CNN Image Classification

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
import os
import cv2
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,GlobalAveragePooling2D
from PIL import Image
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report
```

```
# Define the paths to your image and csv folders
train_val_dir = "C:\\Users\\335224\\Downloads\\Chart\\train_val"
test_dir = "C:\\Users\\335224\\Downloads\\Chart\\test"
train_path_labels = "C:\\Users\\335224\\Downloads\\Chart\\train_v"
train_val_labels = pd.read_csv(train_path_labels)
```

```
# Load training dataset in numpy array

images = []
labels = []

for filename in os.listdir(train_val_dir):
    if filename.endswith('.png'):
        # Load the images and resize them to (128, 128) with 3 color channels
        img = cv2.imread(os.path.join(train_val_dir, filename))
        img = cv2.resize(img, (128, 128))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#       img = Image.open(os.path.join(train_val_dir, filename))
        img_array = np.array(img)
        # Append the array to the list of images
        images.append(img_array)
        labels.append(filename)

# Convert the string labels to numerical labels
le = LabelEncoder()
labels = le.fit_transform(labels)
```

```
# Convert the lists to NumPy arrays
images = np.array(images)
labels = np.array(labels)
# Save the arrays in NumPy format
np.save('x_train.npy', images)
np.save('y_train.npy', labels)
x_train = np.load('x_train.npy')
y_train = np.load('y_train.npy')
```

```
x_train.shape
```

```
(1000, 128, 128, 3)
```

```
x_train[:5]
y_train[:5]
```

```
array([0, 1, 2, 3, 4], dtype=int64)
```

```
# Load test dataset in numpy array

images = []
labels = []

 for filename in os.listdir(test_dir):
     if filename.endswith('.png'):
         # Load the images and resize them to (128, 128) with 3 color channels
         img = cv2.imread(os.path.join(test_dir, filename))
         img = cv2.resize(img, (128, 128))
         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#        img = Image.open(os.path.join(test_dir, filename))
         img_array = np.array(img)
         # Append the array to the list of images
         images.append(img_array)
         labels.append(filename)

# Convert the string labels to numerical labels
le = LabelEncoder()
labels = le.fit_transform(labels)

# Convert the lists to NumPy arrays
```

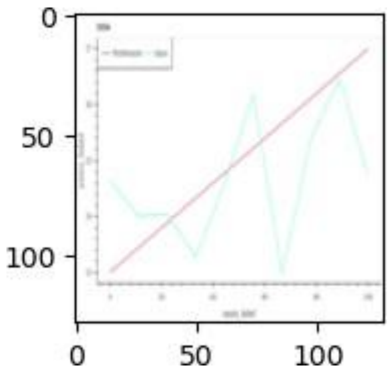
```
images = np.array(images)
labels = np.array(labels)

# Save the arrays in NumPy format
np.save('x_test.npy', images)
np.save('y_test.npy', labels)

x_test = np.load('x_test.npy')
y_test = np.load('y_test.npy')
```

```
x_test.shape
```

```
(50, 128, 128, 3)
```

```
# check the images loaded
plt.figure(figsize = (10,2))
plt.imshow(x_train[10])
plt.imshow(x_train[208])
plt.imshow(x_train[444])
```

```
<matplotlib.image.AxesImage at 0x1b79ae3b730>
```



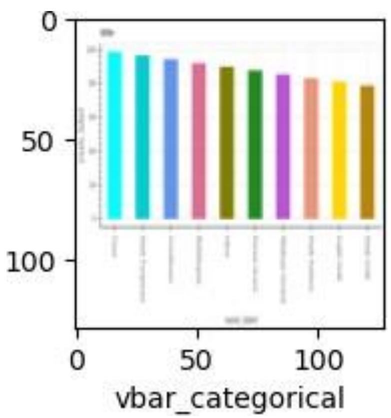

```
# define some classes from the images we have observed
image_classes = ['line', 'dot_line', 'hbar_categorical', 'vbar_categorical', 'pie']
image_classes[0]

# map the categories to the labels array i.e y_train
label_map = {'line': 0, 'dot_line': 1, 'hbar_categorical': 2, 'vbar_categorical': 3, 'pie': 4}
y_train = np.array([label_map[label] for label in train_val_labels['type']])
```

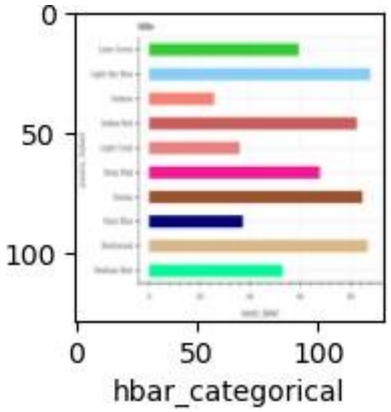
```
y_train
y_train.shape
y_test.shape
```

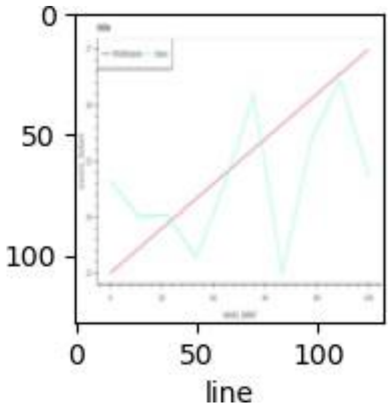
(50,)

```
# function to test the chart sample

def image_sample(x, y, index):
    plt.figure(figsize = (10,2))
    plt.imshow(x[index])
    #image_label = train_val_labels.iloc[index]['type']
    #plt.xlabel(image_label)
    plt.xlabel(image_classes[y[index]])
```

```
image_sample(x_train,y_train,0)
image_sample(x_train,y_train,208)
image_sample(x_train,y_train,444)
```








```
# normalize the image

x_train=x_train /255
x_test=x_train /255
```

```
x_test.shape
```

(1000, 128, 128, 3)

```python
# take the label for train data from csv file

y_train_index = train_val_labels['image_index']
y_train_type = train_val_labels['type']
```

```python
y_train_type[:5]
```

```
0    vbar_categorical
1    vbar_categorical
2    vbar_categorical
3    vbar_categorical
4    vbar_categorical
Name: type, dtype: object
```

```python
# Define the model architecture
model = Sequential([
    Flatten(input_shape=(128,128,3)),
    Dense(3000, activation='relu'),
    Dense(1000, activation='relu'),
    Dense(5, activation='softmax')
])

# Compile the model
model.compile(optimizer='SGD',  loss='sparse_categorical_crossentropy',  metrics=['accuracy'])

model.fit(x_train,y_train,epochs=10)
```

```
Epoch 1/10
32/32 [==============================]  - 16s 473ms/step - loss: 7.7490 - accuracy: 0.1940
Epoch 2/10
32/32 [==============================]  - 15s 475ms/step - loss: 1.6554 - accuracy: 0.2430
Epoch 3/10
32/32 [==============================]  - 15s 474ms/step - loss: 1.6163 - accuracy: 0.2260
Epoch 4/10
32/32 [==============================]  - 15s 469ms/step - loss: 1.6223 - accuracy: 0.2020
Epoch 5/10
32/32 [==============================]  - 15s 473ms/step - loss: 1.6105 - accuracy: 0.2300
Epoch 6/10
32/32 [==============================]  - 15s 471ms/step - loss: 1.6102 - accuracy: 0.2120
Epoch 7/10
32/32 [==============================]  - 15s 472ms/step - loss: 1.6109 - accuracy: 0.2030
Epoch 8/10
32/32 [==============================]  - 15s 472ms/step - loss: 1.6130 - accuracy: 0.1970
Epoch 9/10
32/32 [==============================]  - 15s 473ms/step - loss: 1.6102 - accuracy: 0.1960
Epoch 10/10
32/32 [==============================]  - 15s 472ms/step - loss: 1.6099 - accuracy: 0.2030

<keras.callbacks.History at 0x1b79a6714c0>
```

```
# Split the training images and labels into training and validation sets
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2, random_state=42)
```

```
model.evaluate(x_test,y_test)
```

```
7/7 [==============================] - 1s 106ms/step - loss: 1.6102 - accuracy: 0.1650
[1.6102104187011719, 0.16500000655651093]
```

```
y_pred = model.predict(x_test)
y_pred
y_pred_classes = [np.argmax(ele) for ele in y_pred]
# print("classificaton report : \n",classification_report(y_test,y_pred_classes))
```

```
7/7 [==============================] - 1s 111ms/step
```

```
# here we see the accuracy is very low and we need to modify our nn to add more layers for better accuracy
```

```python
# Print the shapes of the arrays to verify that they are loaded correctly
print("Train Images Shape:", x_train.shape)
print("Train Labels Shape:", y_train.shape)
print("Test Images Shape:", x_test.shape)
print("Test Labels Shape:", y_test.shape)
```

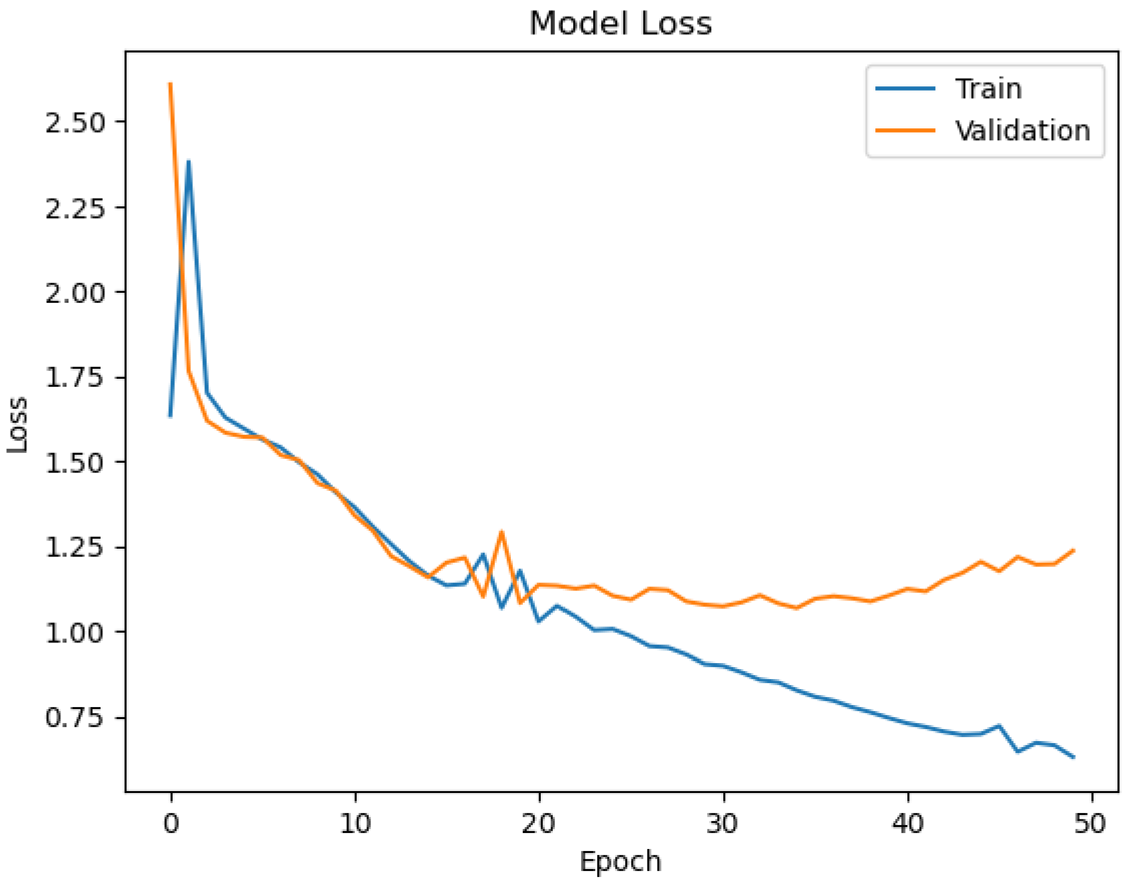
```
Train Images Shape: (800, 128, 128, 3)
Train Labels Shape: (800,)
Test Images Shape: (200, 128, 128, 3)
Test Labels Shape: (200,)
```

```python
# modifying the model architecture to cnn
cnn_model = Sequential([
    Conv2D(filters=16 ,kernel_size=(3,3), activation='relu', input_shape=(128,128,3)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(32, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(5, activation='softmax')
])

cnn_model.compile(optimizer='adam',  loss='sparse_categorical_crossentropy',  metrics=['accuracy'])

# Train the model
history = cnn_model.fit(x_train, y_train, batch_size=1000, epochs=50,validation_data=(x_test, y_test))

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

```
1/1 [==============================] - 14s 14s/step - loss: 1.6346 - accuracy: 0.1950 - val_loss: 2.6084 - val_accuracy: 0.1650
Epoch 2/50
1/1 [==============================] - 6s 6s/step - loss: 2.3806 - accuracy: 0.2087 - val_loss: 1.7656 - val_accuracy: 0.2950
Epoch 3/50
1/1 [==============================] - 6s 6s/step - loss: 1.7012 - accuracy: 0.3050 - val_loss: 1.6191 - val_accuracy: 0.1900
Epoch 4/50
1/1 [==============================] - 6s 6s/step - loss: 1.6281 - accuracy: 0.2025 - val_loss: 1.5837 - val_accuracy: 0.3500
Epoch 5/50
1/1 [==============================] - 6s 6s/step - loss: 1.5960 - accuracy: 0.3050 - val_loss: 1.5716 - val_accuracy: 0.3250
Epoch 6/50
1/1 [==============================] - 7s 7s/step - loss: 1.5638 - accuracy: 0.3013 - val_loss: 1.5697 - val_accuracy: 0.3000
Epoch 7/50
1/1 [==============================] - 6s 6s/step - loss: 1.5404 - accuracy: 0.3600 - val_loss: 1.5178 - val_accuracy: 0.4600
Epoch 8/50
1/1 [==============================] - 7s 7s/step - loss: 1.4966 - accuracy: 0.5200 - val_loss: 1.5030 - val_accuracy: 0.3500
Epoch 9/50
1/1 [==============================] - 9s 9s/step - loss: 1.4610 - accuracy: 0.3862 - val_loss: 1.4351 - val_accuracy: 0.4850
Epoch 10/50
1/1 [==============================] - 6s 6s/step - loss: 1.4076 - accuracy: 0.5425 - val_loss: 1.4130 - val_accuracy: 0.4400
Epoch 11/50
1/1 [==============================] - 6s 6s/step - loss: 1.3648 - accuracy: 0.4900 - val_loss: 1.3407 - val_accuracy: 0.5250
Epoch 12/50
1/1 [==============================] - 6s 6s/step - loss: 1.3073 - accuracy: 0.5575 - val_loss: 1.2953 - val_accuracy: 0.5250
Epoch 13/50
1/1 [==============================] - 6s 6s/step - loss: 1.2554 - accuracy: 0.5600 - val_loss: 1.2201 - val_accuracy: 0.5550
Epoch 14/50
1/1 [==============================] - 6s 6s/step - loss: 1.2049 - accuracy: 0.5412 - val_loss: 1.1901 - val_accuracy: 0.5450
Epoch 15/50
1/1 [==============================] - 6s 6s/step - loss: 1.1631 - accuracy: 0.5387 - val_loss: 1.1579 - val_accuracy: 0.5600
Epoch 16/50
1/1 [==============================] - 6s 6s/step - loss: 1.1340 - accuracy: 0.5325 - val_loss: 1.2010 - val_accuracy: 0.5400
Epoch 17/50
1/1 [==============================] - 6s 6s/step - loss: 1.1389 - accuracy: 0.5250 - val_loss: 1.2159 - val_accuracy: 0.5200
Epoch 18/50
1/1 [==============================] - 6s 6s/step - loss: 1.2253 - accuracy: 0.4750 - val_loss: 1.1012 - val_accuracy: 0.5850
Epoch 19/50
1/1 [==============================] - 6s 6s/step - loss: 1.0679 - accuracy: 0.5575 - val_loss: 1.2918 - val_accuracy: 0.4900
Epoch 20/50
1/1 [==============================] - 6s 6s/step - loss: 1.1772 - accuracy: 0.5200 - val_loss: 1.0819 - val_accuracy: 0.6400
Epoch 21/50
1/1 [==============================] - 6s 6s/step - loss: 1.0278 - accuracy: 0.6300 - val_loss: 1.1356 - val_accuracy: 0.5150
Epoch 22/50
1/1 [==============================] - 6s 6s/step - loss: 1.0737 - accuracy: 0.5550 - val_loss: 1.1331 - val_accuracy: 0.5200
```

```
1/1 [==============================]  - 6s 6s/step - loss: 1.0418 - accuracy: 0.6100 - val_loss: 1.1245 - val_accuracy: 0.5450
Epoch 24/50
1/1 [==============================]  - 6s 6s/step - loss: 1.0024 - accuracy: 0.5800 - val_loss: 1.1334 - val_accuracy: 0.5600
Epoch 25/50
1/1 [==============================]  - 6s 6s/step - loss: 1.0057 - accuracy: 0.5913 - val_loss: 1.1035 - val_accuracy: 0.5950
Epoch 26/50
1/1 [==============================]  - 6s 6s/step - loss: 0.9845 - accuracy: 0.5938 - val_loss: 1.0920 - val_accuracy: 0.6050
Epoch 27/50
1/1 [==============================]  - 6s 6s/step - loss: 0.9554 - accuracy: 0.6400 - val_loss: 1.1239 - val_accuracy: 0.5650
Epoch 28/50
1/1 [==============================]  - 6s 6s/step - loss: 0.9514 - accuracy: 0.6550 - val_loss: 1.1200 - val_accuracy: 0.5250
Epoch 29/50
1/1 [==============================]  - 6s 6s/step - loss: 0.9308 - accuracy: 0.6313 - val_loss: 1.0869 - val_accuracy: 0.5500
Epoch 30/50
1/1 [==============================]  - 6s 6s/step - loss: 0.9033 - accuracy: 0.6263 - val_loss: 1.0772 - val_accuracy: 0.5750
Epoch 31/50
1/1 [==============================]  - 6s 6s/step - loss: 0.8984 - accuracy: 0.6500 - val_loss: 1.0723 - val_accuracy: 0.6050
Epoch 32/50
1/1 [==============================]  - 6s 6s/step - loss: 0.8794 - accuracy: 0.6725 - val_loss: 1.0841 - val_accuracy: 0.5700
Epoch 33/50
1/1 [==============================]  - 6s 6s/step - loss: 0.8572 - accuracy: 0.6875 - val_loss: 1.1049 - val_accuracy: 0.5650
Epoch 34/50
1/1 [==============================]  - 6s 6s/step - loss: 0.8499 - accuracy: 0.6825 - val_loss: 1.0810 - val_accuracy: 0.5850
Epoch 35/50
1/1 [==============================]  - 6s 6s/step - loss: 0.8264 - accuracy: 0.6825 - val_loss: 1.0673 - val_accuracy: 0.5950
Epoch 36/50
1/1 [==============================]  - 6s 6s/step - loss: 0.8075 - accuracy: 0.6975 - val_loss: 1.0948 - val_accuracy: 0.6050
Epoch 37/50
1/1 [==============================]  - 6s 6s/step - loss: 0.7960 - accuracy: 0.6963 - val_loss: 1.1023 - val_accuracy: 0.5850
Epoch 38/50
1/1 [==============================]  - 6s 6s/step - loss: 0.7769 - accuracy: 0.6988 - val_loss: 1.0961 - val_accuracy: 0.6000
Epoch 39/50
1/1 [==============================]  - 6s 6s/step - loss: 0.7621 - accuracy: 0.7225 - val_loss: 1.0868 - val_accuracy: 0.6050
Epoch 40/50
1/1 [==============================]  - 6s 6s/step - loss: 0.7451 - accuracy: 0.7250 - val_loss: 1.1040 - val_accuracy: 0.6100
Epoch 41/50
1/1 [==============================]  - 5s 5s/step - loss: 0.7294 - accuracy: 0.7225 - val_loss: 1.1236 - val_accuracy: 0.6050
Epoch 42/50
1/1 [==============================]  - 6s 6s/step - loss: 0.7186 - accuracy: 0.7350 - val_loss: 1.1163 - val_accuracy: 0.5900
Epoch 43/50
1/1 [==============================]  - 6s 6s/step - loss: 0.7050 - accuracy: 0.7287 - val_loss: 1.1502 - val_accuracy: 0.5700
Epoch 44/50
1/1 [==============================]  - 6s 6s/step - loss: 0.6954 - accuracy: 0.7387 - val_loss: 1.1710 - val_accuracy: 0.5900
```

```
Epoch 45/50
1/1 [==============================]  - 5s 5s/step - loss: 0.6979 - accuracy: 0.7163 - val_loss: 1.2036 - val_accuracy: 0.5450
Epoch 46/50
1/1 [==============================]  - 6s 6s/step - loss: 0.7217 - accuracy: 0.7063 - val_loss: 1.1758 - val_accuracy: 0.5900
Epoch 47/50
1/1 [==============================]  - 6s 6s/step - loss: 0.6455 - accuracy: 0.7575 - val_loss: 1.2177 - val_accuracy: 0.5450
Epoch 48/50
1/1 [==============================]  - 5s 5s/step - loss: 0.6721 - accuracy: 0.7300 - val_loss: 1.1955 - val_accuracy: 0.5650
Epoch 49/50
1/1 [==============================]  - 5s 5s/step - loss: 0.6645 - accuracy: 0.7375 - val_loss: 1.1970 - val_accuracy: 0.5850
Epoch 50/50
1/1 [==============================]  - 6s 6s/step - loss: 0.6303 - accuracy: 0.7650 - val_loss: 1.2362 - val_accuracy: 0.5850
```

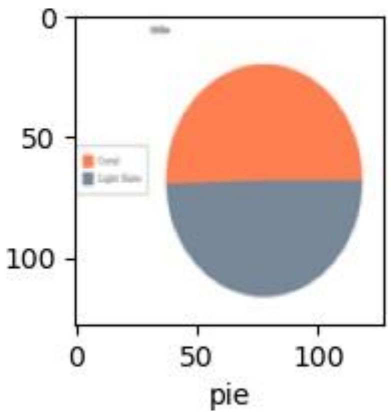


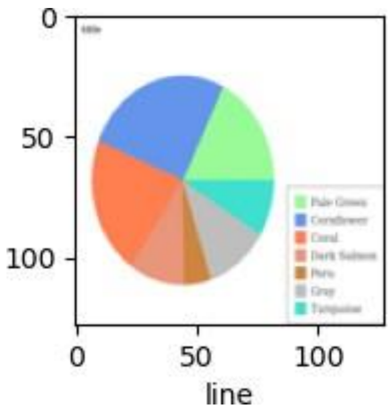
```python
cnn_model.evaluate(x_test,y_test)
```

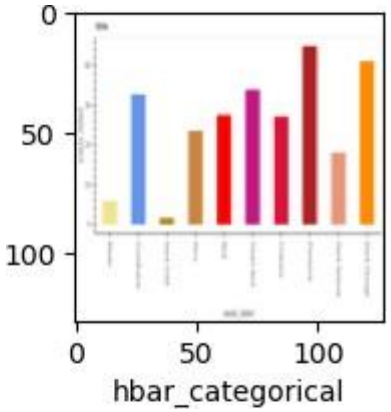
```
7/7 [==============================] - 0s 41ms/step - loss: 1.2362 - accuracy: 0.5850
```

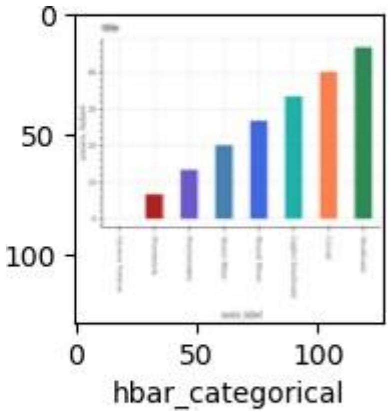
```
[1.2362478971481323, 0.5849999785423279]
```

```
image_sample(x_test,y_test,1)
image_sample(x_test,y_test,50)
image_sample(x_test,y_test,25)
image_sample(x_test,y_test,30)
```










```
# Observation:  we are able to see some wrong predictions here
```

```
y_pred = cnn_model.predict(x_test)
y_pred[:5]
```

```
7/7 [==============================] - 0s 39ms/step
```

```
array([[1.88634619e-01,  4.53442723e-01,  1.89865723e-01,  3.13057527e-02,
        1.36751115e-01],
       [1.16687104e-01, 1.73624381e-02,  6.09002418e-05,  7.50791188e-03,
        8.58381629e-01],
       [4.57645766e-02,  1.14214392e-02,  9.25663917e-05,  2.46962626e-03,
        9.40251827e-01],
       [5.41297436e-01, 5.38021093e-04,  9.51929204e-03,  3.77870835e-02,
        4.10858095e-01],
       [3.89628053e-01,  3.40805709e-01,  1.78500041e-01,  1.64468344e-02,
        7.46192932e-02]], dtype=float32)
```

```python
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

```
[1, 4, 4, 0, 0]
```

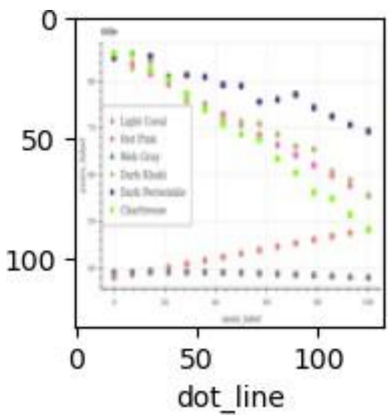
```python
y_test[:5]
```

```
array([0, 4, 4, 4, 0])
```

```python
# we can see some values are not matching here
```

```python
# test actual and predicted

#  image_sample(x_test,y_test,1)  #actual
#  image_classes[y_classes[1]]  #predicted


#  image_sample(x_test,y_test,10)  #actual
#  image_classes[y_classes[10]]  #predicted

image_sample(x_test,y_test,15)  #actual
image_classes[y_classes[15]]  #predicted
```

```
'dot_line'
```




```
# here we can see some values are not matching
```

```
print("classification report: \n", classification_report(y_test,y_classes))
```

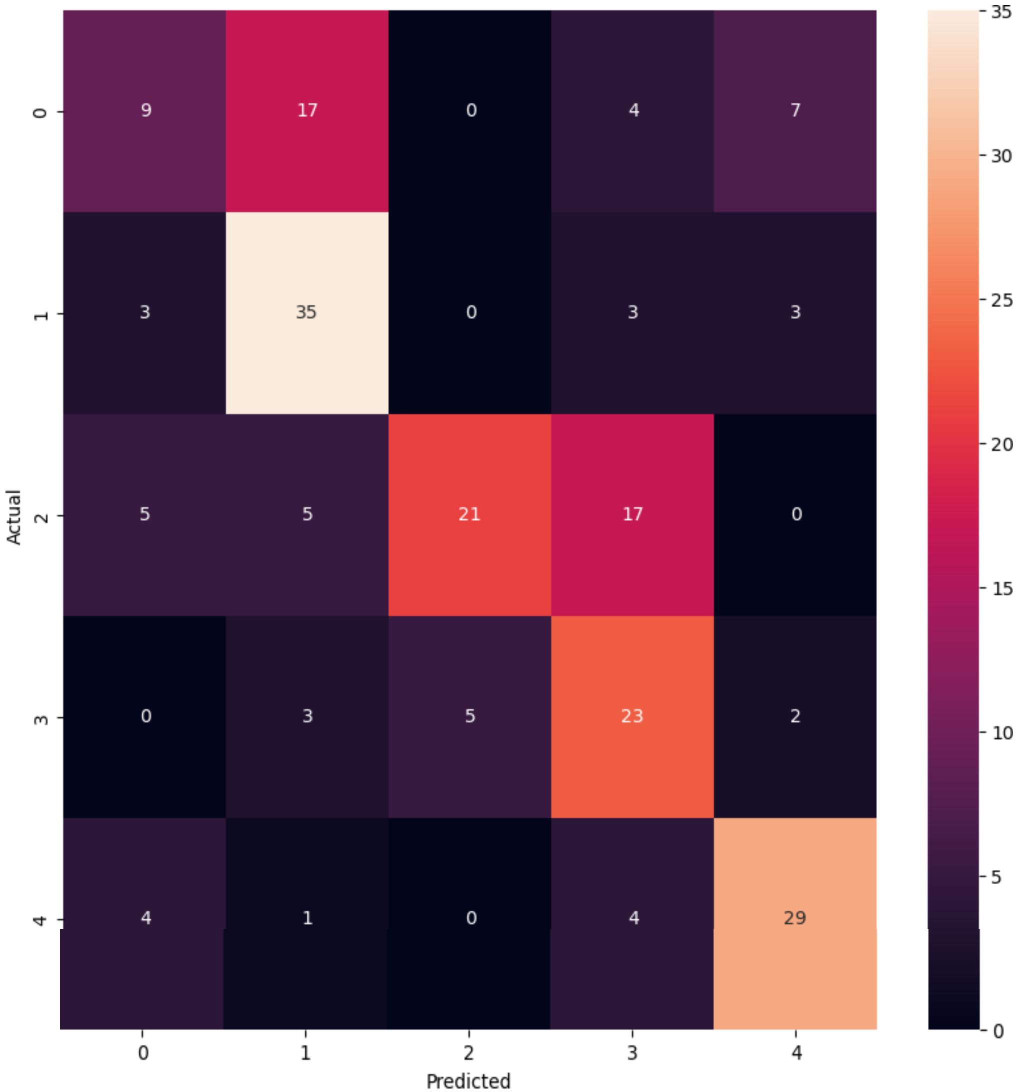
```
classification  report:
               precision    recall  f1-score   support

           0       0.43      0.24      0.31        37
           1       0.57      0.80      0.67        44
           2       0.81      0.44      0.57        48
           3       0.45      0.70      0.55        33
           4       0.71      0.76      0.73        38

    accuracy                           0.58       200
   macro avg       0.59      0.59      0.57       200
weighted avg       0.61      0.58      0.57       200
```

```
# Generating  confusion  Matrix
conf_mat = confusion_matrix(y_test, y_classes)

print('Confusion Matrix:')
print(conf_mat)
```

```
Confusion Matrix:
[[ 9 17  0  4  7]
 [ 3 35  0  3  3]
 [ 5  5 21 17  0]
 [ 0  3  5 23  2]
 [ 4  1  0  4 29]]
```

```python
# Plot the confusion matrix
import seaborn as sn
plt.figure(figsize = (10,10))
sn.heatmap(conf_mat,annot=True,fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Text(95.72222222222221, 0.5, 'Actual')
```



Actual
Predicted
0
1
2
3
4
9
17
0
4
7
3
35
0
3
3
5
5
21
17
0
0
3
5
23
2
4
1
0
4
29
35
30
25
20
15
10
5
0

```python
# for 50 iterations, we can see some promising accuracy, more training will be required for better accuracy
# in the confusion matrix, whatever is not in diagonal is considered an error
```

```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the pre-trained model
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```python
# Replace the final classification layer with a new layer
x = vgg16_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)

predictions = Dense(5, activation='softmax')(x)
pt_model = tf.keras.Model(inputs=vgg16_model.input, outputs=predictions)
```

```python
# Freeze the weights of all layers except the new classification layer
for layer in pt_model.layers:
    layer.trainable = False
```

```python
# Compile the model with categorical crossentropy loss and Adam optimizer
pt_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```python
# Print the summary of the model architecture
pt_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0

 global_average_pooling2d (G  (None, 512)               0
 lobalAveragePooling2D)
```

```
 dense_5 (Dense)             (None, 128)             65664

 dense_6 (Dense)             (None, 5)               645

=================================================================
Total params: 14,780,997
Trainable params: 0
Non-trainable params: 14,780,997
_________________________________________________________________
```

```python
# Set up data generators for image augmentation and feeding data to the model

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)
```

```python
train_generator = train_datagen.flow(x_train, y_train, batch_size=32)

test_generator = train_datagen.flow(x_test, y_test, batch_size=32)
```

```python
# Training the model with early stopping

from tensorflow.keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode='min', restore_best_weights=True)

history = pt_model.fit(train_generator, epochs=100, validation_data=test_generator, callbacks=[es])
```

```
Epoch 1/100
```