Import Statements

In [1]:

```
import os
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNorma
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('dark_background')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

In [25]:

```
!pip install keras
```

```
Requirement already satisfied: keras in c:\users\arivu\appdata\local\programs\pyth
on\python310\lib\site-packages (2.10.0)
```

```
[notice] A new release of pip available: 22.3 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

One Hot Encoding the Target Classes

In [2]:

```
encoder = OneHotEncoder()
encoder.fit([[0], [1]])


# 0 - hand loom
# 1 - power loom
```

Out[2]:

```
OneHotEncoder()
```

Creating 3 Important Lists --

1. data list for storing image data in numpy array form
2. paths list for storing paths of all images
3. result list for storing one hot encoded form of target class whether handloom or powerloom

In [3]:

```
# handloom

data = []
paths = []
result = []

for r, d, f in os.walk(r"C:\Users\arivu\OneDrive\Dataset_1\Handloom"):
    for file in f:
        if '.jpg' in file:
            paths.append(os.path.join(r, file))

for path in paths:
    img = Image.open(path)
    img = img.resize((128,128))
    img = np.array(img)
    if(img.shape == (128,128,3)):
        data.append(np.array(img))
        result.append(encoder.transform([[0]]).toarray())
```

In [4]:

```
# Powerloom

paths = []
for r, d, f in os.walk(r"C:\Users\arivu\OneDrive\Dataset_1\powerloom"):
    for file in f:
        if '.jpg' in file:
            paths.append(os.path.join(r, file))

for path in paths:
    img = Image.open(path)
    img = img.resize((128,128))
    img = np.array(img)
    if(img.shape == (128,128,3)):
        data.append(np.array(img))
        result.append(encoder.transform([[1]]).toarray())
```

In [5]:

```
data = np.array(data)
data.shape
```

Out[5]:

```
(378, 128, 128, 3)
```

In [6]:

```
result = np.array(result)
result = result.reshape(378,2)
```

Feature Extraction from images

In [17]:

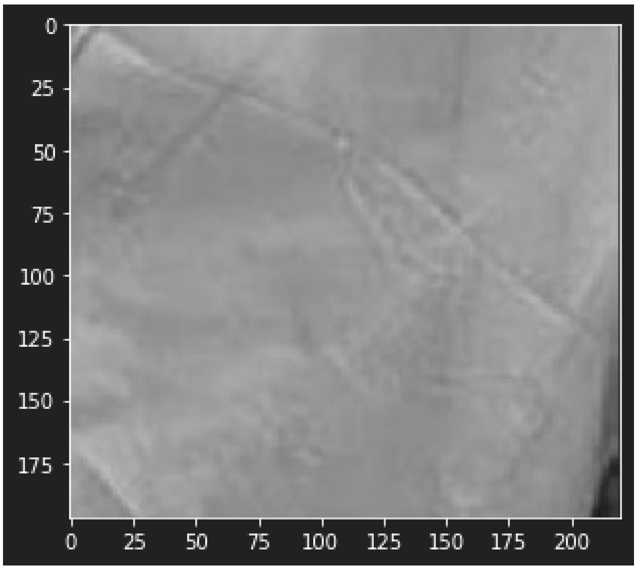
```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

In [24]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from skimage.io import imread, imshow
image = imread('D:\Arivu\Documents\project\Dataset\WhatsApp Image 2022-11-01 at 5.
imshow(image)
```

Out[24]:

```
<matplotlib.image.AxesImage at 0x2366a4fd120>
```

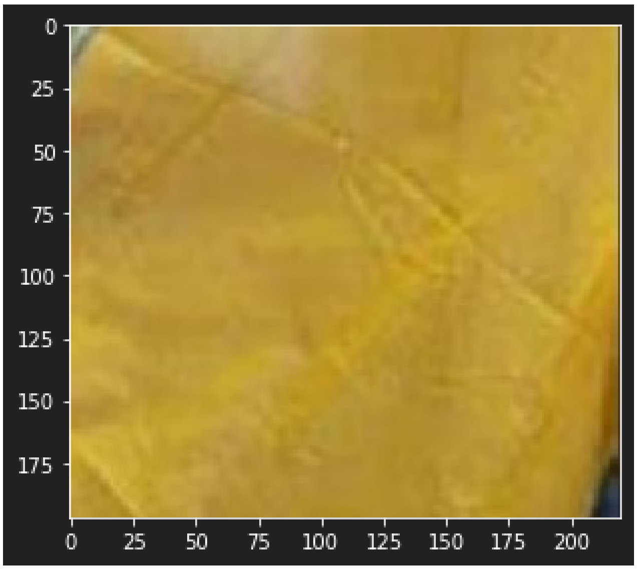


In [25]:
```
#check the image shape
print(image.shape)

print(image)
```

```
(197, 220)
[[0.57000588 0.53499451 0.53275451 ... 0.63594902 0.63911098 0.31981922]
 [0.51538667 0.51538667 0.54059765 ... 0.64771373 0.64919412 0.32374078]
 [0.48569569 0.5209898  0.57645725 ... 0.65947843 0.66095882 0.33158392]
 ...
 [0.58914235 0.59698549 0.60454588 ... 0.19534941 0.20738196 0.15219725]
 [0.59306392 0.59698549 0.60454588 ... 0.20262706 0.21102078 0.15611882]
 [0.59698549 0.59698549 0.60454588 ... 0.20654863 0.21494235 0.15555333]]
```

In [26]:
```
image = imread('D:\Arivu\Documents\project\Dataset\WhatsApp Image 2022-11-01 at 5.
imshow(image)
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x236773d6350>
```



In [27]:
```
print(image.shape)
```

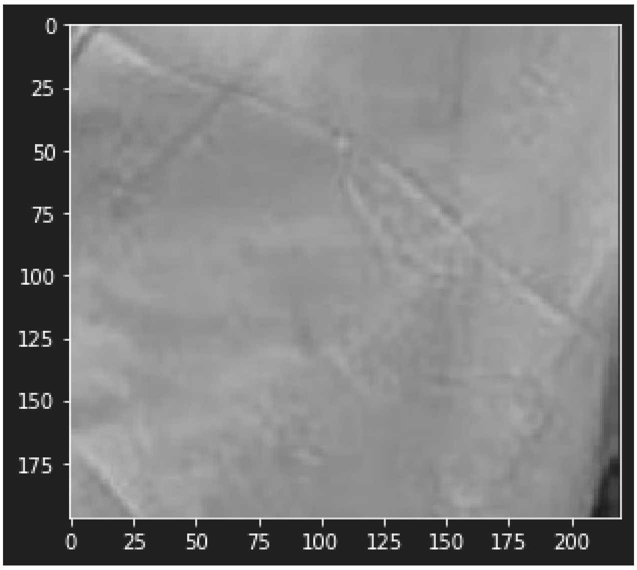
```
(197, 220, 3)
```

In [28]:
```
image
```

Out[28]:
```
array([[[140, 150, 115],
        [131, 141, 107],
        [131, 140, 109],
        ...,
        [191, 164,  59],
        [189, 163,  86],
        [106,  81,  15]],

       [[126, 136, 102],
        [126, 136, 102],
        [133, 142, 111],
        ...,
        [194, 167,  62],
        [191, 166,  86],
        [107,  82,  16]],

       [[119, 128,  97],
        [128, 137, 106],
        [142, 151, 122],
        ...,
        [197, 170,  65],
        [194, 169,  89],
        [109,  84,  18]],

       ...,

       [[184, 150,  53],
        [186, 152,  55],
        [188, 154,  56],
        ...,
        [ 42,  50,  71],
        [ 46,  53,  72],
        [ 32,  39,  57]],

       [[185, 151,  54],
        [186, 152,  55],
        [188, 154,  56],
        ...,
        [ 44,  52,  71],
        [ 47,  54,  72],
        [ 33,  40,  58]],

       [[186, 152,  55],
        [186, 152,  55],
        [188, 154,  56],
        ...,
        [ 45,  53,  72],
        [ 48,  55,  73],
        [ 33,  40,  56]]], dtype=uint8)
```

In [29]:
```
image = imread('D:\Arivu\Documents\project\Dataset\WhatsApp Image 2022-11-01 at 5.
image.shape, imshow(image)
```

Out[29]:
```
((197, 220), <matplotlib.image.AxesImage at 0x236774581f0>)
```



```
In [30]: print(image.shape)
```

```
(197, 220)
```

```
In [31]: image
```

```
Out[31]: array([[0.57000588, 0.53499451, 0.53275451, ..., 0.63594902, 0.63911098,
                 0.31981922],
                [0.51538667, 0.51538667, 0.54059765, ..., 0.64771373, 0.64919412,
                 0.32374078],
                [0.48569569, 0.5209898 , 0.57645725, ..., 0.65947843, 0.66095882,
                 0.33158392],
                ...,
                [0.58914235, 0.59698549, 0.60454588, ..., 0.19534941, 0.20738196,
                 0.15219725],
                [0.59306392, 0.59698549, 0.60454588, ..., 0.20262706, 0.21102078,
                 0.15611882],
                [0.59698549, 0.59698549, 0.60454588, ..., 0.20654863, 0.21494235,
                 0.15555333]])
```

```
In [32]: #Find the pixel features
         feature = np.reshape(image, (197*220))
         feature.shape
```

```
Out[32]: (43340,)
```

```
In [34]: feature
```

```
Out[34]: array([0.57000588, 0.53499451, 0.53275451, ..., 0.20654863, 0.21494235,
                0.15555333])
```

```
In [35]: image = imread('D:\Arivu\Documents\project\Dataset\WhatsApp Image 2022-11-01 at 5.
         feature_matrix_image = np.zeros((197,220))
         feature_matrix_image
```

```
Out[35]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])
```

```
In [36]: feature_matrix_image.shape
```

```
Out[36]: (197, 220)
```

In [37]:
```
for i in range(0,image.shape[0]):

    for j in range(0,image.shape[1]):

        feature_matrix_image[i][j] = ((int(image[i,j,0]) + int(image[i,j,1]) + int
```

In [38]:
```
feature_matrix_image
```

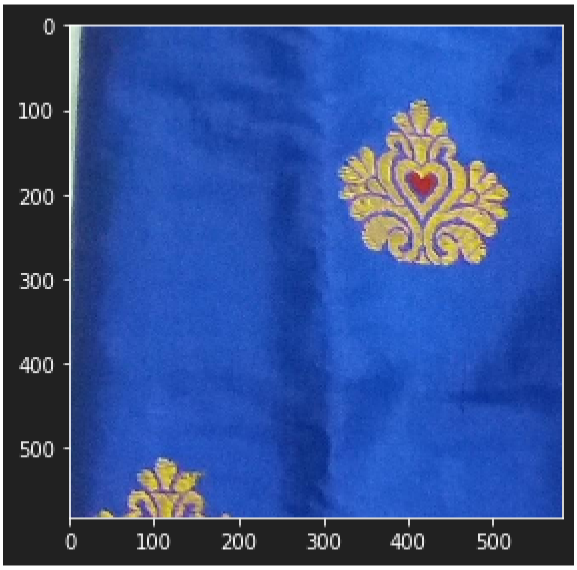
Out[38]:
```
array([[135.        , 126.33333333, 126.66666667, ..., 138.        ,
        146.        ,  67.33333333],
       [121.33333333, 121.33333333, 128.66666667, ..., 141.        ,
        147.66666667,  68.33333333],
       [114.66666667, 123.66666667, 138.33333333, ..., 144.        ,
        150.66666667,  70.33333333],
       ...,
       [129.        , 131.        , 132.66666667, ...,  54.33333333,
         57.        ,  42.66666667],
       [130.        , 131.        , 132.66666667, ...,  55.66666667,
         57.66666667,  43.66666667],
       [131.        , 131.        , 132.66666667, ...,  56.66666667,
         58.66666667,  43.        ]])
```

In [39]:
```
feature_sample = np.reshape(feature_matrix_image, (197*220))

feature_sample
```

Out[39]:
```
array([135.        , 126.33333333, 126.66666667, ...,  56.66666667,
        58.66666667,  43.        ])
```

In [43]:
```
from skimage.io import imread, imshow

image = imread(r"C:\Users\arivu\OneDrive\Dataset_1\Handloom\1.jpg")

imshow(image)
```

Out[43]:
```
<matplotlib.image.AxesImage at 0x23677af2260>
```



In [44]:
```
import cv2

import numpy as np

import cv2

import matplotlib.pyplot as plt
```

```
%matplotlib inline
img_load = cv2.imread(r"C:\Users\arivu\OneDrive\Dataset_1\Handloom\1.jpg")
img_load
```

Out[44]:

```
array([[[216, 232, 215],
        [214, 230, 213],
        [211, 227, 209],
        ...,
        [169,  65,  22],
        [169,  65,  22],
        [169,  65,  22]],

       [[215, 231, 214],
        [213, 229, 212],
        [212, 228, 210],
        ...,
        [169,  65,  22],
        [169,  65,  22],
        [169,  65,  22]],

       [[214, 230, 213],
        [213, 229, 212],
        [212, 228, 210],
        ...,
        [169,  65,  22],
        [169,  65,  22],
        [170,  66,  23]],

       ...,

       [[114,  58,  47],
        [111,  55,  44],
        [107,  51,  40],
        ...,
        [160,  63,  27],
        [159,  62,  26],
        [158,  61,  25]],

       [[113,  57,  46],
        [111,  55,  44],
        [108,  52,  41],
        ...,
        [158,  61,  25],
        [157,  60,  24],
        [157,  60,  24]],

       [[113,  57,  46],
        [111,  55,  44],
        [109,  53,  42],
        ...,
        [157,  60,  24],
        [156,  59,  23],
        [156,  59,  23]]], dtype=uint8)
```

In [49]:

```
 # Convert from cv's BRG default color order to RGB
img_load1 = cv2.cvtColor(img_load, cv2.COLOR_BGR2RGB)
img_load1
```

Out[49]:

```
array([[[215, 232, 216],
        [213, 230, 214],
        [209, 227, 211],
        ...,
        [ 22,  65, 169],
        [ 22,  65, 169],
        [ 22,  65, 169]],

       [[214, 231, 215],
        [212, 229, 213],
        [210, 228, 212],
        ...,
        [ 22,  65, 169],
        [ 22,  65, 169],
        [ 22,  65, 169]],

       [[213, 230, 214],
        [212, 229, 213],
        [210, 228, 212],
        ...,
        [ 22,  65, 169],
        [ 22,  65, 169],
        [ 23,  66, 170]],

       ...,

       [[ 47,  58, 114],
        [ 44,  55, 111],
        [ 40,  51, 107],
        ...,
        [ 27,  63, 160],
        [ 26,  62, 159],
        [ 25,  61, 158]],

       [[ 46,  57, 113],
        [ 44,  55, 111],
        [ 41,  52, 108],
        ...,
        [ 25,  61, 158],
        [ 24,  60, 157],
        [ 24,  60, 157]],

       [[ 46,  57, 113],
        [ 44,  55, 111],
        [ 42,  53, 109],
        ...,
        [ 24,  60, 157],
        [ 23,  59, 156],
        [ 23,  59, 156]]], dtype=uint8)
```

In [50]:

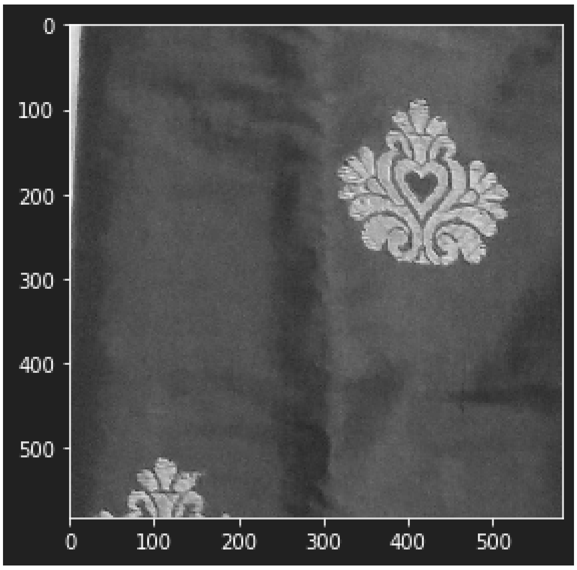
```
#converting image to Gray scale

gray_image = cv2.cvtColor(img_load,cv2.COLOR_BGR2GRAY)
```

In [52]:

```
imshow(gray_image)
```

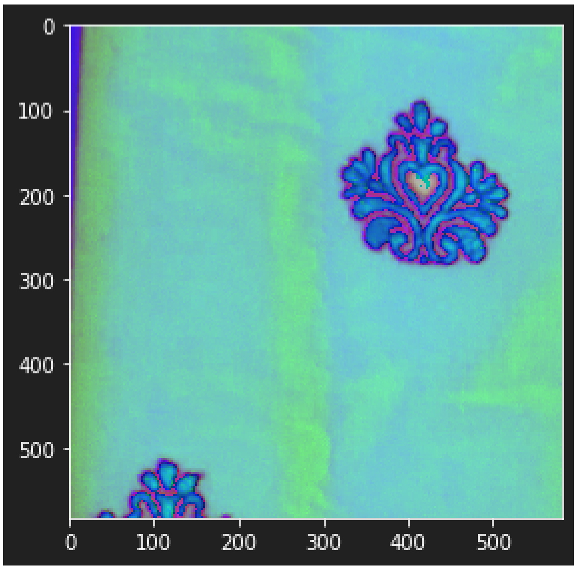
Out[52]:

```
<matplotlib.image.AxesImage at 0x23677df86a0>
```



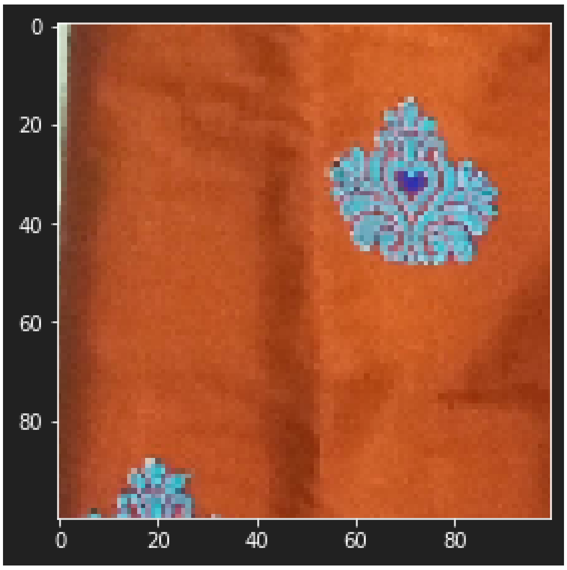
In [54]:
```
hsv_image_load = cv2.cvtColor(img_load,cv2.COLOR_BGR2HSV)
imshow(hsv_image_load)
```

Out[54]:
```
<matplotlib.image.AxesImage at 0x23677e19f90>
```



In [55]:
```
smaller_image_size = cv2.resize(img_load,(100,100))
imshow(smaller_image_size)
```

Out[55]:
```
<matplotlib.image.AxesImage at 0x23677e472b0>
```

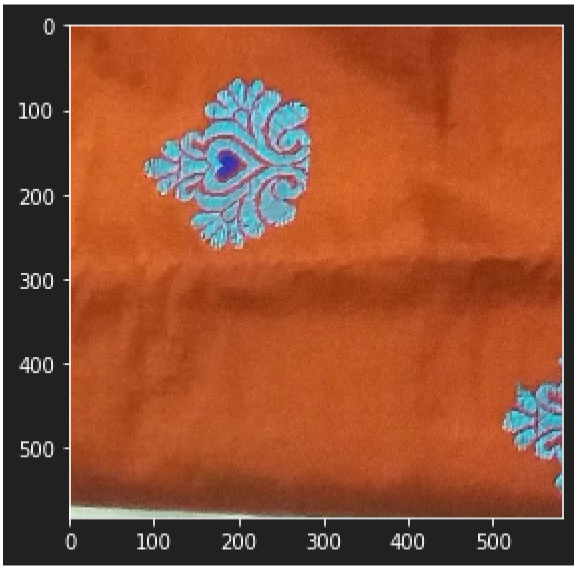


In [57]:

```
rows,colums = img_load.shape[:2]

#(col/2,rows/2) is the center of rotation for the image

# M is the cordinates of the center

M_load = cv2.getRotationMatrix2D((colums/2,rows/2),90,1)

dst_load = cv2.warpAffine(img_load,M_load,(colums,rows))
imshow(dst_load)
```

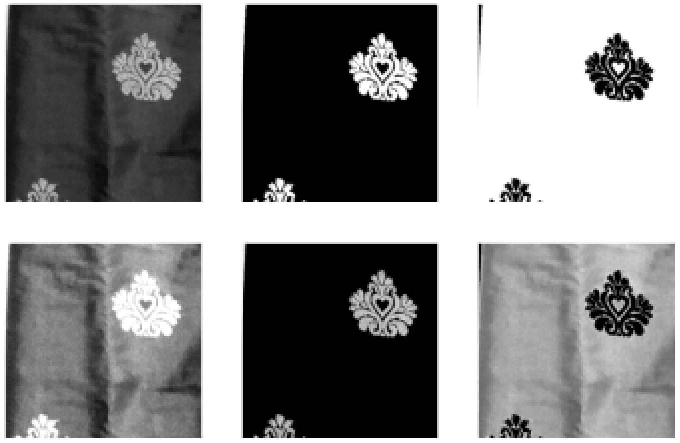
Out[57]:

```
<matplotlib.image.AxesImage at 0x23677e9b790>
```



In [58]:

```
ret,thresh_binary = cv2.threshold(gray_image,127,255,cv2.THRESH_BINARY)

ret,thresh_binary_inv = cv2.threshold(gray_image,127,255,cv2.THRESH_BINARY_INV)

ret,thresh_trunc = cv2.threshold(gray_image,127,255,cv2.THRESH_TRUNC)

ret,thresh_tozero = cv2.threshold(gray_image,127,255,cv2.THRESH_TOZERO)

ret,thresh_tozero_inv = cv2.threshold(gray_image,127,255,cv2.THRESH_TOZERO_INV)
#DISPLAYING THE DIFFERENT THRESHOLDING STYLES using OpenCV

names = ['Oiriginal Image','BINARY','THRESH_BINARY','THRESH_TRUNC','THRESH_TOZERO',

images = gray_image,thresh_binary,thresh_binary_inv,thresh_trunc,thresh_tozero,thr

for i in range(6):

    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')

    plt.title(names[i])

    plt.xticks([]),plt.yticks([])
```

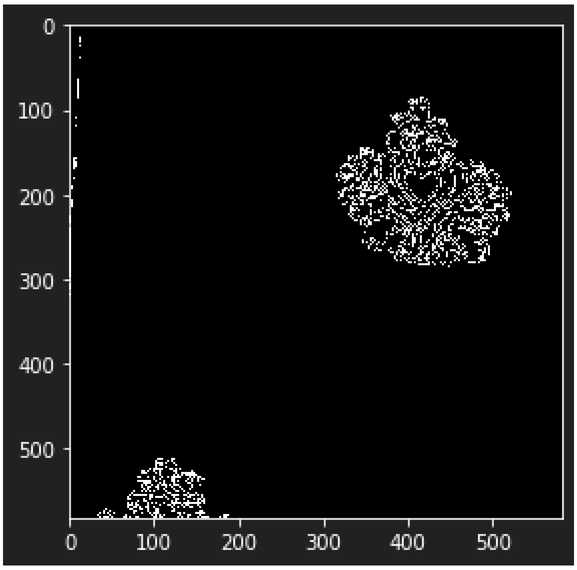


In [59]:

```
#calculate the edges using Canny edge algorithm

edges_of_image = cv2.Canny(img_load,100,200)
imshow(edges_of_image)
```

Out[59]:

```
<matplotlib.image.AxesImage at 0x2367806c4c0>
```



Splitting the Data into Training & Testing

In [7]:

```
x_train,x_test,y_train,y_test = train_test_split(data, result, test_size=0.2, shuf
```

In [8]:

```
# Normalization
x_train = x_train/255.0
x_test = x_test/255.0
```

In [9]:

```
#sklearn expects i/p to be 2d array-model.fit(x_train,y_train)=>reshape to 2d arra
nsamples, nx, ny, nrgb = x_train.shape
x_train2 = x_train.reshape((nsamples,nx*ny*nrgb))
```

In [10]:

```
#so,eventually,model.predict() should also be a 2d input
nsamples, nx, ny, nrgb = x_test.shape
x_test2 = x_test.reshape((nsamples,nx*ny*nrgb))
```

Random Forest

In [11]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [12]:

```
model=RandomForestClassifier()
```

In [13]:

```
model.fit(x_train2,y_train)
```

Out[13]:

```
RandomForestClassifier()
```

In [14]:

```
y_pred=model.predict(x_test2)
y_pred
```

```
Out[14]: array([[1., 0.],
               [1., 0.],
               [0., 1.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [0., 1.],
               [1., 0.],
               [1., 0.],
               [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

In [15]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
import numpy as np
```

In [60]:

```
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
0.9605263157894737
              precision    recall  f1-score   support

           0       1.00      0.95      0.97        58
           1       0.86      1.00      0.92        18

   micro avg       0.96      0.96      0.96        76
   macro avg       0.93      0.97      0.95        76
weighted avg       0.97      0.96      0.96        76
 samples avg       0.96      0.96      0.96        76
```

Model Building

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

In [41]:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(2, 2), input_shape=(128, 128, 3), padding = 'Sam
model.add(Conv2D(32, kernel_size=(2, 2),  activation ='relu', padding = 'Same'))


model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size = (2,2), activation ='relu', padding = 'Same'))
model.add(Conv2D(64, kernel_size = (2,2), activation ='relu', padding = 'Same'))

model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
model.compile(loss = "categorical_crossentropy", optimizer='Adamax')
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      416       
                                                                 
 conv2d_1 (Conv2D)           (None, 128, 128, 32)      4128      
                                                                 
 batch_normalization (BatchN  (None, 128, 128, 32)     128       
 ormalization)                                                   
                                                                 
 max_pooling2d (MaxPooling2D  (None, 64, 64, 32)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 64, 64, 32)        0         
                                                                 
 conv2d_2 (Conv2D)           (None, 64, 64, 64)        8256      
                                                                 
 conv2d_3 (Conv2D)           (None, 64, 64, 64)        16448     
                                                                 
 batch_normalization_1 (Batc  (None, 64, 64, 64)       256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)       0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 32, 32, 64)        0         
                                                                 
 flatten (Flatten)           (None, 65536)             0         
                                                                 
 dense (Dense)               (None, 512)               33554944  
                                                                 
 dropout_2 (Dropout)         (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 2)                 1026      
                                                                 
=================================================================
Total params: 33,585,602
Trainable params: 33,585,410
Non-trainable params: 192
_________________________________________________________________
None
```

In [42]:

```
y_train.shape
```

Out[42]:

```
(302, 2)
```

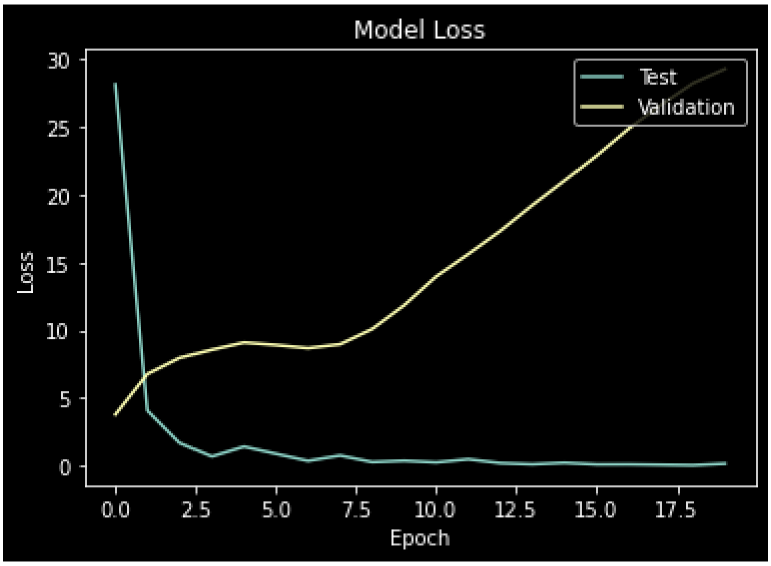
In [43]:

```
history = model.fit(x_train, y_train, epochs = 20, batch_size = 40, verbose = 1,va
Epoch 1/20
8/8 [==============================] - 21s 2s/step - loss: 28.1231 - val_loss: 3.7
905
Epoch 2/20
8/8 [==============================] - 13s 2s/step - loss: 4.0835 - val_loss: 6.77
73
Epoch 3/20
8/8 [==============================] - 12s 2s/step - loss: 1.7031 - val_loss: 7.96
37
Epoch 4/20
8/8 [==============================] - 13s 2s/step - loss: 0.6950 - val_loss: 8.56
59
Epoch 5/20
8/8 [==============================] - 11s 1s/step - loss: 1.4280 - val_loss: 9.09
77
Epoch 6/20
8/8 [==============================] - 12s 2s/step - loss: 0.8927 - val_loss: 8.91
22
Epoch 7/20
8/8 [==============================] - 11s 1s/step - loss: 0.3681 - val_loss: 8.68
10
Epoch 8/20
8/8 [==============================] - 10s 1s/step - loss: 0.7676 - val_loss: 8.96
77
Epoch 9/20
8/8 [==============================] - 10s 1s/step - loss: 0.2815 - val_loss: 10.0
897
Epoch 10/20
8/8 [==============================] - 10s 1s/step - loss: 0.3781 - val_loss: 11.8
448
Epoch 11/20
8/8 [==============================] - 10s 1s/step - loss: 0.2573 - val_loss: 13.9
670
Epoch 12/20
8/8 [==============================] - 10s 1s/step - loss: 0.4924 - val_loss: 15.6
124
Epoch 13/20
8/8 [==============================] - 10s 1s/step - loss: 0.2035 - val_loss: 17.3
186
Epoch 14/20
8/8 [==============================] - 10s 1s/step - loss: 0.1178 - val_loss: 19.2
191
Epoch 15/20
8/8 [==============================] - 10s 1s/step - loss: 0.2262 - val_loss: 21.0
284
Epoch 16/20
8/8 [==============================] - 10s 1s/step - loss: 0.1067 - val_loss: 22.8
740
Epoch 17/20
8/8 [==============================] - 11s 1s/step - loss: 0.1093 - val_loss: 24.8
715
Epoch 18/20
8/8 [==============================] - 11s 1s/step - loss: 0.0805 - val_loss: 26.6
067
Epoch 19/20
8/8 [==============================] - 11s 1s/step - loss: 0.0486 - val_loss: 28.2
087
Epoch 20/20
8/8 [==============================] - 10s 1s/step - loss: 0.1662 - val_loss: 29.2
612
```

Plotting Losses

In [44]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Test', 'Validation'], loc='upper right')
plt.show()
```



Just Checking the Model

In [45]:

```
def names(number):
    if number==0:
        return 'Its a handloom'
    else:
        return 'Its powerloom'
```

In [46]:

```
from matplotlib.pyplot import imshow
img = Image.open(r"D:\Arivu\Documents\project\Dataset\WhatsApp Image 2022-11-01 at
x = np.array(img.resize((128,128)))
x = x.reshape(1,128,128,3)
res = model.predict_on_batch(x)
classification = np.where(res == np.amax(res))[1][0]
imshow(img)
print( names(classification))
```

```
Its a handloom
```