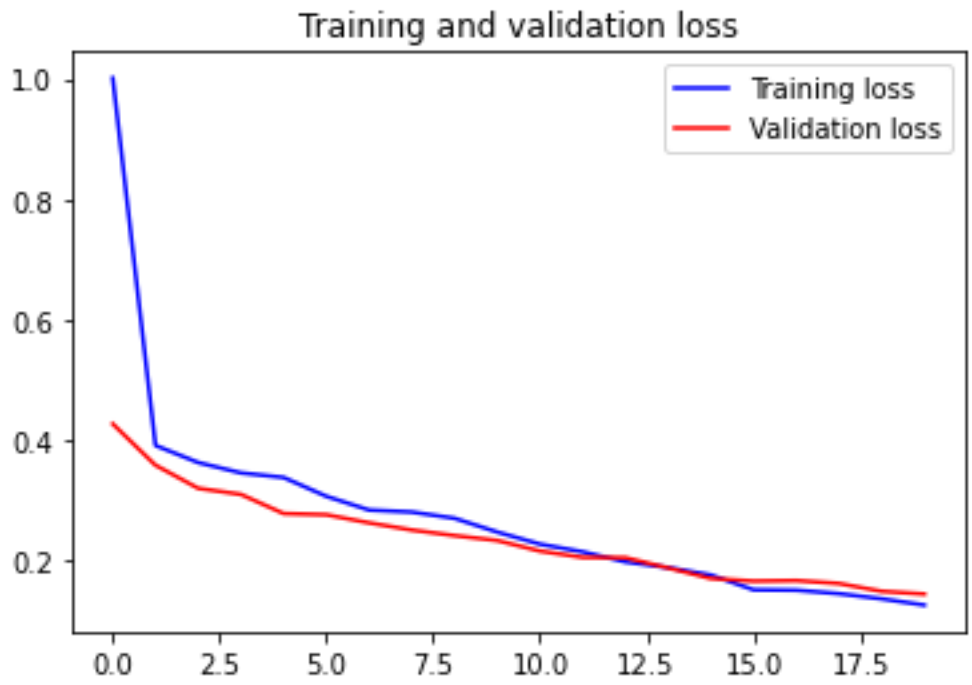
Part I CNN without Augmentation

Training Results:

```
################################################################
Load Datasets

Data Directory List 1- >  ['COIN', 'SCRAP']
Data Directory List 2- >  ['COIN', 'SCRAP']
(array([0, 1]), array([400, 921]))

################################################################
Create Model
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 99, 99, 64)        640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 49, 49, 64)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 49, 49, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 153664)            0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               19669120
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 258
=================================================================
Total params: 19,670,018
Trainable params: 19,670,018
Non-trainable params: 0
_________________________________________________________________

################################################################
Train Model

Epoch 1/20
33/33 [==============================] - 16s 448ms/step - loss: 1.4788 - accuracy: 0.6029 -
val_loss: 0.4264 - val_accuracy: 0.8075
Epoch 2/20
33/33 [==============================] - 14s 430ms/step - loss: 0.4046 - accuracy: 0.8288 -
val_loss: 0.3578 - val_accuracy: 0.8491
Epoch 3/20
33/33 [==============================] - 14s 438ms/step - loss: 0.3740 - accuracy: 0.8578 -
val_loss: 0.3194 - val_accuracy: 0.8642
Epoch 4/20
33/33 [==============================] - 15s 452ms/step - loss: 0.3435 - accuracy: 0.8636 -
val_loss: 0.3092 - val_accuracy: 0.8679
Epoch 5/20
33/33 [==============================] - 14s 438ms/step - loss: 0.3417 - accuracy: 0.8615 -
val_loss: 0.2774 - val_accuracy: 0.8679
Epoch 6/20
33/33 [==============================] - 15s 447ms/step - loss: 0.3071 - accuracy: 0.8672 -
val_loss: 0.2754 - val_accuracy: 0.8717
Epoch 7/20
33/33 [==============================] - 14s 439ms/step - loss: 0.2847 - accuracy: 0.8779 -
val_loss: 0.2620 - val_accuracy: 0.8679
Epoch 8/20
33/33 [==============================] - 14s 435ms/step - loss: 0.2737 - accuracy: 0.8749 -
val_loss: 0.2500 - val_accuracy: 0.8717
Epoch 9/20
33/33 [==============================] - 14s 436ms/step - loss: 0.2663 - accuracy: 0.8779 -
val_loss: 0.2408 - val_accuracy: 0.8792
Epoch 10/20
33/33 [==============================] - 14s 435ms/step - loss: 0.2478 - accuracy: 0.8874 -
val_loss: 0.2326 - val_accuracy: 0.8755
```

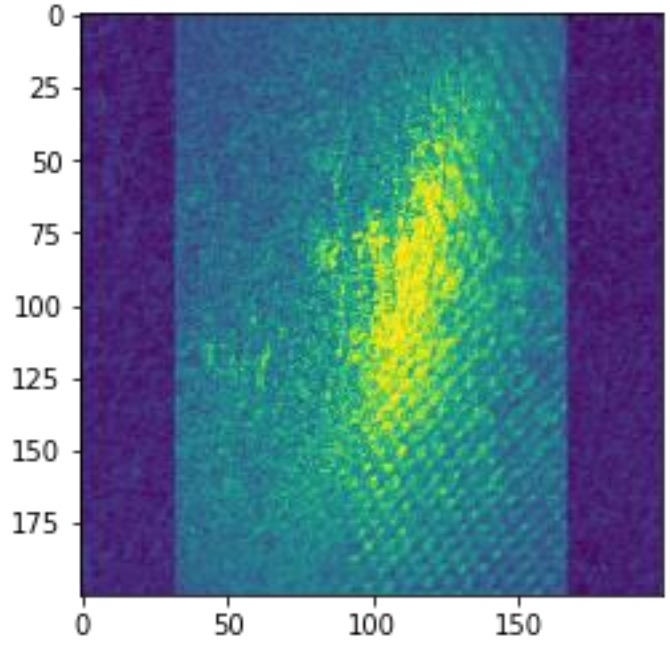
```
Epoch 11/20
33/33 [==============================] - 14s 435ms/step - loss: 0.2267 - accuracy: 0.8923 -
val_loss: 0.2150 - val_accuracy: 0.8830
Epoch 12/20
33/33 [==============================] - 14s 438ms/step - loss: 0.2133 - accuracy: 0.8977 -
val_loss: 0.2045 - val_accuracy: 0.9019
Epoch 13/20
33/33 [==============================] - 14s 438ms/step - loss: 0.2050 - accuracy: 0.9203 -
val_loss: 0.2040 - val_accuracy: 0.8868
Epoch 14/20
33/33 [==============================] - 14s 430ms/step - loss: 0.2036 - accuracy: 0.9106 -
val_loss: 0.1870 - val_accuracy: 0.8981
Epoch 15/20
33/33 [==============================] - 14s 434ms/step - loss: 0.1819 - accuracy: 0.9316 -
val_loss: 0.1695 - val_accuracy: 0.9132
Epoch 16/20
33/33 [==============================] - 14s 432ms/step - loss: 0.1522 - accuracy: 0.9426 -
val_loss: 0.1652 - val_accuracy: 0.9132
Epoch 17/20
33/33 [==============================] - 14s 438ms/step - loss: 0.1501 - accuracy: 0.9440 -
val_loss: 0.1660 - val_accuracy: 0.9094
Epoch 18/20
33/33 [==============================] - 14s 429ms/step - loss: 0.1503 - accuracy: 0.9425 -
val_loss: 0.1612 - val_accuracy: 0.9170
Epoch 19/20
33/33 [==============================] - 15s 442ms/step - loss: 0.1376 - accuracy: 0.9507 -
val_loss: 0.1482 - val_accuracy: 0.9170
Epoch 20/20
33/33 [==============================] - 14s 432ms/step - loss: 0.1236 - accuracy: 0.9587 -
val_loss: 0.1437 - val_accuracy: 0.9170
```



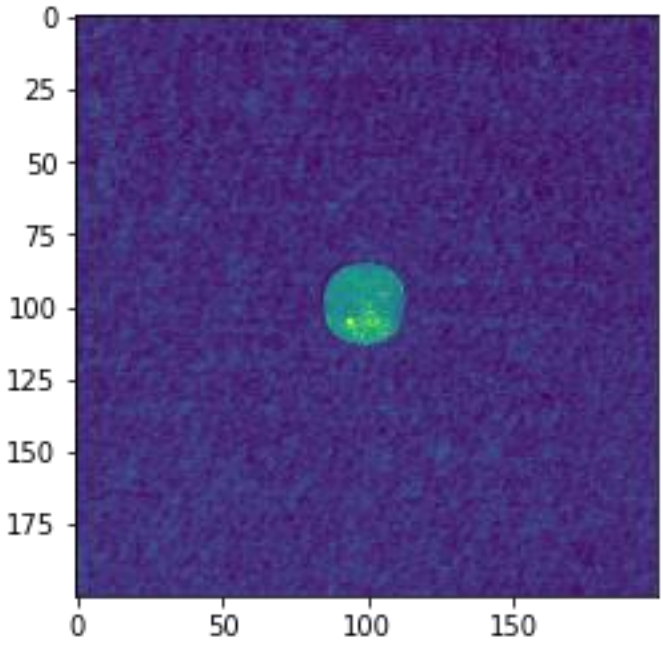


Testing Results:

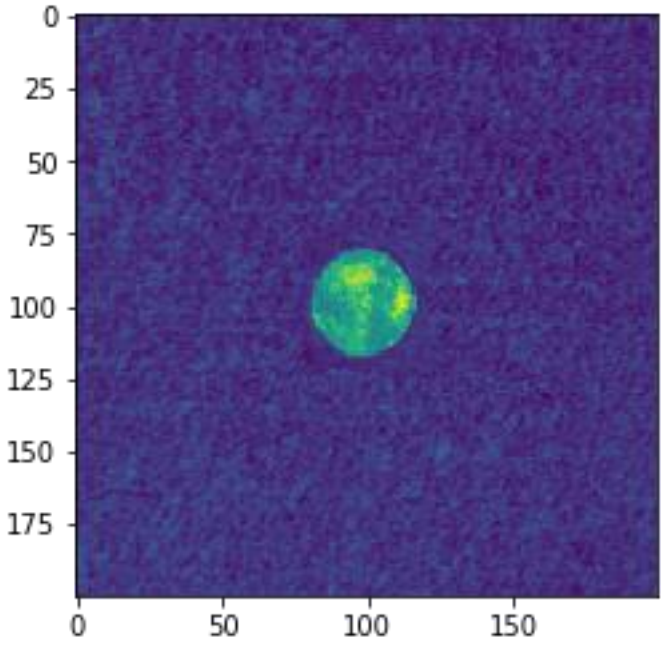
```
######################################################################
Test model with validation Data

9/9 [==============================] - 1s 82ms/step - loss: 0.1437 - accuracy: 0.9170
Validation accuracy - >  91.69811606407166

The predicted validation image is = SCRAP verify below
```



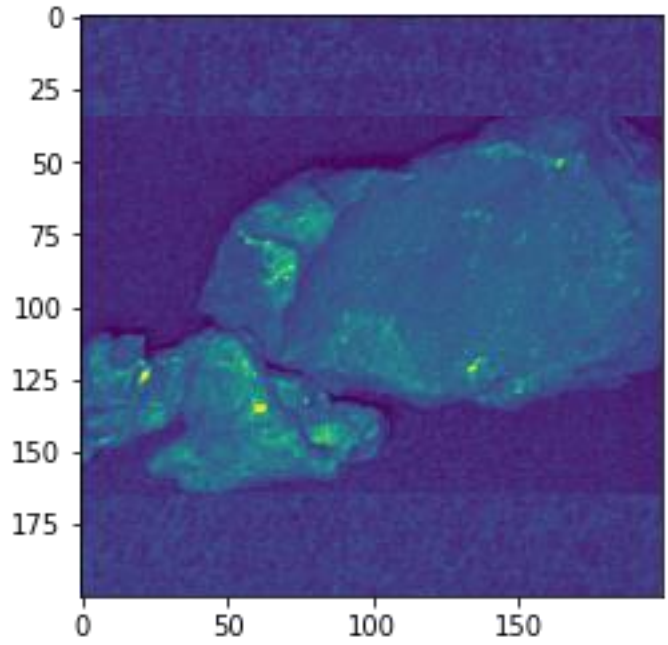

```
The predicted validation image is = COIN verify below
```



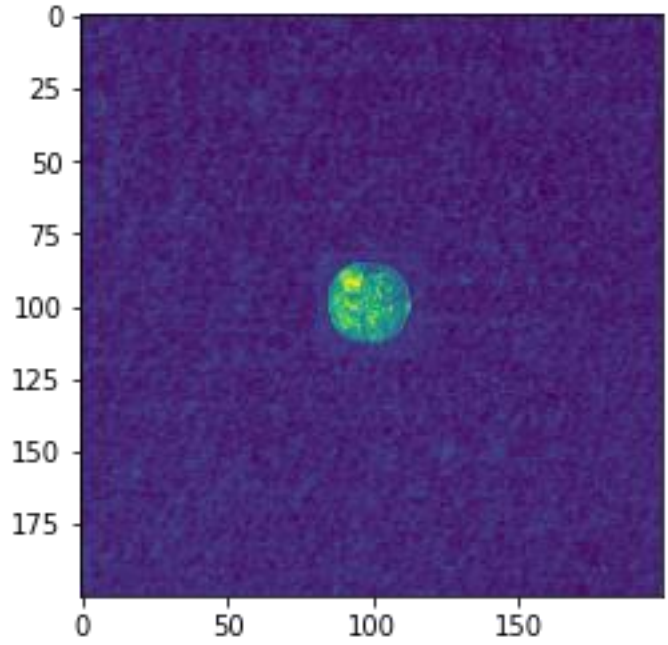
The predicted validation image is = COIN verify below



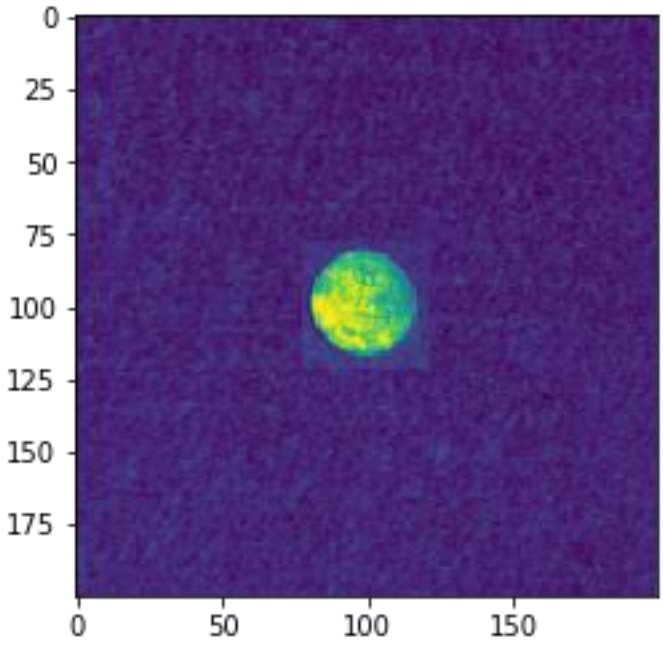
The predicted validation image is = SCRAP verify below



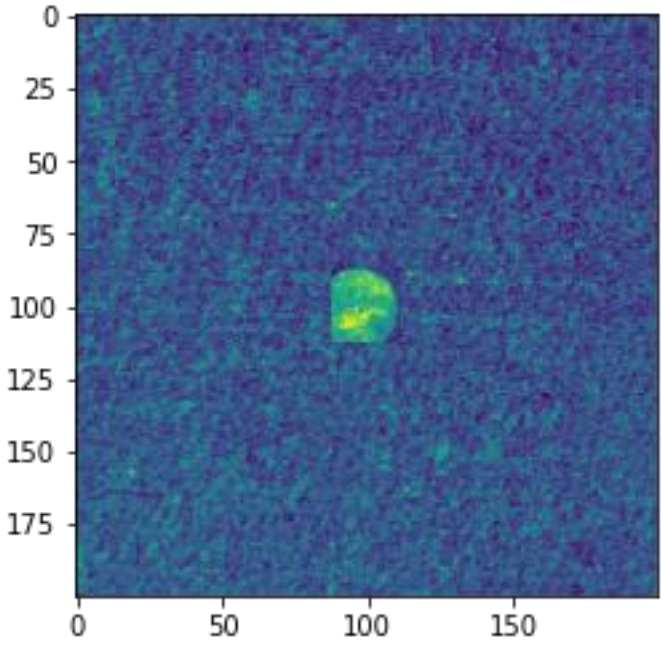

```
The predicted validation image is = COIN verify below
```



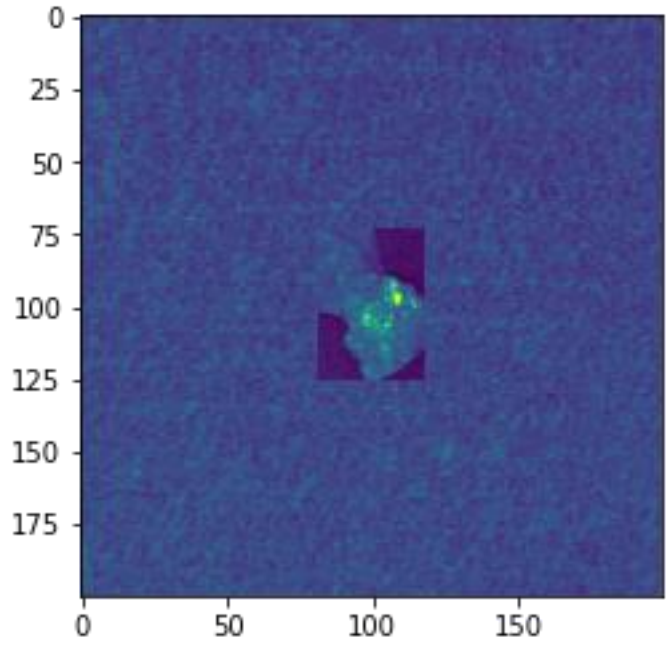

```
The predicted validation image is = COIN verify below
```



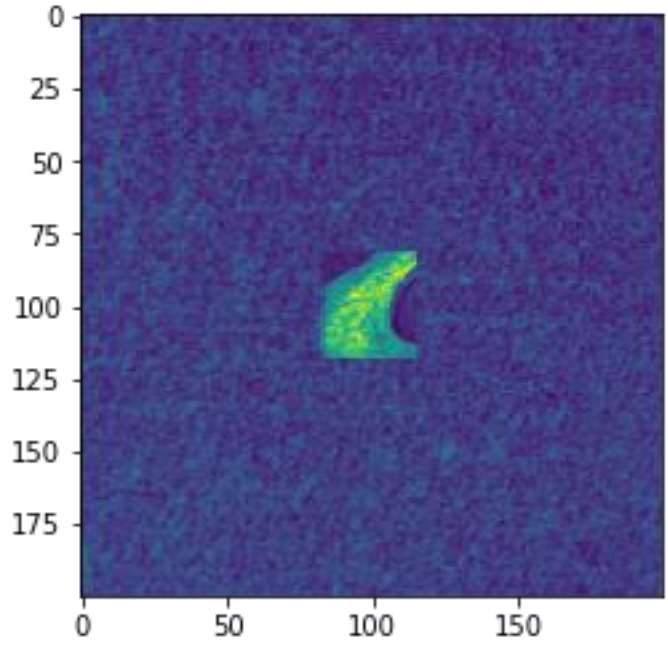

```
The predicted validation image is = COIN verify below
```



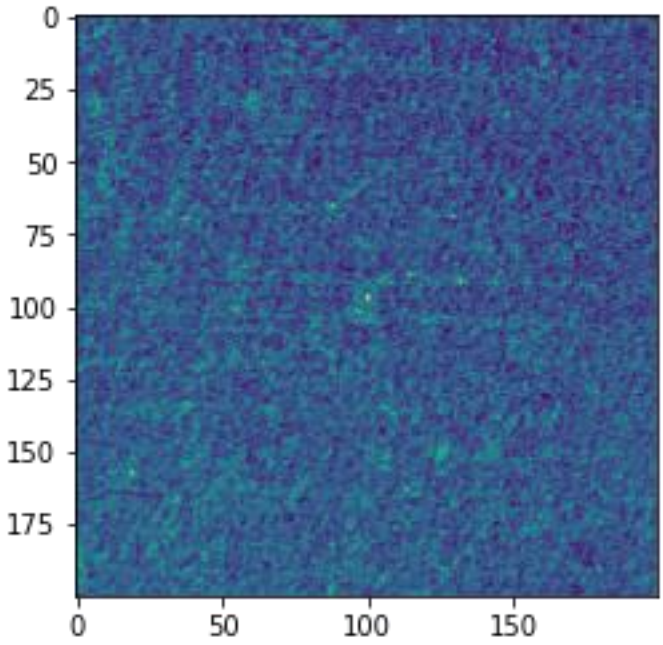

```
The predicted validation image is = COIN verify below
```



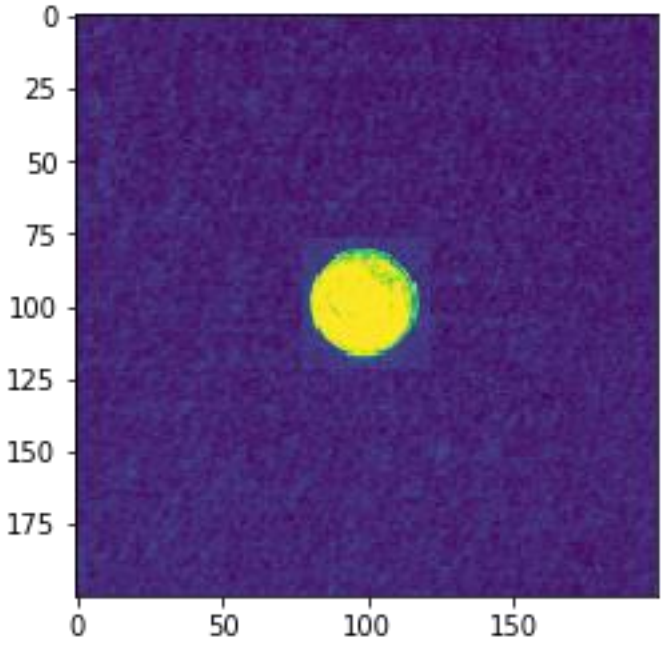

```
The predicted validation image is = COIN verify below
```



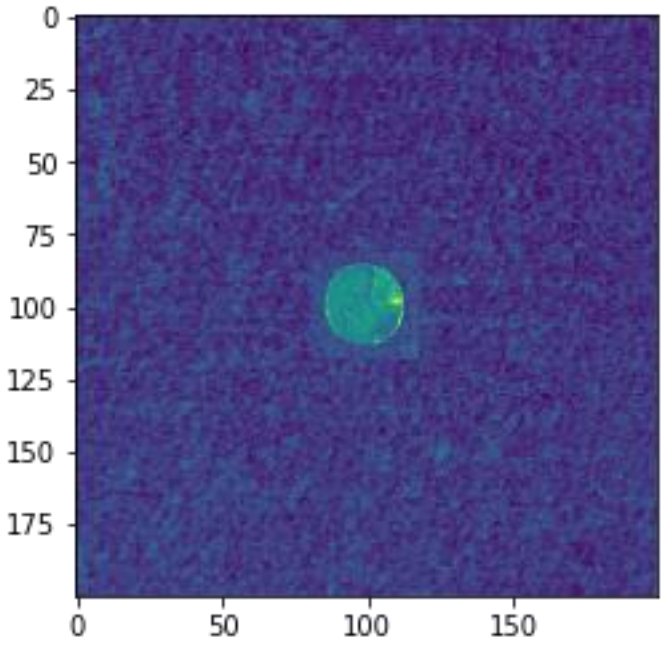

```
The predicted validation image is = COIN verify below
```



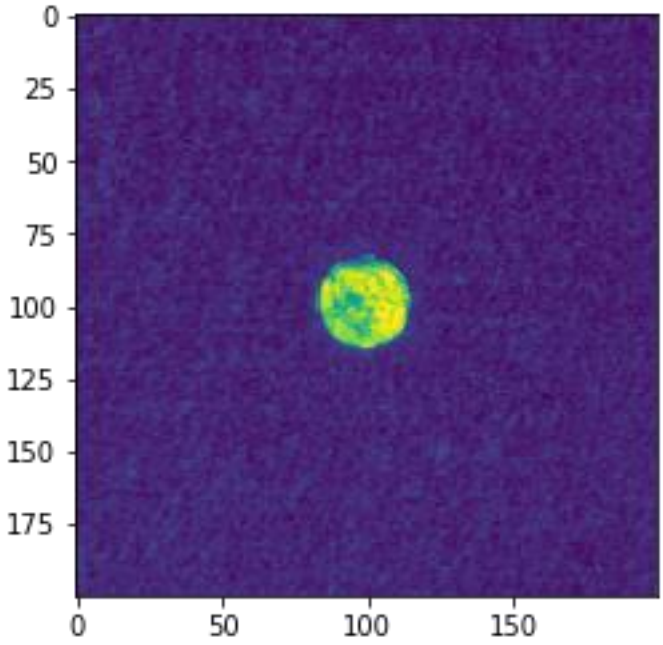

```
######################################################################
Test model with test Data

13/13 [==============================] - 1s 93ms/step - loss: 0.1053 - accuracy: 0.9470
Testing accuracy - >  94.69696879386902

The predicted testing image is = COIN verify below
```



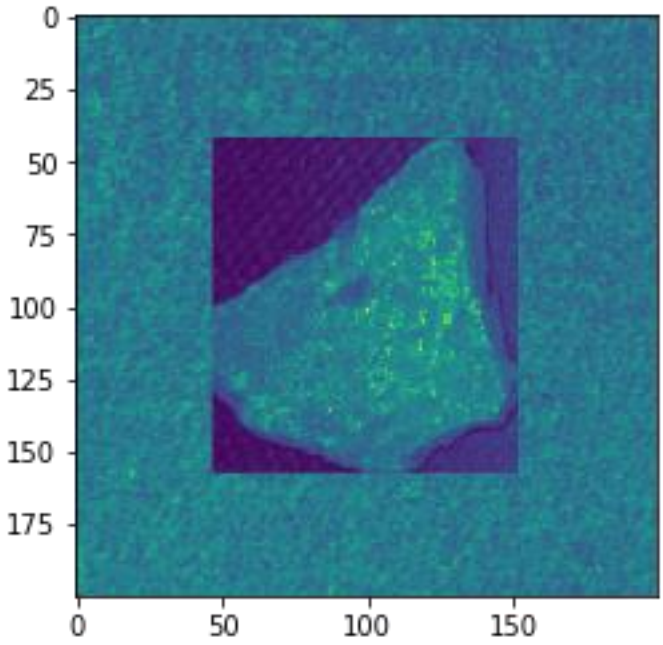

```
The predicted testing image is = COIN verify below
```



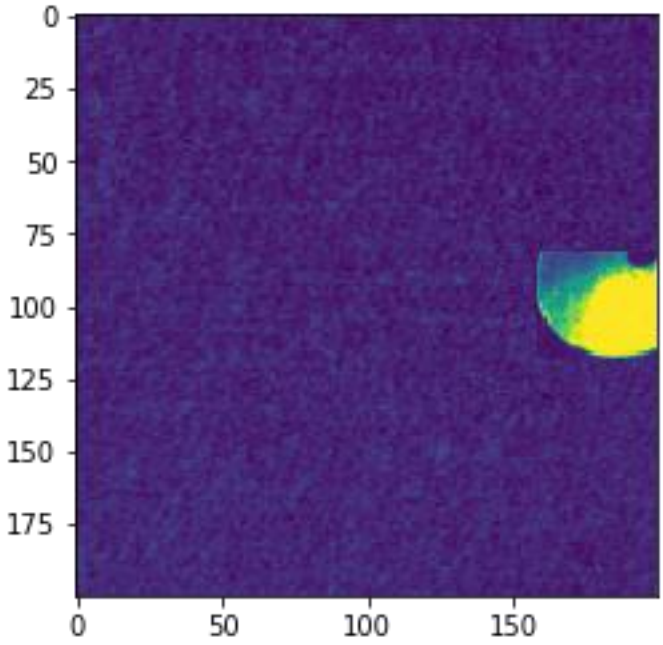
The predicted testing image is = COIN verify below



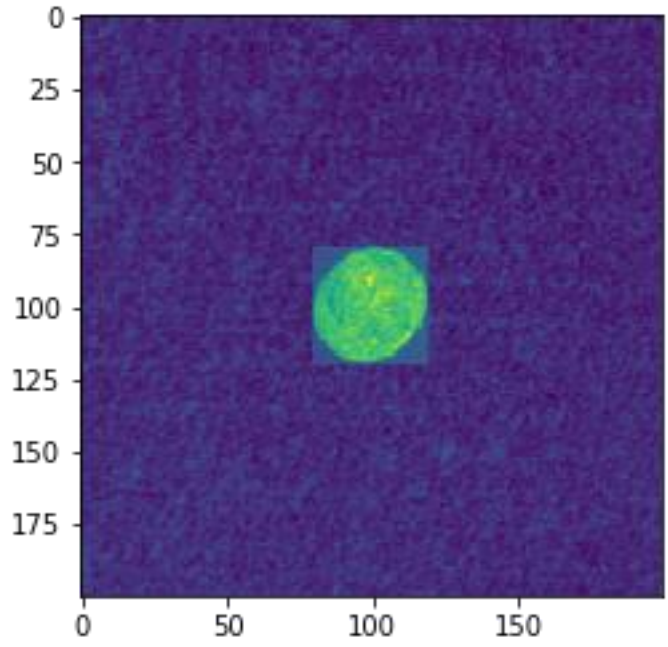
The predicted testing image is = SCRAP verify below



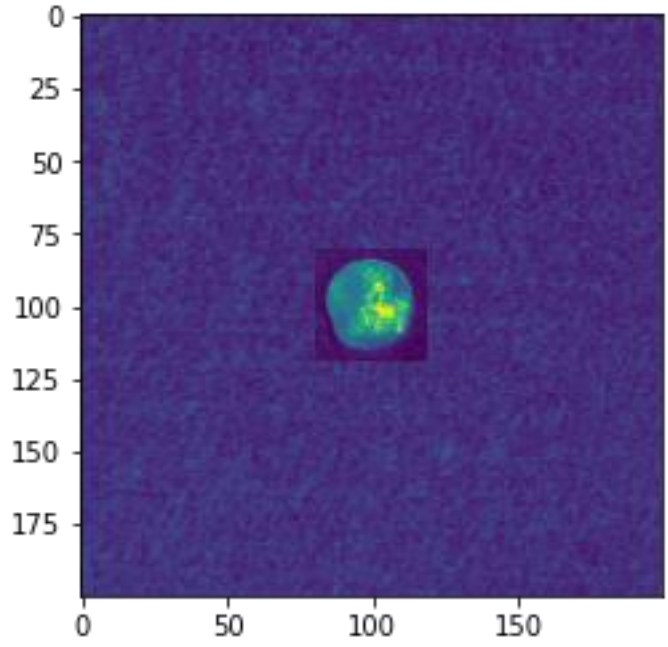

```
The predicted testing image is = SCRAP verify below
```



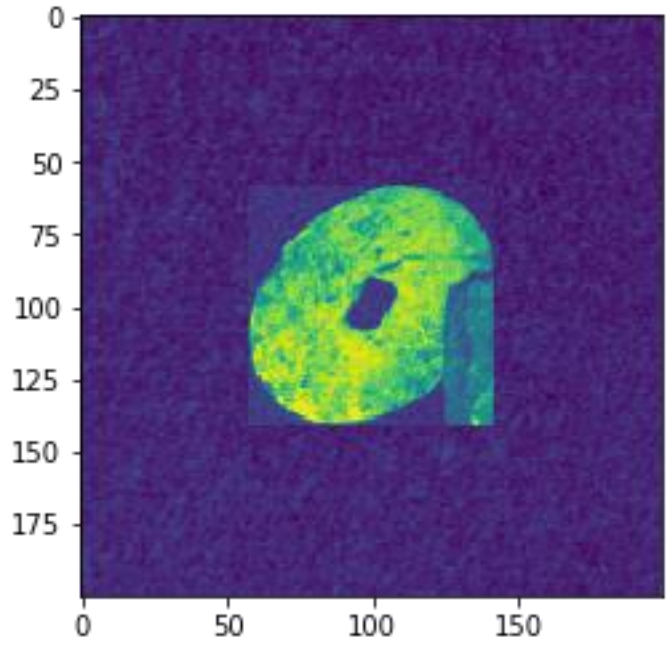

```
The predicted testing image is = COIN verify below
```



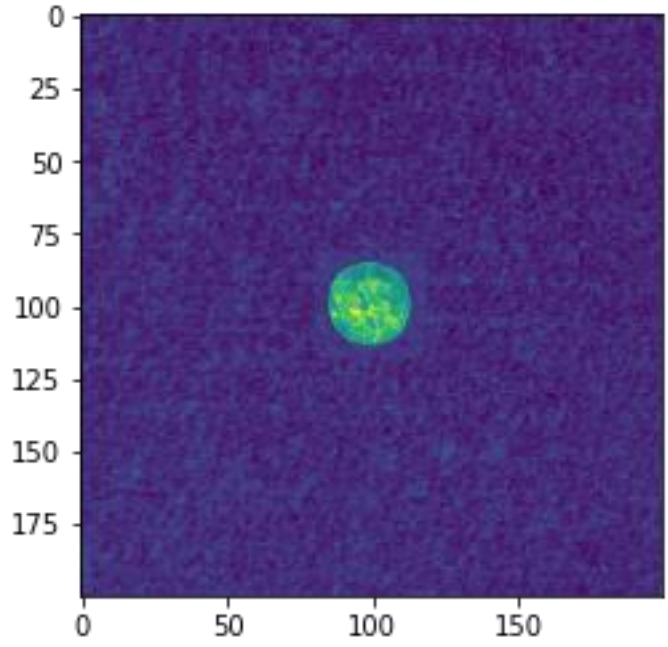
The predicted testing image is = COIN verify below



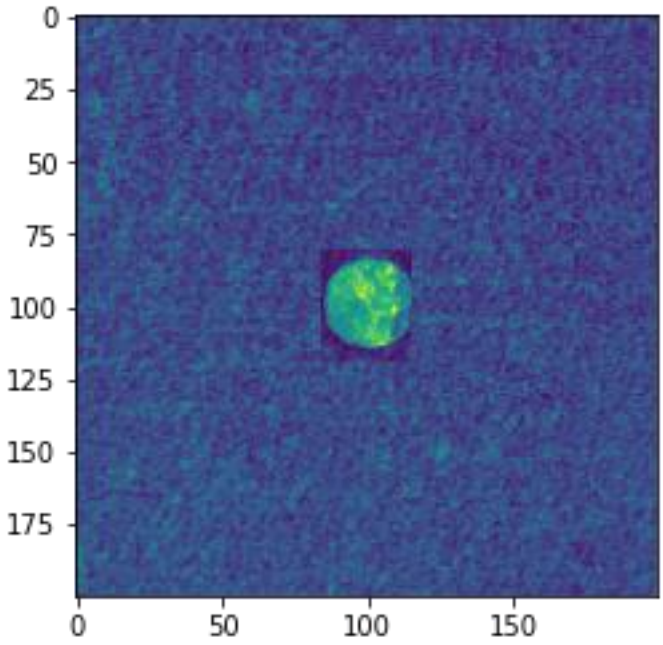
The predicted testing image is = SCRAP verify below

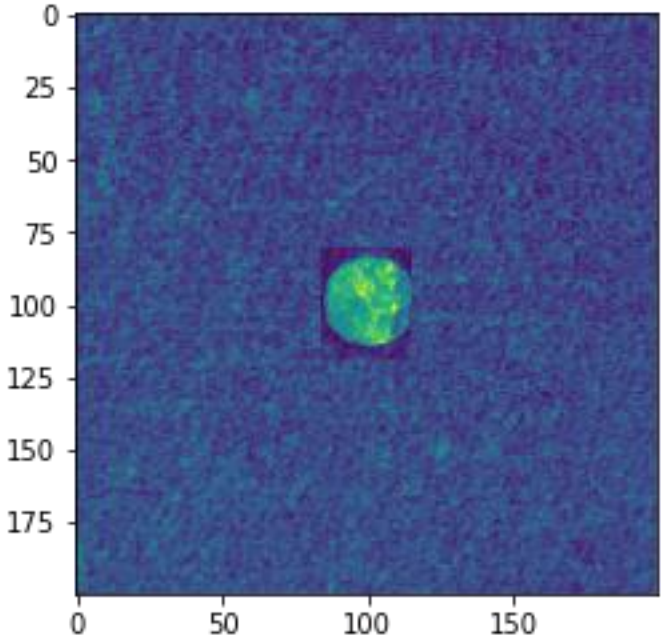



```
The predicted testing image is = COIN verify below
```




```
The predicted testing image is = COIN verify below
```





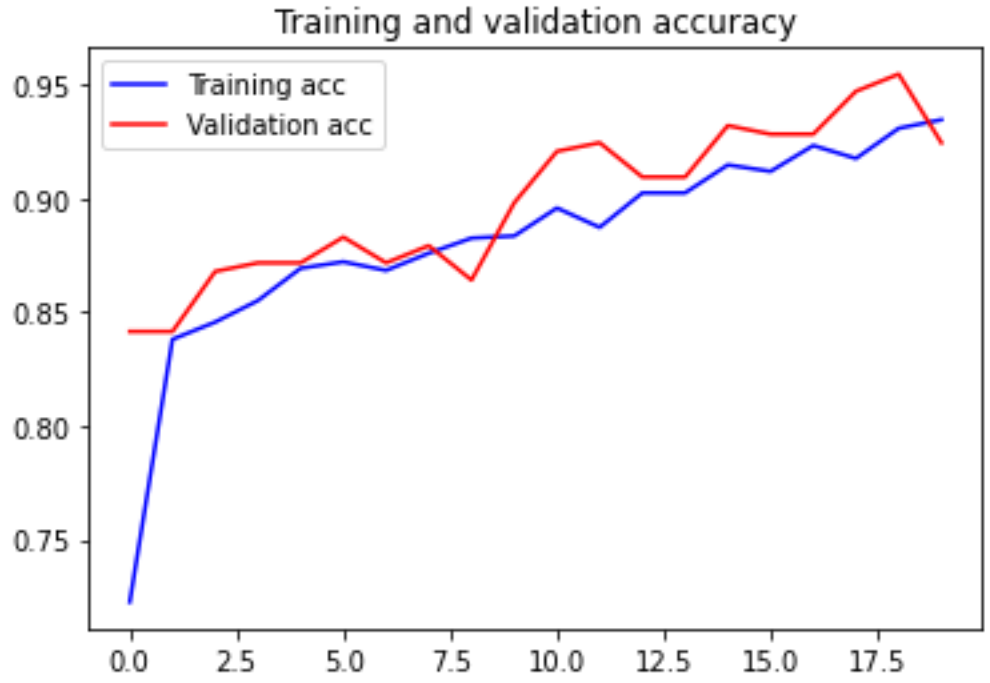
Part II CNN with Augmentation

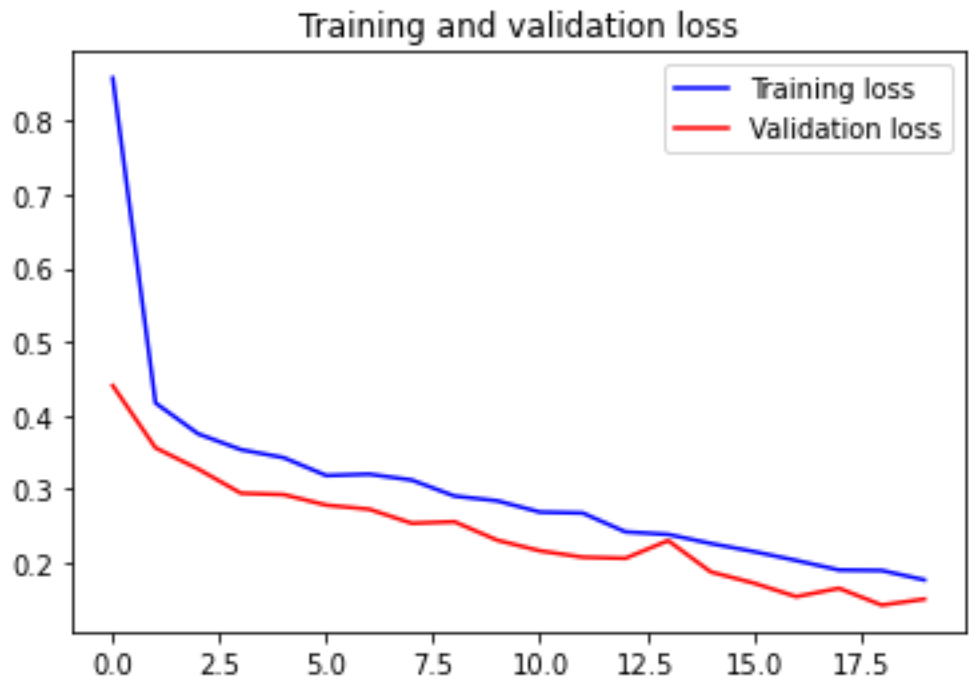
Training Results:

```
#################################################################
Load Datasets

Data Directory List 1- >  ['COIN', 'SCRAP']
Data Directory List 2- >  ['COIN', 'SCRAP']
```

```
#################################################################
Create Model

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 99, 99, 64)        640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 49, 49, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 49, 49, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 153664)            0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               19669120
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 19,670,018
Trainable params: 19,670,018
Non-trainable params: 0
_________________________________________________________________

#################################################################
Train Model

Epoch 1/20
33/33 [==============================] - 20s 574ms/step - loss: 1.1797 - accuracy: 0.6460 -
val_loss: 0.4404 - val_accuracy: 0.8415
Epoch 2/20
33/33 [==============================] - 17s 518ms/step - loss: 0.4447 - accuracy: 0.8299 -
val_loss: 0.3561 - val_accuracy: 0.8415
Epoch 3/20
33/33 [==============================] - 15s 462ms/step - loss: 0.3565 - accuracy: 0.8516 -
val_loss: 0.3273 - val_accuracy: 0.8679
Epoch 4/20
33/33 [==============================] - 15s 461ms/step - loss: 0.3574 - accuracy: 0.8505 -
val_loss: 0.2944 - val_accuracy: 0.8717
Epoch 5/20
33/33 [==============================] - 15s 458ms/step - loss: 0.3371 - accuracy: 0.8717 -
val_loss: 0.2926 - val_accuracy: 0.8717
Epoch 6/20
33/33 [==============================] - 15s 453ms/step - loss: 0.3063 - accuracy: 0.8817 -
val_loss: 0.2782 - val_accuracy: 0.8830
Epoch 7/20
33/33 [==============================] - 15s 462ms/step - loss: 0.3447 - accuracy: 0.8573 -
val_loss: 0.2727 - val_accuracy: 0.8717
Epoch 8/20
33/33 [==============================] - 15s 459ms/step - loss: 0.3386 - accuracy: 0.8678 -
val_loss: 0.2537 - val_accuracy: 0.8792
Epoch 9/20
33/33 [==============================] - 15s 460ms/step - loss: 0.2848 - accuracy: 0.8903 -
val_loss: 0.2557 - val_accuracy: 0.8642
Epoch 10/20
33/33 [==============================] - 15s 458ms/step - loss: 0.3095 - accuracy: 0.8812 -
val_loss: 0.2307 - val_accuracy: 0.8981
Epoch 11/20
33/33 [==============================] - 15s 457ms/step - loss: 0.2868 - accuracy: 0.8815 -
val_loss: 0.2163 - val_accuracy: 0.9208
Epoch 12/20
33/33 [==============================] - 15s 453ms/step - loss: 0.2601 - accuracy: 0.8948 -
val_loss: 0.2076 - val_accuracy: 0.9245
Epoch 13/20
33/33 [==============================] - 15s 453ms/step - loss: 0.2512 - accuracy: 0.8946 -
val_loss: 0.2061 - val_accuracy: 0.9094
Epoch 14/20
33/33 [==============================] - 15s 455ms/step - loss: 0.2129 - accuracy: 0.9063 -
val_loss: 0.2302 - val_accuracy: 0.9094
```

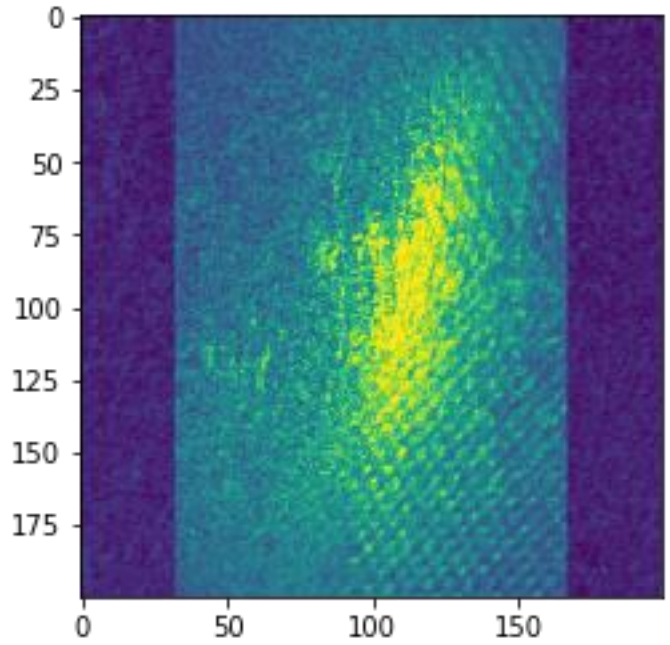
```
Epoch 15/20
33/33 [==============================] - 15s 452ms/step - loss: 0.2096 - accuracy: 0.9154 -
val_loss: 0.1876 - val_accuracy: 0.9321
Epoch 16/20
33/33 [==============================] - 15s 454ms/step - loss: 0.2228 - accuracy: 0.9232 -
val_loss: 0.1726 - val_accuracy: 0.9283
Epoch 17/20
33/33 [==============================] - 15s 453ms/step - loss: 0.2324 - accuracy: 0.9171 -
val_loss: 0.1541 - val_accuracy: 0.9283
Epoch 18/20
33/33 [==============================] - 15s 455ms/step - loss: 0.1892 - accuracy: 0.9167 -
val_loss: 0.1654 - val_accuracy: 0.9472
Epoch 19/20
33/33 [==============================] - 15s 454ms/step - loss: 0.2087 - accuracy: 0.9187 -
val_loss: 0.1426 - val_accuracy: 0.9547
Epoch 20/20
33/33 [==============================] - 15s 450ms/step - loss: 0.1881 - accuracy: 0.9327 -
val_loss: 0.1507 - val_accuracy: 0.9245
```



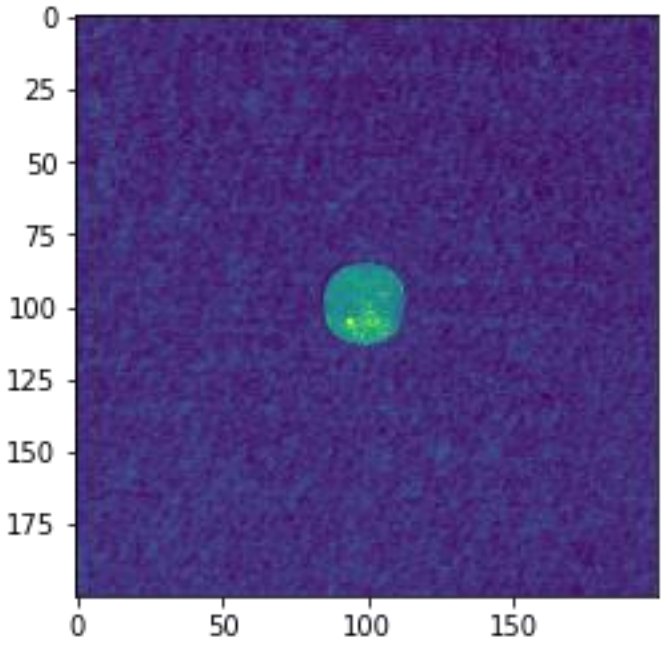


Testing Results:

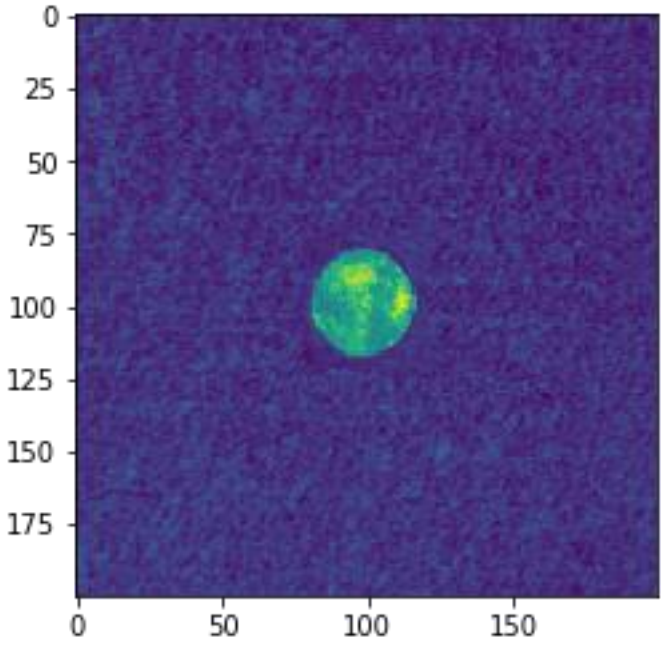
```
######################################################################
Test model with validation Data

9/9 [==============================] - 1s 88ms/step - loss: 0.1507 - accuracy: 0.9245
Validation accuracy - >  92.45283007621765

The predicted validation image is = SCRAP verify below
```



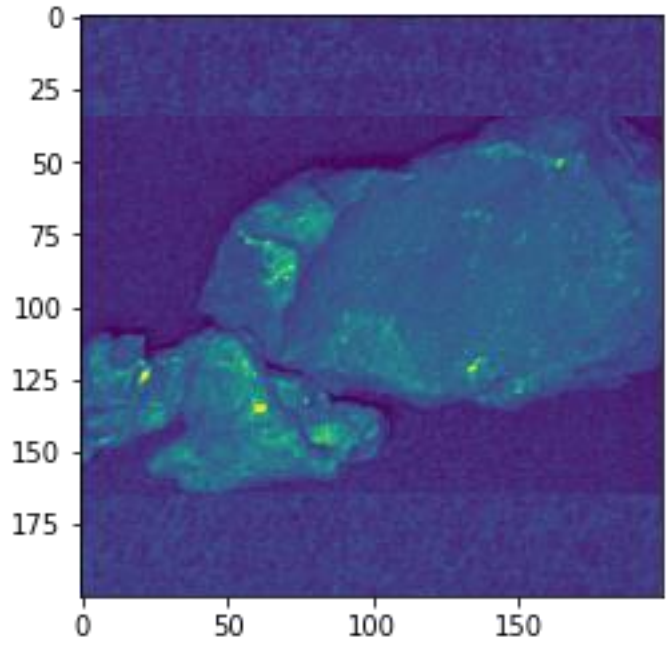

```
The predicted validation image is = COIN verify below
```



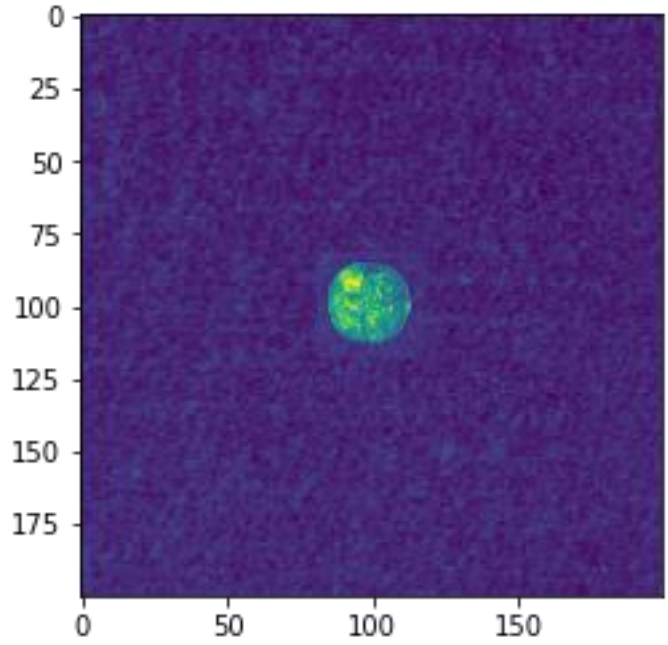
The predicted validation image is = COIN verify below



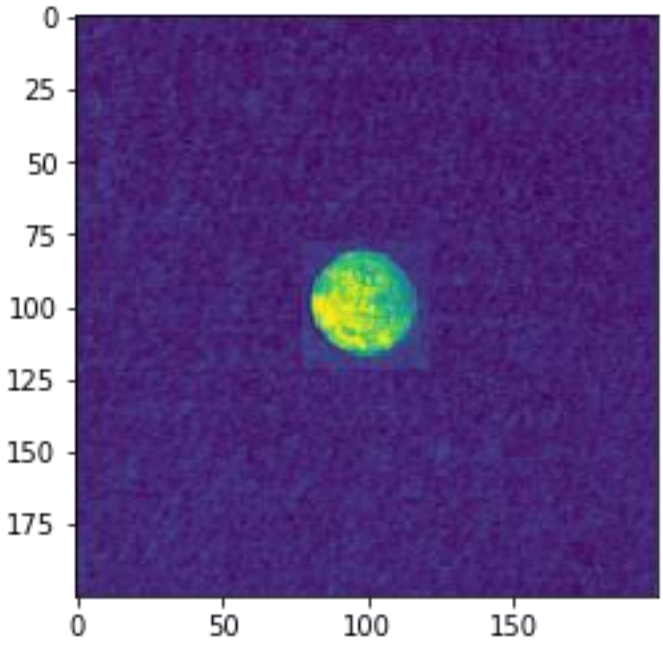
The predicted validation image is = SCRAP verify below



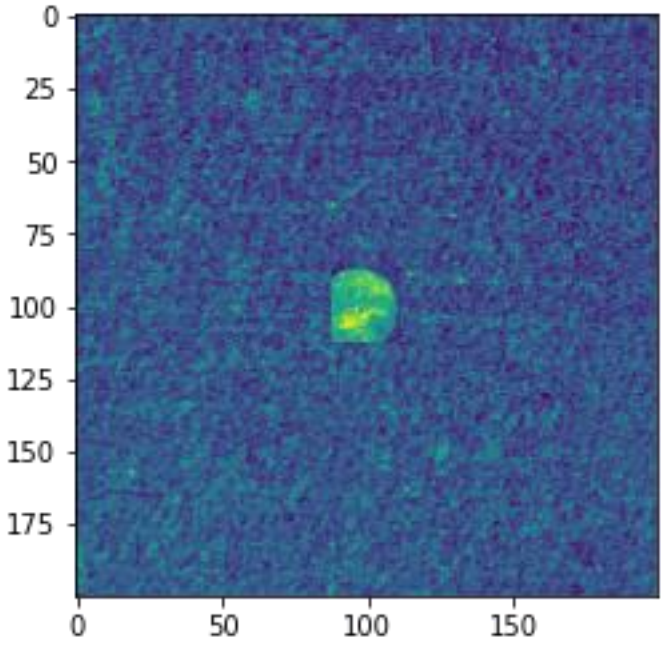

```
The predicted validation image is = COIN verify below
```



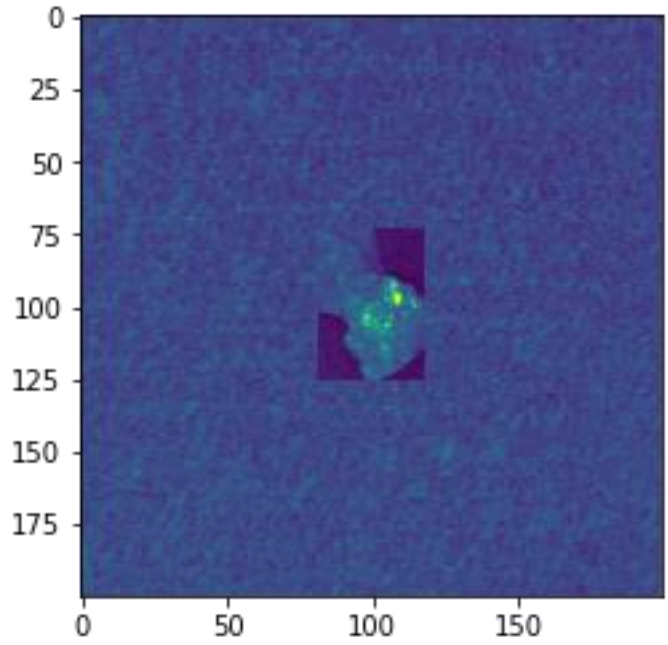

```
The predicted validation image is = COIN verify below
```



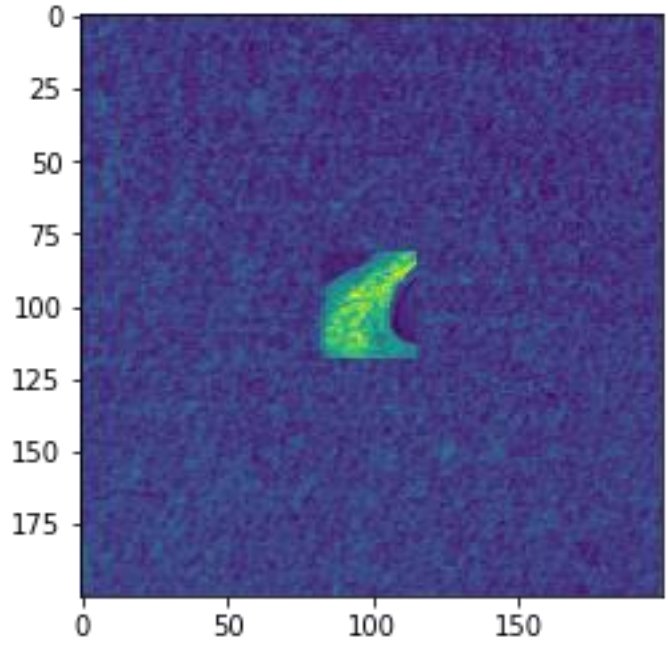
The predicted validation image is = COIN verify below



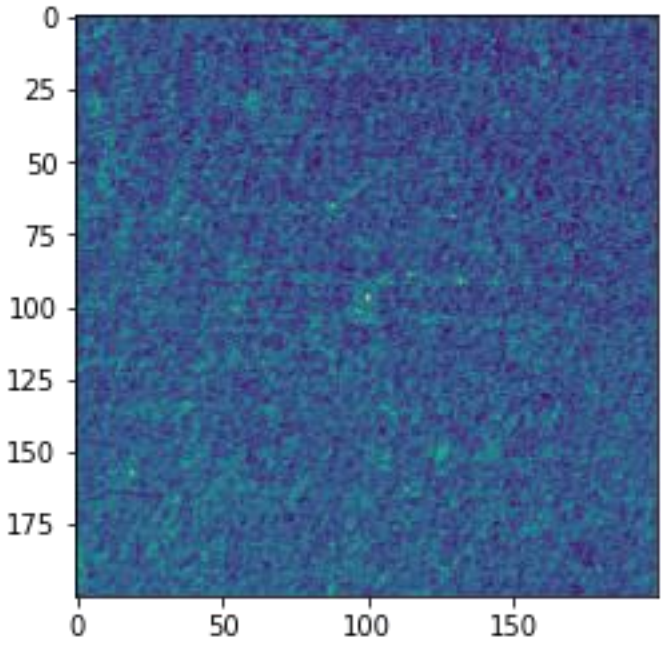
The predicted validation image is = SCRAP verify below



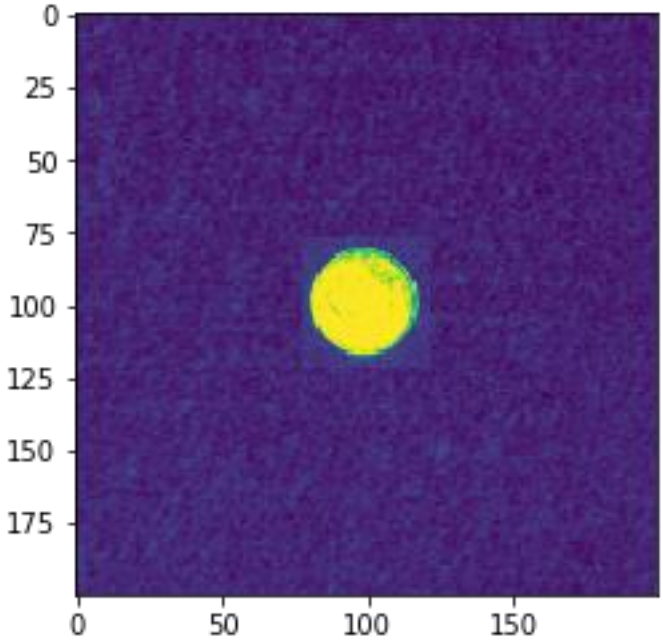

```
The predicted validation image is = COIN verify below
```



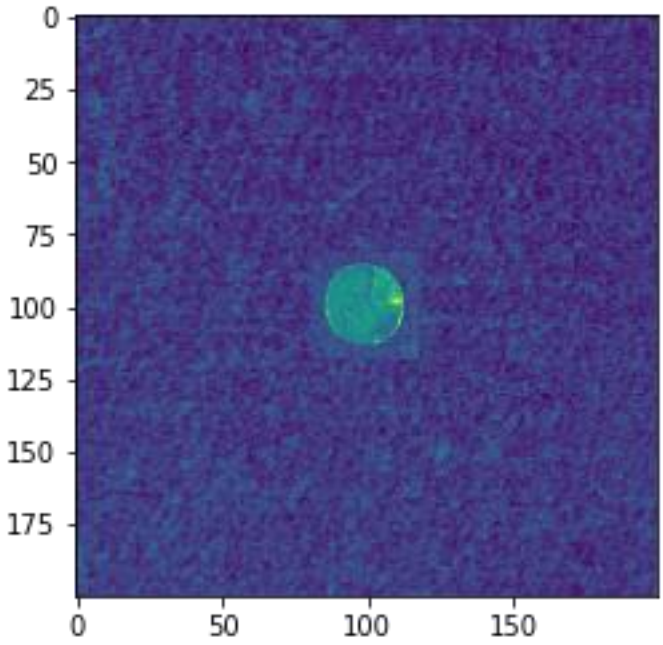

```
The predicted validation image is = COIN verify below
```



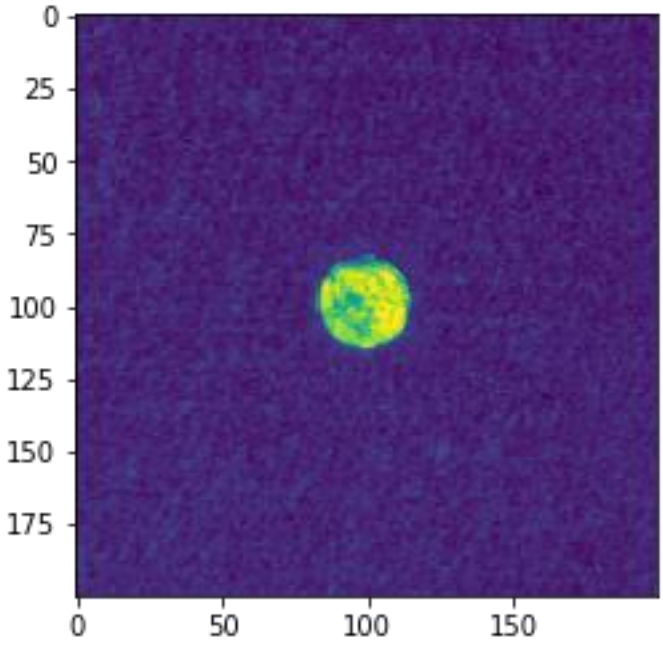

```
######################################################################
Test model with test Data

13/13 [==============================] - 1s 98ms/step - loss: 0.1612 - accuracy: 0.9369
Testing accuracy - >  93.6868667602539

The predicted testing image is = COIN verify below
```



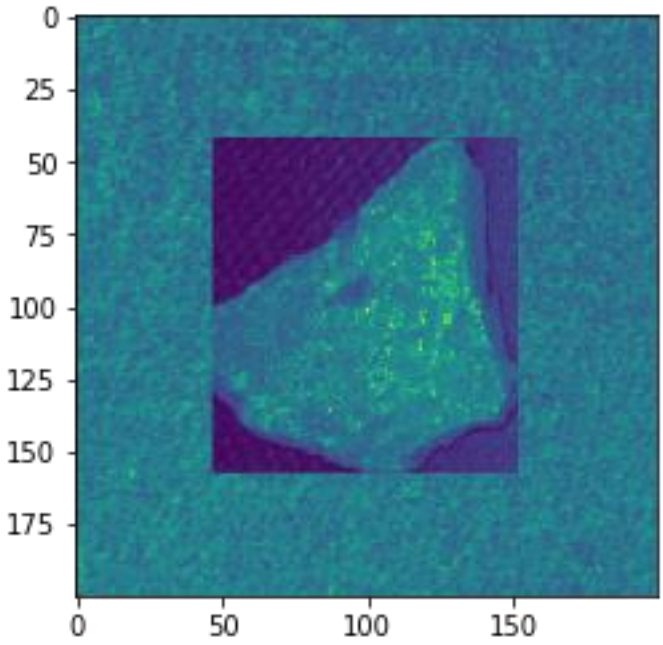

```
The predicted testing image is = COIN verify below
```



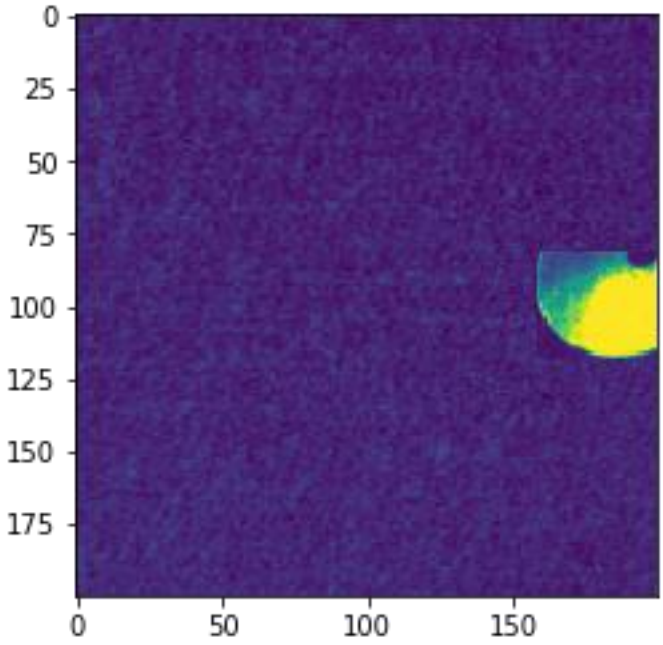
The predicted testing image is = COIN verify below



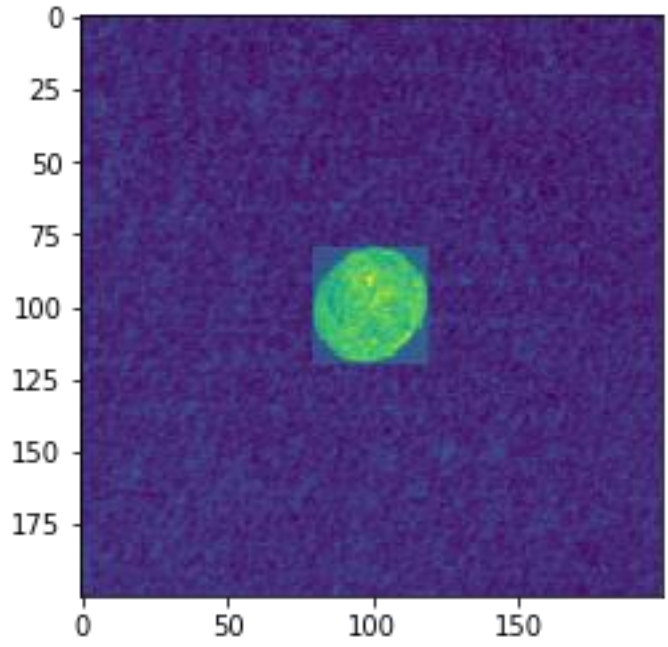
The predicted testing image is = SCRAP verify below



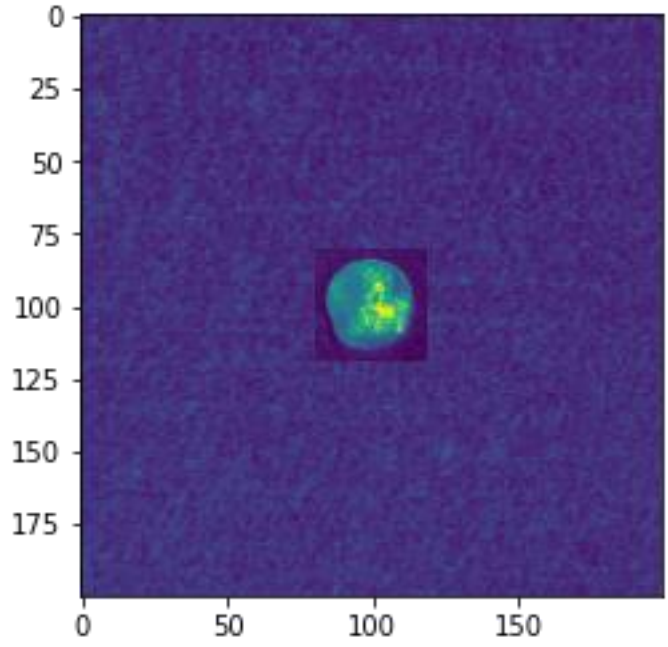

```
The predicted testing image is = COIN verify below
```



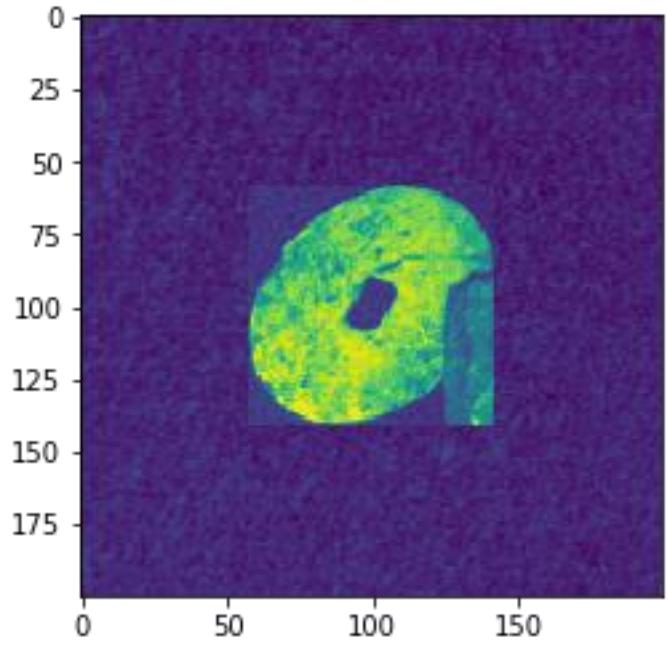

```
The predicted testing image is = COIN verify below
```



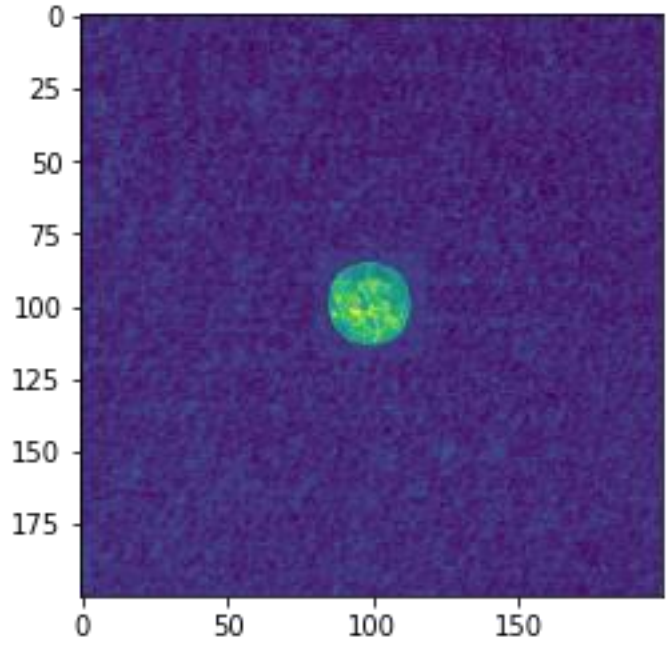
The predicted testing image is = COIN verify below



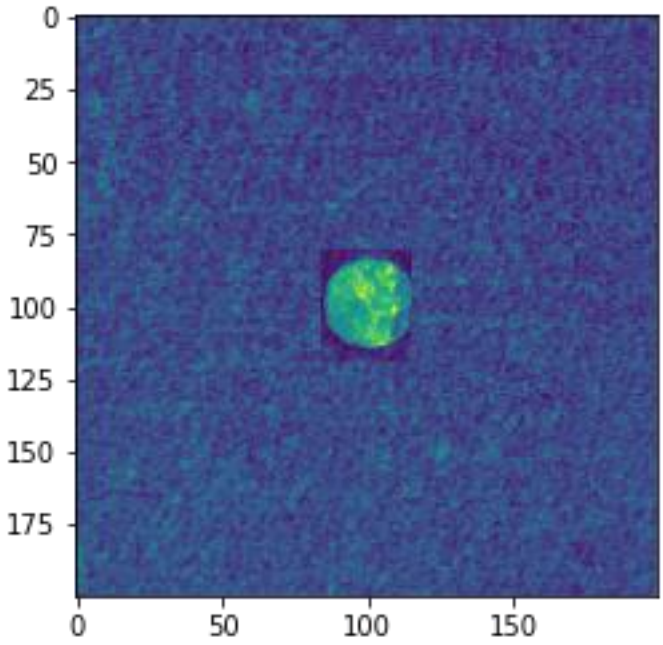
The predicted testing image is = SCRAP verify below




```
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:

https://github.com/jnn-dev

Conclusion:

This programing exercise utilized convolutional neural nets with augmentation to capture spatial and temporal dependencies of images through the application of filters. The augmentation included a horizontal and vertical flip to the training dataset. This augmentation helped improved the validation accuracy, but slightly lowered the training accuracy.