Step 1: Import libraries

```
# Import libraries

import os
import zipfile
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

import time

import warnings
warnings.filterwarnings('ignore')
```

```
pip install -U kaleido
```

```
Requirement already satisfied: kaleido in /usr/local/lib/python3.10/dist-packages (0.2.1)
```

```
pip install git+https://github.com/hyperopt/hyperopt-sklearn
```

```
Collecting git+https://github.com/hyperopt/hyperopt-sklearn
  Cloning https://github.com/hyperopt/hyperopt-sklearn to /tmp/pip-req-build-krsr4ww4
  Running command git clone --filter=blob:none --quiet https://github.com/hyperopt/hyperopt-sklearn /tmp/pip-req-build-krsr4ww4
  Resolved https://github.com/hyperopt/hyperopt-sklearn to commit 4bc286479677a0bfd2178dac4546ea268b3f3b77
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: hyperopt>=0.2.6 in /usr/local/lib/python3.10/dist-packages (from hpsklearn==1.0.3) (0.2.7)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.10/dist-packages (from hpsklearn==1.0.3) (1.26.2)
Requirement already satisfied: scikit-learn>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from hpsklearn==1.0.3) (1.3.2)
Requirement already satisfied: scipy>=1.11.2 in /usr/local/lib/python3.10/dist-packages (from hpsklearn==1.0.3) (1.11.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from hyperopt>=0.2.6->hpsklearn==1.0.3) (1.16.0)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from hyperopt>=0.2.6->hpsklearn==1.0.3) (3.2.1)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from hyperopt>=0.2.6->hpsklearn==1.0.3) (0.18.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from hyperopt>=0.2.6->hpsklearn==1.0.3) (4.66.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from hyperopt>=0.2.6->hpsklearn==1.0.3) (2.2.1)
Requirement already satisfied: py4j in /usr/local/lib/python3.10/dist-packages (from hyperopt>=0.2.6->hpsklearn==1.0.3) (0.10.9.7)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.3.0->hpsklearn==1.0.3) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.3.0->hpsklearn==1.0.3) (3.2.0)
```

```
from hyperopt import tpe
from hpsklearn import HyperoptEstimator, svc
```

Step 2: Load the dataset. Split the data with test size 0.2.

```
os.getcwd()
```

```
'/content'
```

```
os.listdir()
```

```
['.config',
 '__MACOSX',
 'IndianCurrencyNotesDataset.zip',
 'IndianCurrencyNotesDataset',
 'sample_data']
```

```
f_name = 'IndianCurrencyNotesDataset.zip'
print('Size of ZIP file', os.path.getsize(f_name) / 1024 / 1024, 'MB')
```

```
Size of ZIP file 17.619593620300293 MB
```

```
# To unzip the archive 'IndianCurrencyNotesDataset.zip' in the current directory

start = time.perf_counter()   # Get current (relative) time in program

with zipfile.ZipFile(f_name, 'r') as fd:
    fd.extractall('.')

end = time.perf_counter()  # Get current (relative) time in program
print('Time to unzip', f_name, ':', end - start, 'seconds')
```

```
Time to unzip IndianCurrencyNotesDataset.zip : 0.32238248599969666 seconds
```

```
os.listdir()
```

```
['.config',
 '__MACOSX',
 'IndianCurrencyNotesDataset.zip',
 'IndianCurrencyNotesDataset',
 'sample_data']
```

```
os.listdir('./IndianCurrencyNotesDataset/')
```

```
['.DS_Store', 'AllImages']
```

```
def count_folders(directory_path):
  '''
    Function to count the number of non-files in a directory.
  '''
  count = 0
  non_folders = []

  for idx, path in enumerate(os.listdir(directory_path)):
      # Check if current path is not a file
      if not os.path.isfile(os.path.join(directory_path, path)):
        count += 1
      else:
        non_folders.append(idx)

  return count, non_folders
```

```
def count_images(directory_path):
  '''
    Function to count the number of non-files in a directory.
  '''

  if os.path.isfile(directory_path):
    return None

  count = 0
  for path in os.listdir(directory_path):
    # Check if current path is a file
    if os.path.isfile(os.path.join(directory_path, path)) and path.endswith('.jpg'):
      count += 1

  return count
```

```
dir_path = './IndianCurrencyNotesDataset/AllImages'

note_directories = os.listdir(dir_path)
n_folders, non_folders = count_folders(dir_path)
print(f'Number of sub-directories in /AllImages = {n_folders}    No of non-files = {non_folders}\n')
note_directories
```

```
Number of sub-directories in /AllImages = 7    No of non-files = [2]

['2000 Note',
 '200 Note',
 '.DS_Store',
 '20 Note',
 '500 Note',
 '100 Note',
 '50 Note',
 '10 Note']
```

```
for idx in range(1, len(note_directories)):
  note_dir = note_directories[idx]
  new_path = os.path.join(dir_path, note_dir)

  n_images = count_images(new_path)
  if n_images != None:
    n_non_images = len(os.listdir(new_path)) - n_images
    print(f'Number of images in /{note_dir} = {n_images}; No of non-images = {n_non_images}')
```

```
Number of images in /200 Note = 26; No of non-images = 0
Number of images in /20 Note = 25; No of non-images = 0
Number of images in /500 Note = 26; No of non-images = 0
Number of images in /100 Note = 25; No of non-images = 0
Number of images in /50 Note = 25; No of non-images = 0
Number of images in /10 Note = 26; No of non-images = 0
```

```
import cv2

img = cv2.imread('./IndianCurrencyNotesDataset/AllImages/10 Note/1.jpg')
img.shape
```

```
(168, 300, 3)
```

```
img2 = cv2.imread('./IndianCurrencyNotesDataset/AllImages/10 Note/10.jpg')
img2.shape
```

```
(239, 404, 3)
```

```
batch_size = 32
image_size = (224, 224)
```

```
# Generate Training and Testing dataset

train_dataset, test_dataset = image_dataset_from_directory(
    dir_path,
    validation_split = 0.2,
    subset = 'both',
    seed = 42,
    image_size = image_size,
    batch_size = batch_size,
)
```

```
Found 178 files belonging to 7 classes.
Using 143 files for training.
Using 35 files for validation.
```

```
train_dataset
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>
```

```
train_dataset.take(1)
```

```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>
```

```
for images, labels in train_dataset.take(1):
  print(labels, '\n')
  print(images.shape, type(images), images.dtype)
  break
```

```
tf.Tensor([4 0 3 5 5 2 5 1 0 4 3 5 5 3 4 2 0 5 2 0 1 4 5 5 3 1 0 2 2 4 1 3], shape=(32,), dtype=int32)

(32, 224, 224, 3) <class 'tensorflow.python.framework.ops.EagerTensor'> <dtype: 'float32'>
```

```
class_labels = train_dataset.class_names
class_labels
```

```
['10 Note',
 '100 Note',
 '20 Note',
 '200 Note',
 '2000 Note',
 '50 Note',
 '500 Note']
```

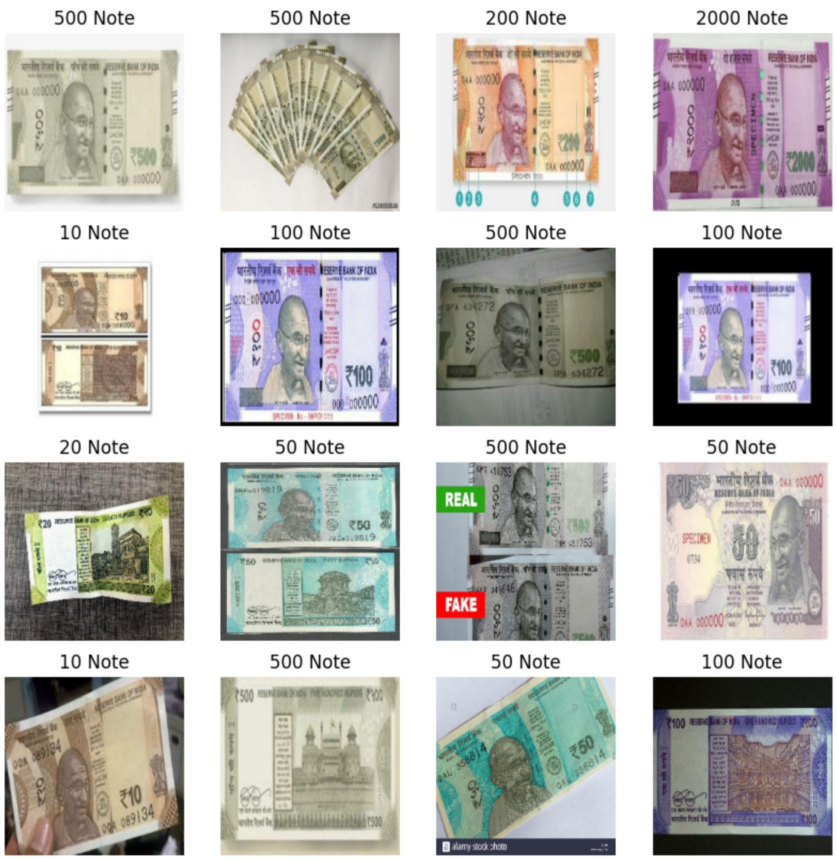
Step 3: Do some data augmentation like resizing and rotation (this is only done on the training set). Data augmentation is never done on the test set. Keep batch size 32 and rescale all the images to 224*224. Display few samples from training dataset.

```
# Display few samples from training dataset

plt.figure(figsize = (10, 10))

for images, labels in train_dataset.take(1):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.title(class_labels[labels[i]])
        plt.axis('off')

plt.show()
```




```
# Data Augmentation

data_augmentation = models.Sequential([
    layers.experimental.preprocessing.RandomRotation(0.2),  # Rotation
    layers.experimental.preprocessing.Resizing(224, 224),   # Resizing
])
```

```
# Applying Data Augmentation on training set

train_dataset = train_dataset.map(lambda x, y: (data_augmentation(x), y))
```

Step 4: Load pretrained ResNet50 model and extract features using this pretrained network.
Store the extracted features and its labels in two separate list.

```
pretrained_model = ResNet50(include_top = False, weights = 'imagenet', input_shape = (224, 224, 3))
```

```
pretrained_model
```

```
<keras.src.engine.functional.Functional at 0x7db3b9fbfa90>
```

```
pretrained_model.summary()
```

```
 ation)

 conv5_block2_2_conv (Conv2  (None, 7, 7, 512)            2359808   ['conv5_block2_1_relu[0][0]']
 D)

 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block2_2_conv[0][0]']
 rmalization)

 conv5_block2_2_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block2_2_bn[0][0]']
 ation)

 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)           1050624   ['conv5_block2_2_relu[0][0]']
 D)

 conv5_block2_3_bn (BatchNo  (None, 7, 7, 2048)           8192      ['conv5_block2_3_conv[0][0]']
 rmalization)

 conv5_block2_add (Add)      (None, 7, 7, 2048)           0         ['conv5_block1_out[0][0]',
                                                                     'conv5_block2_3_bn[0][0]']

 conv5_block2_out (Activati  (None, 7, 7, 2048)           0         ['conv5_block2_add[0][0]']
 on)

 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)            1049088   ['conv5_block2_out[0][0]']
 D)

 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block3_1_conv[0][0]']
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block3_1_bn[0][0]']
 ation)

 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)            2359808   ['conv5_block3_1_relu[0][0]']
 D)

 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block3_2_conv[0][0]']
 rmalization)

 conv5_block3_2_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block3_2_bn[0][0]']
 ation)

 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)           1050624   ['conv5_block3_2_relu[0][0]']
 D)

 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)           8192      ['conv5_block3_3_conv[0][0]']
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)           0         ['conv5_block2_out[0][0]',
                                                                     'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)           0         ['conv5_block3_add[0][0]']
 on)

==================================================================================================
Total params: 23587712 (89.98 MB)
Trainable params: 23534592 (89.78 MB)
Non-trainable params: 53120 (207.50 KB)
__________________________________________________________________________________________________
```

```
pretrained_model_shape = pretrained_model.layers[-1].output_shape
pretrained_model_shape
```

```
(None, 7, 7, 2048)
```

```
pretrained_model_shape[1] * pretrained_model_shape[2] * pretrained_model_shape[3]
```

```
100352
```

```
extracted_features_list = []
extracted_labels_list = []
```

```
for images, labels in train_dataset:
    features = pretrained_model.predict(images)
    features_flatten = np.reshape(features, (features.shape[0], -1))
    extracted_features_list.append(features_flatten)
    extracted_labels_list.append(labels.numpy())
```

```
1/1 [==============================] - 15s 15s/step
1/1 [==============================] - 12s 12s/step
1/1 [==============================] - 6s 6s/step
1/1 [==============================] - 7s 7s/step
1/1 [==============================] - 4s 4s/step
```

```
X_train = np.concatenate(extracted_features_list)
y_train = np.concatenate(extracted_labels_list)
```

```
print('Shape of X_train = ', X_train.shape)
pd.DataFrame(X_train).head()
```

```
Shape of X_train =  (143, 100352)
```

	0	1	2	3	4	5	6	7	8	9	...	100342	100343	100344	100345	100346	100347	10
0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.00
1	0.000000	0.0	0.0	0.0	0.0	0.0	0.740330	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.00
2	1.916649	0.0	0.0	0.0	0.0	0.0	0.136651	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	2.26
3	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.200996	0.0	0.00
4	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	2.59

5 rows × 100352 columns

```
print('Shape of y_train = ', y_train.shape)
pd.DataFrame(y_train).head()
```

```
Shape of y_train =  (143,)
```

	0
0	6
1	6
2	4
3	1
4	1

Step 5: Now train a SVM classifier over extracted features and labels.

```
svm_clf = SVC()
svm_clf.fit(X_train, y_train)
```

```
▾ SVC
SVC()
```

```
extracted_features_list_test = []
extracted_labels_list_test = []
```

```
for images, labels in test_dataset:
    features = pretrained_model.predict(images)
    features_flatten = np.reshape(features, (features.shape[0], -1))
    extracted_features_list_test.append(features_flatten)
    extracted_labels_list_test.append(labels.numpy())
```

```
1/1 [==============================] - 6s 6s/step
1/1 [==============================] - 1s 833ms/step
```

```
X_test = np.concatenate(extracted_features_list_test)
y_test = np.concatenate(extracted_labels_list_test)
```

```
print('Shape of X_test = ', X_test.shape)
pd.DataFrame(X_test).head()
```

```
Shape of X_test =  (35, 100352)
```

	0	1	2	3	4	5	6	7	8	9	...	100342	100343	100344	100345	100346	100347	100348	
0	0.0	0.000000	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.
1	0.0	0.000000	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.
2	0.0	0.000000	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.
3	0.0	0.333771	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.
4	0.0	0.000000	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	17.

5 rows × 100352 columns

```
print('Shape of y_test = ', y_test.shape)
pd.DataFrame(y_test).head()
```

```
Shape of y_test =  (35,)
```

	0
0	5
1	6
2	2
3	2
4	4

Step 6: Make predictions after you fit the SVM classifier.

```
y_pred = svm_clf.predict(X_test)
```

```
print('Shape of y_pred = ', y_pred.shape)
pd.DataFrame(y_pred).head()
```

```
Shape of y_pred =  (35,)
```

	0
0	1
1	6
2	1
3	0

Step 7: Calculate accuracy and print confusion matrix

```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

```
0.3142857142857143
```

```
cf_matrix = confusion_matrix(y_test, y_pred)      # Index = Actual;  Column = Predicted
```

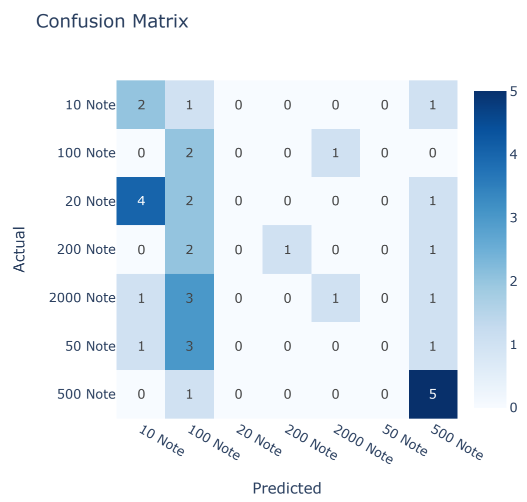
```
cf_matrix
```

```
array([[2, 1, 0, 0, 0, 0, 1],
       [0, 2, 0, 0, 1, 0, 0],
       [4, 2, 0, 0, 0, 0, 1],
       [0, 2, 0, 1, 0, 0, 1],
       [1, 3, 0, 0, 1, 0, 1],
       [1, 3, 0, 0, 0, 0, 1],
       [0, 1, 0, 0, 0, 0, 5]])
```

```
import plotly.express as px
```

```
fig = px.imshow(cf_matrix, text_auto = True, aspect = 'auto', color_continuous_scale = 'blues', width=500, height=500,
                title = 'Confusion Matrix',
                labels = dict(x = 'Predicted', y = 'Actual'),
                x = test_dataset.class_names,
                y = test_dataset.class_names)

fig.show()
fig.write_image('Confusion Matrix.png')
```



Improving Accuracy by using Hyperparameter tuning

```
optimal_svm_model = HyperoptEstimator(classifier = svc('mySVC'),
                                  preprocessing = [] ,
                                  algo = tpe.suggest,
                                  max_evals = 200,
                                  trial_timeout = 60)
```

```
optimal_svm_model.fit(X_train, y_train)
```

```
100%|██████████| 156/156 [00:03<00:00,  3.61s/trial, best loss: 0.3793103448275862]
100%|██████████| 157/157 [00:03<00:00,  3.22s/trial, best loss: 0.3793103448275862]
100%|██████████| 158/158 [00:02<00:00,  2.28s/trial, best loss: 0.3793103448275862]
100%|██████████| 159/159 [00:02<00:00,  2.75s/trial, best loss: 0.3793103448275862]
100%|██████████| 160/160 [00:02<00:00,  2.41s/trial, best loss: 0.3793103448275862]
100%|██████████| 161/161 [00:03<00:00,  3.39s/trial, best loss: 0.3793103448275862]
100%|██████████| 162/162 [00:02<00:00,  2.81s/trial, best loss: 0.3793103448275862]
100%|██████████| 163/163 [00:02<00:00,  2.95s/trial, best loss: 0.3793103448275862]
100%|██████████| 164/164 [00:02<00:00,  2.71s/trial, best loss: 0.3793103448275862]
100%|██████████| 165/165 [00:02<00:00,  2.80s/trial, best loss: 0.3793103448275862]
100%|██████████| 166/166 [00:03<00:00,  3.52s/trial, best loss: 0.3793103448275862]
100%|██████████| 167/167 [00:02<00:00,  2.27s/trial, best loss: 0.3793103448275862]
100%|██████████| 168/168 [00:02<00:00,  2.29s/trial, best loss: 0.3793103448275862]
100%|██████████| 169/169 [00:02<00:00,  2.59s/trial, best loss: 0.3793103448275862]
100%|██████████| 170/170 [00:03<00:00,  3.48s/trial, best loss: 0.3793103448275862]
100%|██████████| 171/171 [00:03<00:00,  3.96s/trial, best loss: 0.3793103448275862]
100%|██████████| 172/172 [00:02<00:00,  2.28s/trial, best loss: 0.3793103448275862]
100%|██████████| 173/173 [00:02<00:00,  2.24s/trial, best loss: 0.3793103448275862]
100%|██████████| 174/174 [00:02<00:00,  2.32s/trial, best loss: 0.3793103448275862]
100%|██████████| 175/175 [00:02<00:00,  2.82s/trial, best loss: 0.3793103448275862]
100%|██████████| 176/176 [00:03<00:00,  3.75s/trial, best loss: 0.3793103448275862]
100%|██████████| 177/177 [00:03<00:00,  3.15s/trial, best loss: 0.3793103448275862]
100%|██████████| 178/178 [00:02<00:00,  2.70s/trial, best loss: 0.3793103448275862]
100%|██████████| 179/179 [00:02<00:00,  2.65s/trial, best loss: 0.3793103448275862]
100%|██████████| 180/180 [00:03<00:00,  3.47s/trial, best loss: 0.3793103448275862]
100%|██████████| 181/181 [00:03<00:00,  3.32s/trial, best loss: 0.3793103448275862]
100%|██████████| 182/182 [00:02<00:00,  2.71s/trial, best loss: 0.3793103448275862]
100%|██████████| 183/183 [00:02<00:00,  2.27s/trial, best loss: 0.3793103448275862]
100%|██████████| 184/184 [00:02<00:00,  2.97s/trial, best loss: 0.3793103448275862]
100%|██████████| 185/185 [00:04<00:00,  4.30s/trial, best loss: 0.3793103448275862]
100%|██████████| 186/186 [00:02<00:00,  2.39s/trial, best loss: 0.3793103448275862]
100%|██████████| 187/187 [00:02<00:00,  2.28s/trial, best loss: 0.3793103448275862]
100%|██████████| 188/188 [00:02<00:00,  2.81s/trial, best loss: 0.3793103448275862]
100%|██████████| 189/189 [00:03<00:00,  3.02s/trial, best loss: 0.3793103448275862]
100%|██████████| 190/190 [00:03<00:00,  3.79s/trial, best loss: 0.3793103448275862]
100%|██████████| 191/191 [00:02<00:00,  2.39s/trial, best loss: 0.3793103448275862]
100%|██████████| 192/192 [00:02<00:00,  2.69s/trial, best loss: 0.3793103448275862]
100%|██████████| 193/193 [00:02<00:00,  2.89s/trial, best loss: 0.3793103448275862]
100%|██████████| 194/194 [00:02<00:00,  2.98s/trial, best loss: 0.3793103448275862]
100%|██████████| 195/195 [00:03<00:00,  3.76s/trial, best loss: 0.3793103448275862]
100%|██████████| 196/196 [00:02<00:00,  2.57s/trial, best loss: 0.3793103448275862]
100%|██████████| 197/197 [00:03<00:00,  3.13s/trial, best loss: 0.3793103448275862]
100%|██████████| 198/198 [00:04<00:00,  4.35s/trial, best loss: 0.3793103448275862]
100%|██████████| 199/199 [00:03<00:00,  3.76s/trial, best loss: 0.3793103448275862]
100%|██████████| 200/200 [00:02<00:00,  2.77s/trial, best loss: 0.3793103448275862]
```

```
optimal_svm_model._best_iters
```

```
best_svm_model = optimal_svm_model.best_model()
best_svm_model
```

```
{'learner': SVC(C=4.292085277542328, coef0=0.18803681576159076, degree=4, gamma='auto',
     kernel='sigmoid', random_state=1, shrinking=False,
     tol=0.0013134222530283514),
 'preprocs': (),
 'ex_preprocs': ()}
```

```
svm_clf = best_svm_model['learner']
```

```
svm_clf.fit(X_train, y_train)
```

```
SVC
SVC(C=4.292085277542328, coef0=0.18803681576159076, degree=4, gamma='auto',
    kernel='sigmoid', random_state=1, shrinking=False,
    tol=0.0013134222530283514)
```

```
y_pred_svm = svm_clf.predict(X_test)
```

```
print('Shape of y_pred = ', y_pred_svm.shape)
pd.DataFrame(y_pred_svm).head()
```

```
Shape of y_pred =  (35,)
```

	0
0	1
1	6
2	2
3	5
4	4

Step 7: Calculate accuracy and print confusion matrix

```
accuracy = accuracy_score(y_test, y_pred_svm)
accuracy
```

```
0.4857142857142857
```

```
1 - 0.3793103448275862
```

```
0.6206896551724138
```

```
cf_matrix = confusion_matrix(y_test, y_pred_svm)      # Index = Actual;  Column = Predicted
```

```
cf_matrix
```

```
array([[2, 0, 1, 0, 0, 0, 1],
       [0, 3, 0, 0, 0, 0, 0],
```

```
            [1, 1, 3, 0, 0, 2, 0],
            [0, 1, 1, 2, 0, 0, 0],
            [0, 3, 0, 0, 3, 0, 0],
            [0, 2, 0, 0, 0, 1, 2],
            [0, 3, 0, 0, 0, 0, 3]])
```

```
import plotly.express as px
```

```
fig = px.imshow(cf_matrix, text_auto = True, aspect = 'auto', color_continuous_scale = 'blues', width=500, height=500,
                title = 'Confusion Matrix with Hyperparameter tuning for SVM',
                labels = dict(x = 'Predicted', y = 'Actual'),
                x = test_dataset.class_names,
                y = test_dataset.class_names)

fig.show()
fig.write_image('Confusion Matrix with Hyperparameter tuning for SVM.png')
```