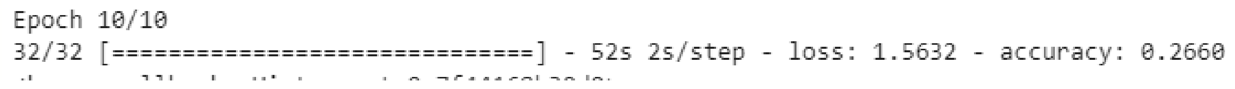
Problem 3: Chart Image Classification using CNN

Colab link :-
https://colab.research.google.com/drive/1_dNOY5ggO4jI4ntpn_FEeVtqC7lV0f37#scrollTo=yoSLp1gc9do
T

In this problem I need to use CNN to classify chart images such as “Line”,“Dot Line”,“HorizontalBar”,“Vertical Bar”, and “Pie” chart. I have used tensorflow and keras library etc to achieve this. First I have converted the training and test images into numpy array to be used further in the process. Then I have created a sequential model using 128*128*3 as input shape ,relu and softmax as activation functions. A rectified linear unit (ReLU) is an activation function that introduces the property of non-linearity to a deep learning model and solves the vanishing gradients issue. It interprets the positive part of its argument. Softmax extends this idea into a multi-class world. That is, Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would. Then I have compiled the model using SGD and sparse_categorical_crossentropy. SGD is the default optimizer for the python Keras library . SGD differs from regular gradient descent in the way it calculates the gradient. Instead of using all the training data to calculate the gradient per epoch, it uses a randomly selected instance from the training data to estimate the gradient. sparse_categorical_crossentropy is used as a loss function for multi-class classification model where the output label is assigned integer value (0, 1, 2, 3…). This loss function is mathematically same as the categorical_crossentropy. Then I have fit the model with 10 epochs.Below is the accuracy I have got out of it.

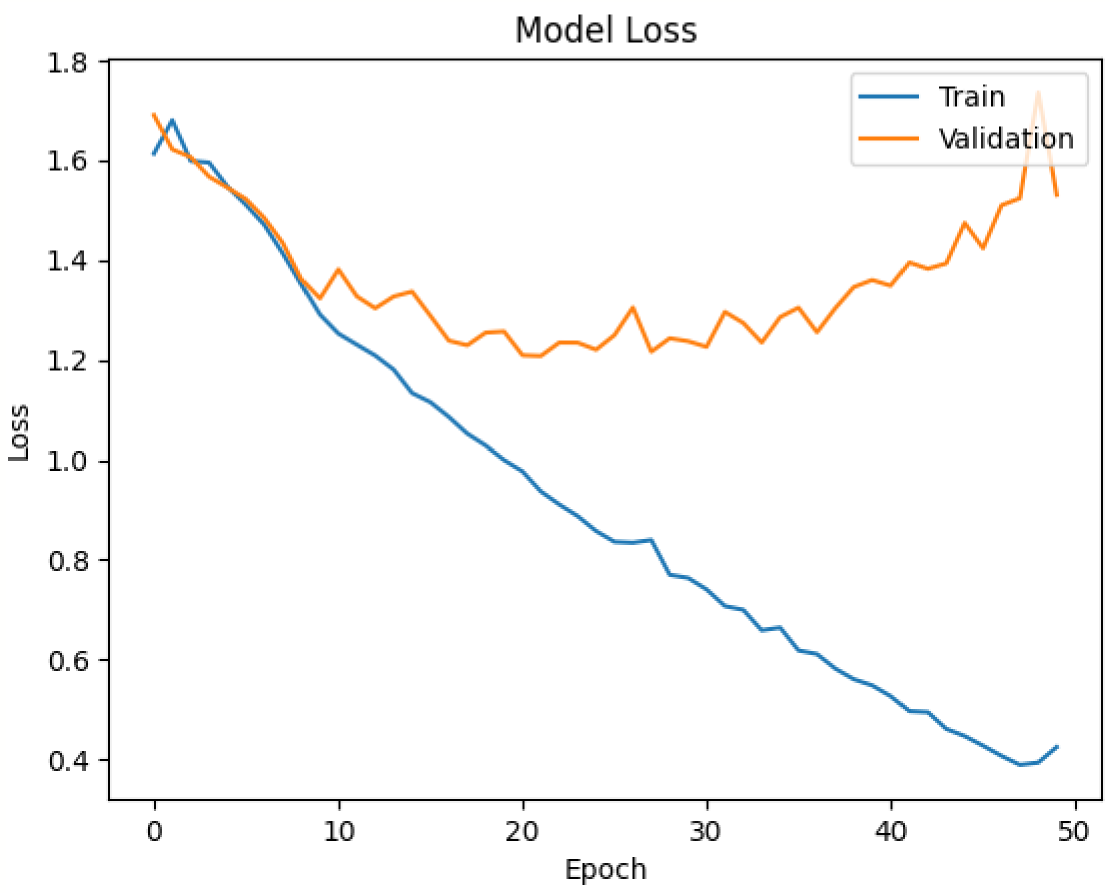


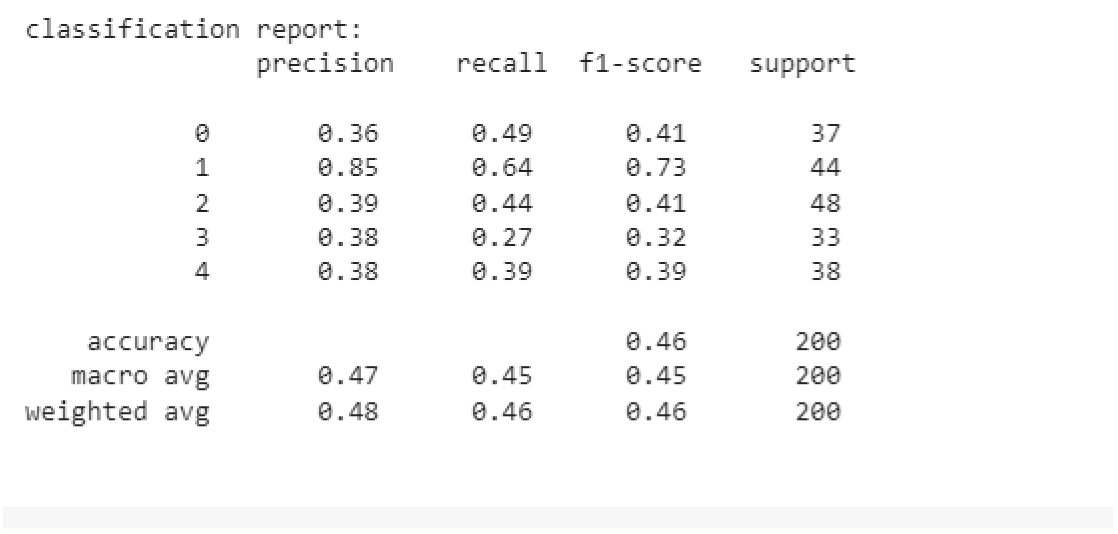
```
Epoch 10/10
32/32 [==============================] - 52s 2s/step - loss: 1.5632 - accuracy: 0.2660
```

Then I have split the dataset into training and validation set and evaluated the model and got accuracy as accuracy: 0.3200.
here we see the accuracy is very low and we need to modify our nn to add more layers for better accura cy.Hence I have used a CNN with Conv2D, MaxPooling2D, Conv2D, MaxPooling2D, Conv2D, MaxPooling2D and relu and softmax as activation functions. Keras conv2D which stands for convolution layer in a 2-dimensional pattern is responsible for generating the kernel of convolution which is then amalgamated with the other input layers of the Keras model so that the final resultant output will contain a tensor. Max pooling works by selecting the maximum value from every pool. Max Pooling retains the most prominent features of the feature map, and the returned image is sharper than the original image. Average pooling layer works by getting the average of the pool. I have compiled the model with adam optimizer and sparse_categorical_crossentropy. Adam optimizer is the extended version of stochastic gradient descent . Then I have fit the model with batch size as 1000 and epochs as 50.Below is the observation for same:-

```
1/1 [==============================] - 18s 18s/step - loss: 0.4614 - accuracy: 0.8438 - val_loss: 1.3929 - val_accuracy: 0.4650
Epoch 45/50
1/1 [==============================] - 19s 19s/step - loss: 0.4470 - accuracy: 0.8375 - val_loss: 1.4739 - val_accuracy: 0.4500
Epoch 46/50
1/1 [==============================] - 16s 16s/step - loss: 0.4277 - accuracy: 0.8550 - val_loss: 1.4233 - val_accuracy: 0.4800
Epoch 47/50
1/1 [==============================] - 16s 16s/step - loss: 0.4074 - accuracy: 0.8750 - val_loss: 1.5093 - val_accuracy: 0.4650
Epoch 48/50
1/1 [==============================] - 15s 15s/step - loss: 0.3894 - accuracy: 0.8737 - val_loss: 1.5234 - val_accuracy: 0.4550
Epoch 49/50
1/1 [==============================] - 15s 15s/step - loss: 0.3941 - accuracy: 0.8587 - val_loss: 1.7355 - val_accuracy: 0.4700
Epoch 50/50
1/1 [==============================] - 15s 15s/step - loss: 0.4252 - accuracy: 0.8350 - val_loss: 1.5303 - val_accuracy: 0.4550
```

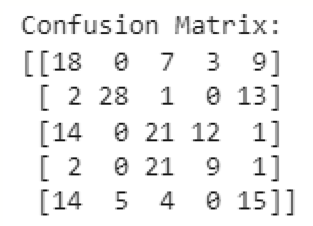
Model Loss



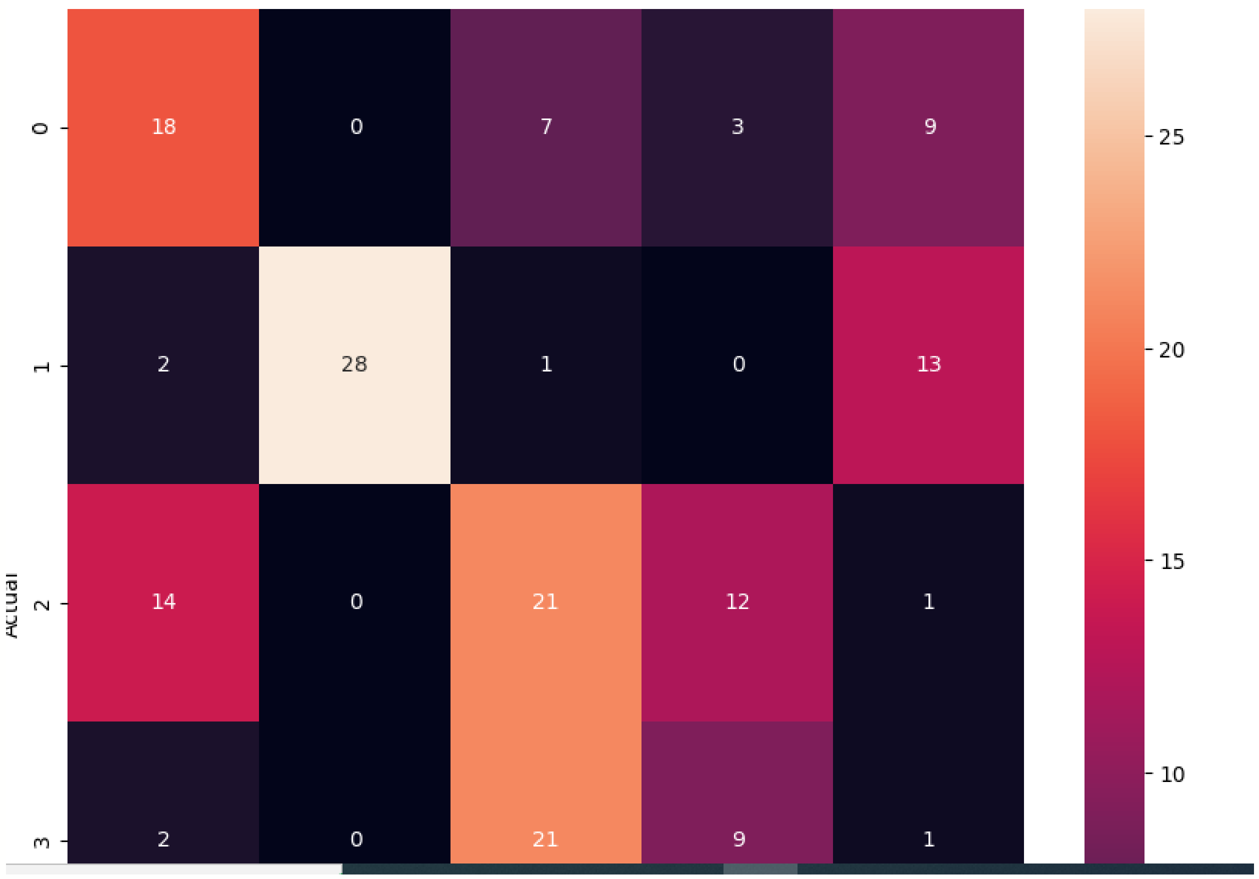


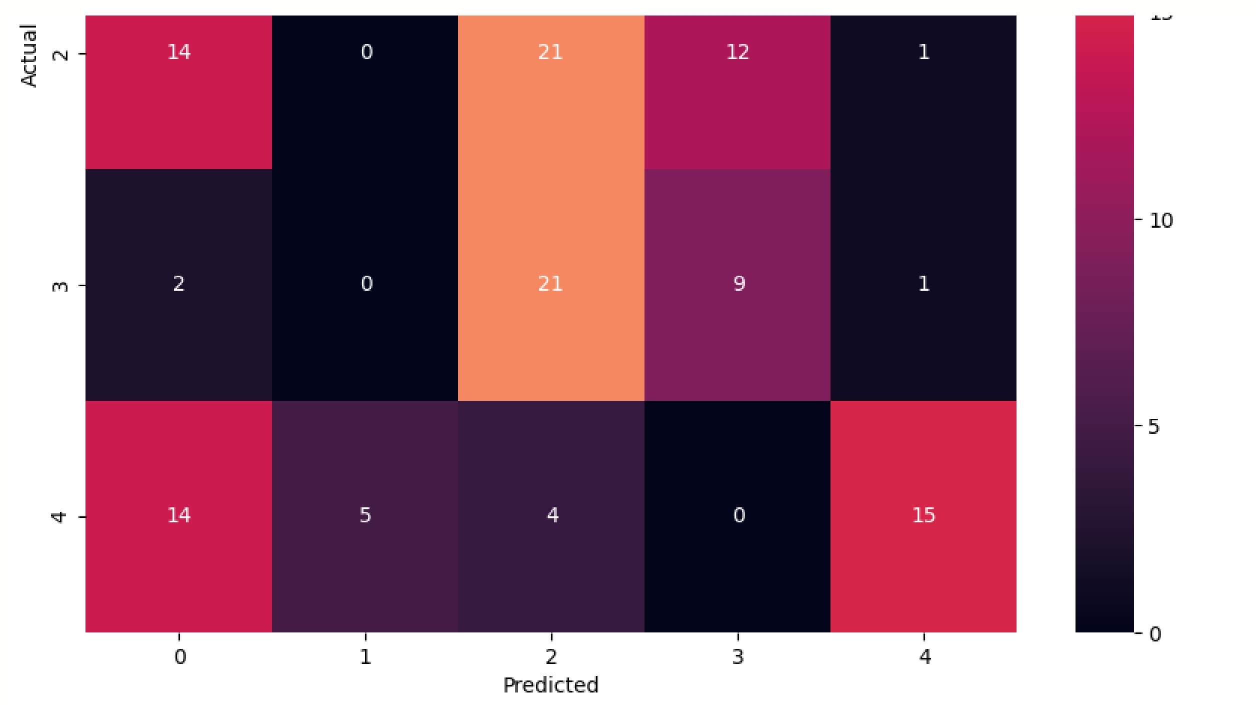
```
classification report:
              precision    recall  f1-score   support

           0       0.36      0.49      0.41        37
           1       0.85      0.64      0.73        44
           2       0.39      0.44      0.41        48
           3       0.38      0.27      0.32        33
           4       0.38      0.39      0.39        38

    accuracy                           0.46       200
   macro avg       0.47      0.45      0.45       200
weighted avg       0.48      0.46      0.46       200
```

Then I have generated the confusion matrix as below:-



```
Confusion Matrix:
[[18  0  7  3  9]
 [ 2 28  1  0 13]
 [14  0 21 12  1]
 [ 2  0 21  9  1]
 [14  5  4  0 15]]
```


Actual
0
1
2
3
18 0 7 3 9
2 28 1 0 13
14 0 21 12 1
2 0 21 9 1
25
20
15
10


Actual
2
3
4
14 0 21 12 1
2 0 21 9 1
14 5 4 0 15
0
1
2
3
4
Predicted
10
5
0

Then I have finetuned VGG16 to perform the same task. VGG16, as its name suggests, is a 16-layer deep neural network. VGG16 is thus a relatively extensive network with a total of 138 million parameters—it's huge even by today's standards. However, the simplicity of the VGGNet16 architecture is its main attraction. The VGGNet architecture incorporates the most important convolution neural network features. A VGG network consists of small convolution filters. VGG16 has three fully connected layers and 13 convolutional layers. Here is a quick outline of the VGG architecture:

- **Input—**VGGNet receives a 224×224 image input. In the ImageNet competition, the model's creators kept the image input size constant by cropping a 224×224 section from the center of each image.
- **Convolutional layers—**the convolutional filters of VGG use the smallest possible receptive field of 3×3. VGG also uses a 1×1 convolution filter as the input's linear transformation.
- **ReLu activation**—next is the Rectified Linear Unit Activation Function (ReLU) component, AlexNet's major innovation for reducing training time. ReLU is a linear function that provides a matching output for positive inputs and outputs zero for negative inputs. VGG has a set convolution stride of 1 pixel to preserve the spatial resolution after convolution (the stride value reflects how many pixels the filter "moves" to cover the entire space of the image).
- **Hidden layers—**all the VGG network's hidden layers use ReLU instead of Local Response Normalization like AlexNet. The latter increases training time and memory consumption with little improvement to overall accuracy.
- **Pooling layers–**A pooling layer follows several convolutional layers—this helps reduce the dimensionality and the number of parameters of the feature maps created by each convolution step. Pooling is crucial given the rapid growth of the number of available filters from 64 to 128, 256, and eventually 512 in the final layers.
- **Fully connected layers—**VGGNet includes three fully connected layers. The first two layers each have 4096 channels, and the third layer has 1000 channels, one for every class.

Below are some observations using finetuned VGG16:-

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (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_1   (None, 512)             0
(GlobalAveragePooling2D)

dense_12 (Dense)          (None, 128)             65664

dense_13 (Dense)          (None, 5)               645

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