CS867 - Computer Vision **NUST**

Traffic Sign Classification

Lecturer: Dr. Muhammad Moazam Fraz *Author: Sarfraz Ahmad*

1 Introduction

Fields like machine learning, and its subsets, deep learning and neural networks, have greatly advanced in recent years as a result of the rising volatility, necessity, and applicability of artificial intelligence. Software and tools like classifiers, which feed enormous amounts of data, evaluate it, and extract valuable characteristics, are required for training. All of the pixels in a digital image are intended to be classified into one of many classes using the classification process. The classification process is often carried out using multi-spectral data, and each pixel's spectral pattern is employed as the numerical basis for classification. The goal of image classification is to identify and describe the features that appear in a picture in terms of the real object that these features actually represent on the ground as a distinct gray level (or colour). The most crucial aspect of digital image analysis is probably image classification. In the realm of computer vision, image classification has been a crucial problem since classifying different objects is a difficult operation. The labelling of images into one of a number of predetermined classes is referred to as image classification. A given image may be categorised into possibly n different classes. It might be time-consuming to manually review and classify large numbers of images, therefore it would be really helpful if we could automate the entire procedure using computer vision. Another excellent example of how image classification is used in practise is the development of autonomous vehicles. We need classifiers to achieve the highest level of accuracy because of the applications, which include automated image organization, stock photography and video websites, visual search for better product discoverability, large visual databases, image and face recognition on social networks, and many more.

A typical Image Classification design cycle is based on following steps:

- **Image Pre-processing:** The purpose of this process is to increase some key visual properties while suppressing undesired distortions, with the goal of providing better image data for computer vision models to work with. Image pre-processing steps include reading the image, resizing the image, and adding data (Gray scaling of image, Reflection, Gaussian Blurring, Histogram, Equalization, Rotation, and Translation).
- **Detection of an object:** Detection refers to the localization of an object which means the segmentation of the image and identifying the position of the object of interest.
- **Feature extraction and training:** This is a vital stage wherein statistical or deep learning techniques are used to pinpoint the image's most intriguing patterns, features that may be exclusive to a class and later aid the model in differentiating across classes. Model training is the process through which the model learns the features from the dataset.
- **Classification of the object:** This step categorizes detected objects into predefined classes by using a suitable classification technique that compares the image patterns with the target patterns.

In this work, the author has implemented a traffic sign classifier for the subset of public traffic signs dataset.

2 Dataset

In this work, the author has implemented traffic sign classifiers for the subset of public traffic signs dataset [1]. Images in subset of original dataset belong to 10 different classes of traffic signs and images from a particular class are placed in a separate folder with image label as folder name. Each class contain 80 training samples and 20 test samples, this makes a total of 1000 images with 800 training images and 200 test images. Images in each folder are divided in training and test set and this information is stored in the form of image name in separate train and test text files. In order to read image files of this dataset, the train and test text files are first used to create data-frame with "folder name", "file name", "file type" (train or test), "file path" and "label" as columns. OpenCV's image read library is utilized to read and store these images on training and test variables by utilizing the aforementioned dataframe.

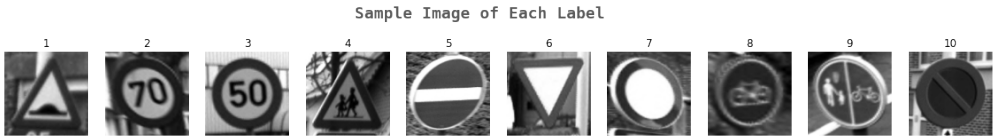


Figure 1: Sample Image from Each Class

3 Feature Extractions and Machine Learning

After reading and loading images, these images are passed to feature descriptors and output of these descriptors are passed to machine learning algorithms.

In this work, I have utilized following feature extraction methods:

- Histogram of Gradients (HOG)
- Scale Invariant Feature Transform (SIFT)

Moreover, following machine learning algorithms are used as classifiers:

- Support Vector Machines
- Logistic Regression
- Decision Tree
- K Nearest Neighbors

4 Results

With HOG feature extractor, 4 machine learning models are trained. SVM and Logistic Regression provide 100% accuracy on test set. While, decision tree and KNN provide 91% and 99% accuracy respectively. When SIFT with K-Means is used to learn the bag of visual words. The accuracy score for feeded SVM model is 74%. For each of the algorithm, both qualitative and quantitative results are computed in notebooks. The Snapshots of the notebook are added at the end of this file.

[1] https://www.kaggle.com/datasets/abhi8923shriv/belgium-ts/versions/9/metadata

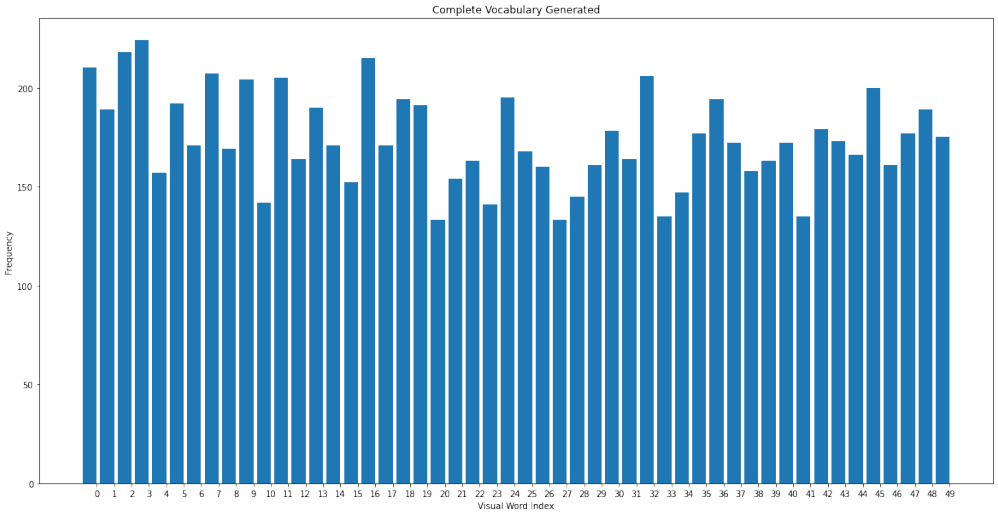


Figure 2: Bag of Visual Words created using SIFT and KMeans

5 Code Availability

The code is available at the following GitHub repository. Hog Descriptors.ipynb and SIFT Descriptors.ipynb contains code for each of the feature descriptor, respectively. Dataset and various confusion matrix images are also added. Code is extensively commented for better understanding.

https://github.com/SarfrazAhmad307/Traffic-Sign-Claasification-Using-Hand-Crafted-Features

Hog Descriptors

October 16, 2022

0.1 Importing Libraries

[2]:
```
import os
import random
import pandas as pd
import numpy as np
import cv2
from tqdm import tqdm
from cv2 import HOGDescriptor

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import LeaveOneOut, cross_validate,␣
↪cross_val_score, StratifiedShuffleSplit

import seaborn as sns
import matplotlib.pyplot as plt

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

0.2 Load and View Images

[3]:
```
folders = os.listdir("CS67_Fall2022_ A1_Dataset")
if ".DS_Store" in folders:
    folders.remove(".DS_Store")

folders = sorted(folders, key=int)
```

```
data_type = ["train", "test"] # txt Files in Dataset folders
df_list = [] # List that will contain the dataframe dict

# In each folder there are 2 txt files, train.txt and test.txt
# train.txt files from each folder are taken and image names in these files are␣
 ↪stored as filename in dict
# Same procedure is then followed for test.txt files
for i in data_type:
    for j in folders:
        with open(f"CS67_Fall2022_ A1_Dataset/{j}/{i}.txt") as f:
            for line in f:
                df_dict = {}
                df_dict["folder"] = j
                df_dict["filename"] = line
                df_dict["file_type"] = i    # Train or Test
                df_dict["file_path"] = f"CS67_Fall2022_ A1_Dataset/{j}/{line}".
 ↪strip()
                df_dict["label"] = int(j)   # Integer Label
                df_list.append(df_dict)

df = pd.DataFrame.from_dict(df_list)
df.head(10)
```

[3]:

	folder	filename	file_type	\
0	00001	00468_00002.ppm\n	train	
1	00001	00475_00000.ppm\n	train	
2	00001	00475_00001.ppm\n	train	
3	00001	00475_00002.ppm\n	train	
4	00001	00801_00000.ppm\n	train	
5	00001	00801_00001.ppm\n	train	
6	00001	00801_00002.ppm\n	train	
7	00001	00806_00000.ppm\n	train	
8	00001	00806_00001.ppm\n	train	
9	00001	00806_00002.ppm\n	train	

	file_path	label
0	CS67_Fall2022_ A1_Dataset/00001/00468_00002.ppm	1
1	CS67_Fall2022_ A1_Dataset/00001/00475_00000.ppm	1
2	CS67_Fall2022_ A1_Dataset/00001/00475_00001.ppm	1
3	CS67_Fall2022_ A1_Dataset/00001/00475_00002.ppm	1
4	CS67_Fall2022_ A1_Dataset/00001/00801_00000.ppm	1
5	CS67_Fall2022_ A1_Dataset/00001/00801_00001.ppm	1
6	CS67_Fall2022_ A1_Dataset/00001/00801_00002.ppm	1
7	CS67_Fall2022_ A1_Dataset/00001/00806_00000.ppm	1
8	CS67_Fall2022_ A1_Dataset/00001/00806_00001.ppm	1
9	CS67_Fall2022_ A1_Dataset/00001/00806_00002.ppm	1

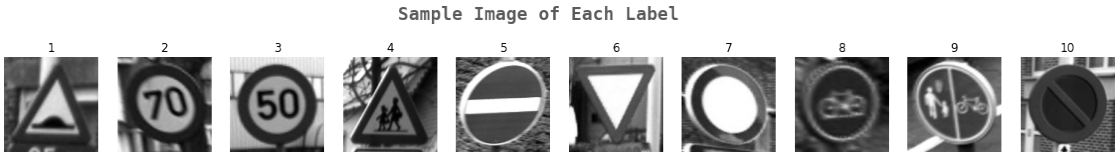
[4]:
```
# Initialize lists to store train and test images
X_train = []
y_train = []
X_test = []
y_test = []

# iterate over the dataframe rows. For each row check type of file that whether␣
 ↪it is train or test.
for _, row in tqdm(df.iterrows(), total=df.shape[0]):
    if row.file_type == "train":
        img = cv2.imread(row.file_path) # Read the file using filepath given in␣
 ↪dataframe.
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to grayscale
        img = cv2.resize(img, (128, 128))
        X_train.append(img) # Append the loaded image in training data list
        y_train.append(row.label)
    else:
        img = cv2.imread(row.file_path) # Read the file using filepath given in␣
 ↪dataframe.
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to grayscale
        img = cv2.resize(img, (128, 128))
        X_test.append(img) # Append the loaded image in test data list
        y_test.append(row.label)

# Convert train and test data to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)

X_test = np.array(X_test)
y_test = np.array(y_test)

print("Training Set Shape: ", X_train.shape)
print("Test Set Shape: ", X_test.shape)
```

```
100%|      | 1000/1000 [00:00<00:00, 2874.27it/s]
```

```
Training Set Shape:  (800, 128, 128)
Test Set Shape:  (200, 128, 128)
```

[5]:
```
# Visiualize 1 image from each of the class

k=0
fig, ax = plt.subplots(1,10,figsize=(20,10))    # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
```

```
    j=0
    while True :
        if y_train[j]==i:
            ax[k].imshow(X_train[j], 'gray')
            ax[k].set_title(y_train[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```



0.3 Hog Descriptor

[6]:
```
# Initialization of Hog Descriptor
winSize = (128,128)
blockSize = (16,16)
blockStride = (8,8)
cellSize = (16,16)
nbins = 9
derivAperture = 1
winSigma = -1.
histogramNormType = 0
L2HysThreshold = 0.2
gammaCorrection = 1
nlevels = 64

signedGradients = True

hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,
cellSize,nbins,derivAperture,
winSigma,histogramNormType,L2HysThreshold,
gammaCorrection,nlevels, signedGradients)
```

[7]:
```
# Initialize lists that will store train and test hog descriptors
train_hg_desc = []
test_hg_desc = []

for i in data_type: # This for loop will handle iterate over "train" amd "test"␣
 ↪and will handle X_train and X_test
```

```
    if i=="train":
        for img in X_train:
            descriptor = hog.compute(img)    # Hog descriptor for each training␣
↪image
            train_hg_desc.append(descriptor)
    else:
        for img in X_test:
            descriptor = hog.compute(img)    # Hog descriptor for each test image
            test_hg_desc.append(descriptor)
```

0.4 Machine Learning Classifiers

0.4.1 SVM

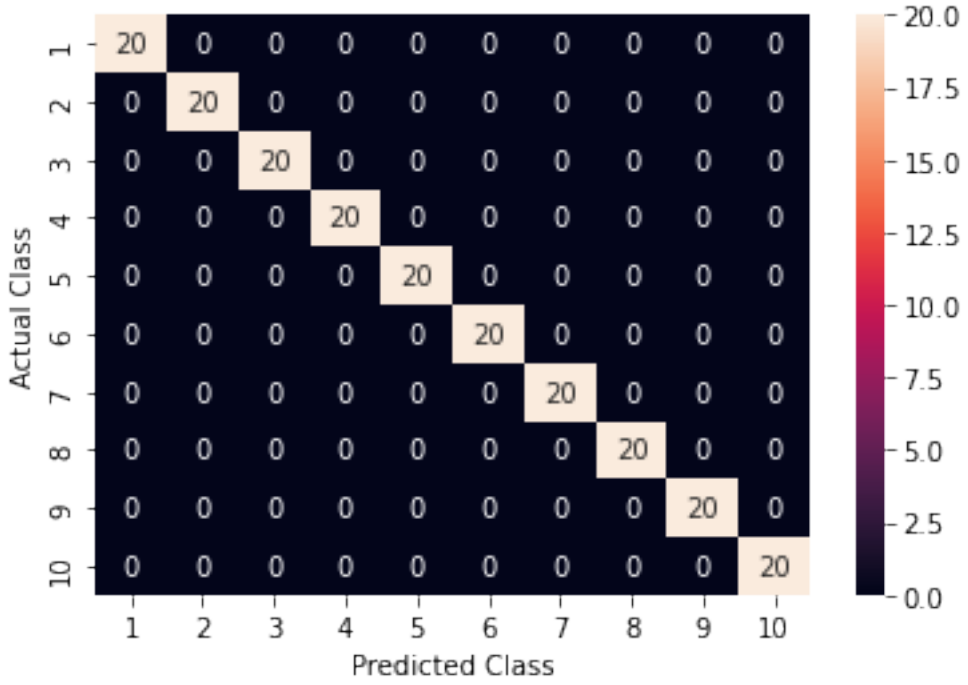
[11]:
```
def SVM(X,Y,x,y):
  # Define parameter grid for Grid Search
  param_grid = {'C': [0.1, 1, 10, 100],
                'kernel': ['poly', 'rbf', 'sigmoid']}

  svm=SVC(random_state=42)  # SVM model initialization
  svm_cv = GridSearchCV(svm, param_grid, scoring='accuracy')

  # fitting the model for grid search
  svm_cv.fit(X,Y)
  svm=svm_cv.best_estimator_  # Best model parameters are stored
  print ('Best accuracy and parameters are: ', svm_cv.best_score_, svm_cv.
↪best_params_)

  print('\n Test Results \n')
  y_p=svm_cv.best_estimator_.predict(x)

  x_axis_labels = sorted(np.unique(Y)) # labels for x-axis
  y_axis_labels = sorted(np.unique(Y)) # labels for y-axis

  sns.heatmap(confusion_matrix(y, y_p), annot = True,␣
↪xticklabels=x_axis_labels, yticklabels=y_axis_labels)
  plt.xlabel("Predicted Class")
  plt.ylabel("Actual Class")
  print(classification_report(y, y_p))

  return y_p
```

Quantitative Results

[12]:
```
y_pred = SVM(train_hg_desc, y_train, test_hg_desc, y_test)
```

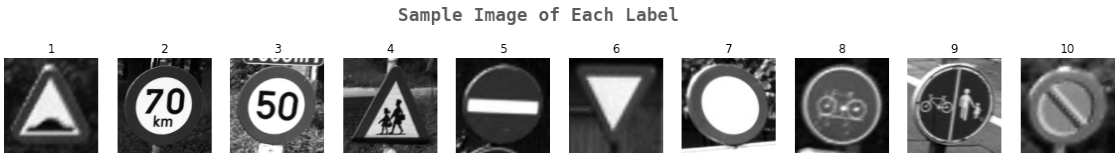
```
Best accuracy and parameters are:  0.9974999999999999 {'C': 0.1, 'kernel':
'poly'}

 Test Results
```

	precision	recall	f1-score	support
1	1.00	1.00	1.00	20
2	1.00	1.00	1.00	20
3	1.00	1.00	1.00	20
4	1.00	1.00	1.00	20
5	1.00	1.00	1.00	20
6	1.00	1.00	1.00	20
7	1.00	1.00	1.00	20
8	1.00	1.00	1.00	20
9	1.00	1.00	1.00	20
10	1.00	1.00	1.00	20
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200



Qualitative Results

[13]:
```
# Correctly Classified Test Images from each class
k=0
fig, ax = plt.subplots(1,10,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True:
        if y_test[j]==i and y_pred[j]==i:
            ax[k].imshow(X_test[j], 'gray')
            ax[k].set_title(y_pred[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```



0.4.2 Logistic Regression

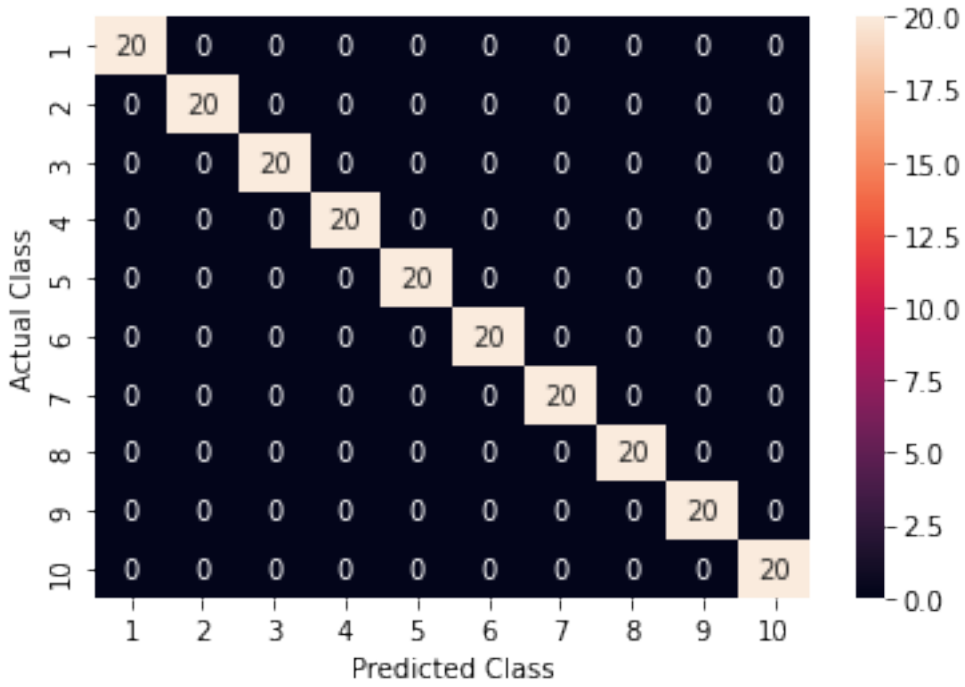
[15]:
```
def LRegression(X,Y,x,y):
  # Define parameter grid for Grid Search
  param_grid = {"C":np.logspace(-3,3,7),
                'penalty':['l1','l2','elasticnet','none']}

  LR=LogisticRegression(random_state=42)  # Logistic Regression Initialization
  lr_cv = GridSearchCV(LR, param_grid, scoring='accuracy')

  # fitting the model for grid search
  lr_cv.fit(X, Y)
  lr=lr_cv.best_estimator_  # Best model parameters are stored
  print('Best accuracy and parameters are: ', lr_cv.best_score_, lr_cv.
 ↪best_params_)

  print('\n Test Results \n')
  y_p=lr_cv.best_estimator_.predict(x)

  x_axis_labels = sorted(np.unique(Y)) # labels for x-axis
  y_axis_labels = sorted(np.unique(Y)) # labels for y-axis
```

```
    sns.heatmap(confusion_matrix(y, y_p), annot = True,␣
↪xticklabels=x_axis_labels, yticklabels=y_axis_labels)
    plt.xlabel("Predicted Class")
    plt.ylabel("Actual Class")
    print(classification_report(y, y_p))

    return y_p
```

Quantitative Results

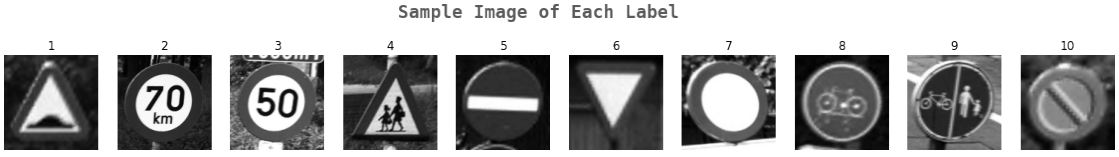
[16]:
```
y_pred = LRegression(train_hg_desc, y_train, test_hg_desc, y_test)
```

```
Best accuracy and parameters are:  1.0 {'C': 0.001, 'penalty': 'none'}

 Test Results

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        20
           2       1.00      1.00      1.00        20
           3       1.00      1.00      1.00        20
           4       1.00      1.00      1.00        20
           5       1.00      1.00      1.00        20
           6       1.00      1.00      1.00        20
           7       1.00      1.00      1.00        20
           8       1.00      1.00      1.00        20
           9       1.00      1.00      1.00        20
          10       1.00      1.00      1.00        20

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```



Qualitative Results

[17]:
```
# Correctly Classified Test Images from each class
k=0
fig, ax = plt.subplots(1,10,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True:
        if y_test[j]==i and y_pred[j]==i:
            ax[k].imshow(X_test[j], 'gray')
            ax[k].set_title(y_pred[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```



0.4.3 Decision Tree

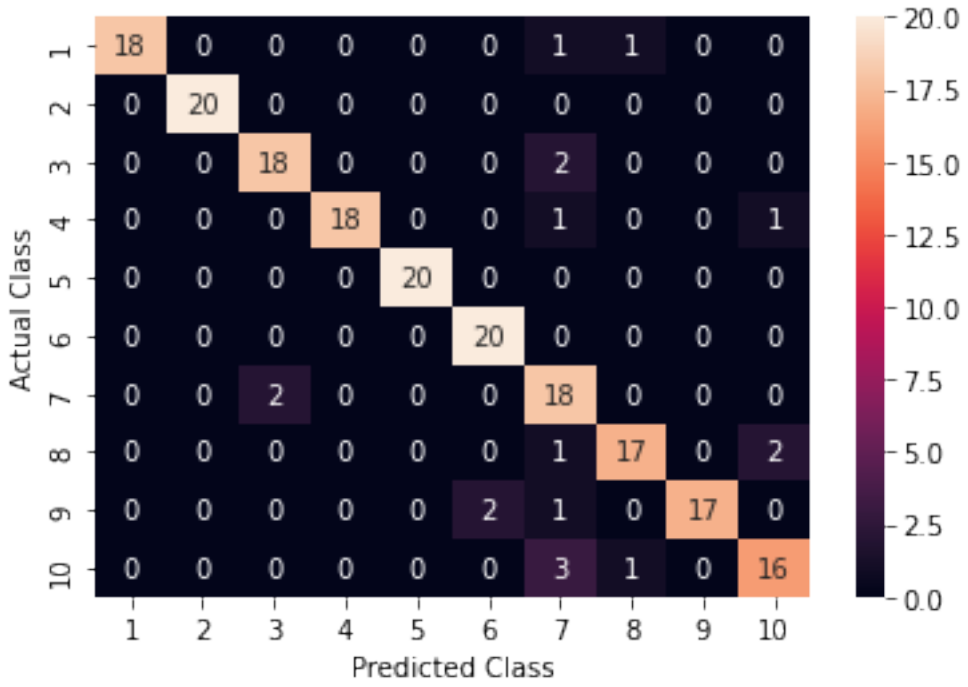
[18]:
```
def Tree(X,Y,x,y):
  # Define parameter grid for Grid Search
  param_grid = {'max_depth':range(2, 20),
                'criterion':['gini', 'entropy']}

  DT=DecisionTreeClassifier(random_state=42)  # Decision Tree Initialization
  tree_cv = GridSearchCV(DT, param_grid, scoring='accuracy')

  # fitting the model for grid search
  tree_cv.fit(X, Y)
  tree=tree_cv.best_estimator_  # Best model parameters are stored
  print('Best accuracy and parameters are: ', tree_cv.best_score_, tree_cv.
↪best_params_)

  print('\n Test Results \n')
  y_p=tree_cv.best_estimator_.predict(x)

  x_axis_labels = sorted(np.unique(Y)) # labels for x-axis
  y_axis_labels = sorted(np.unique(Y)) # labels for y-axis
  sns.heatmap(confusion_matrix(y, y_p), annot = True,␣
↪xticklabels=x_axis_labels, yticklabels=y_axis_labels)
  plt.xlabel("Predicted Class")
  plt.ylabel("Actual Class")
  print(classification_report(y, y_p))

  return y_p
```

Quantitative Results

[19]:
```
y_pred = Tree(train_hg_desc, y_train, test_hg_desc, y_test)
```

```
Best accuracy and parameters are:  0.91625 {'criterion': 'entropy', 'max_depth':
5}

 Test Results

              precision    recall  f1-score   support

           1       1.00      0.90      0.95        20
           2       1.00      1.00      1.00        20
           3       0.90      0.90      0.90        20
           4       1.00      0.90      0.95        20
```

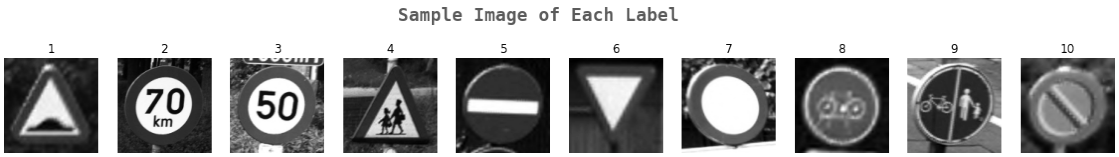
```
           5       1.00      1.00      1.00        20
           6       0.91      1.00      0.95        20
           7       0.67      0.90      0.77        20
           8       0.89      0.85      0.87        20
           9       1.00      0.85      0.92        20
          10       0.84      0.80      0.82        20

    accuracy                           0.91       200
   macro avg       0.92      0.91      0.91       200
weighted avg       0.92      0.91      0.91       200
```



Qualitative Results

[24]:
```
# Correctly Classified Test Images from each class
k=0
fig, ax = plt.subplots(1,10,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True:
        if y_test[j]==i and y_pred[j]==i:
            ax[k].imshow(X_test[j], 'gray')
```

```
            ax[k].set_title(y_pred[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```



[23]:
```
# Incorrectly Classified Test Images from each class
k=0
fig, ax = plt.subplots(1,7,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True:
        if j==200:
            break
        if y_test[j]==i and y_pred[j]!=i:
            ax[k].imshow(X_test[j], 'gray')
            ax[k].set_title(y_pred[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```

0.4.4 K Nearest Neighbors

[27]:
```
def KNN(X,Y,x,y):
  #create a dictionary of all values we want to test for n_neighbors
  param_grid = {'n_neighbors': np.arange(1, 10),
                'metric': ['minkowski', 'euclidean', 'manhattan']}
```

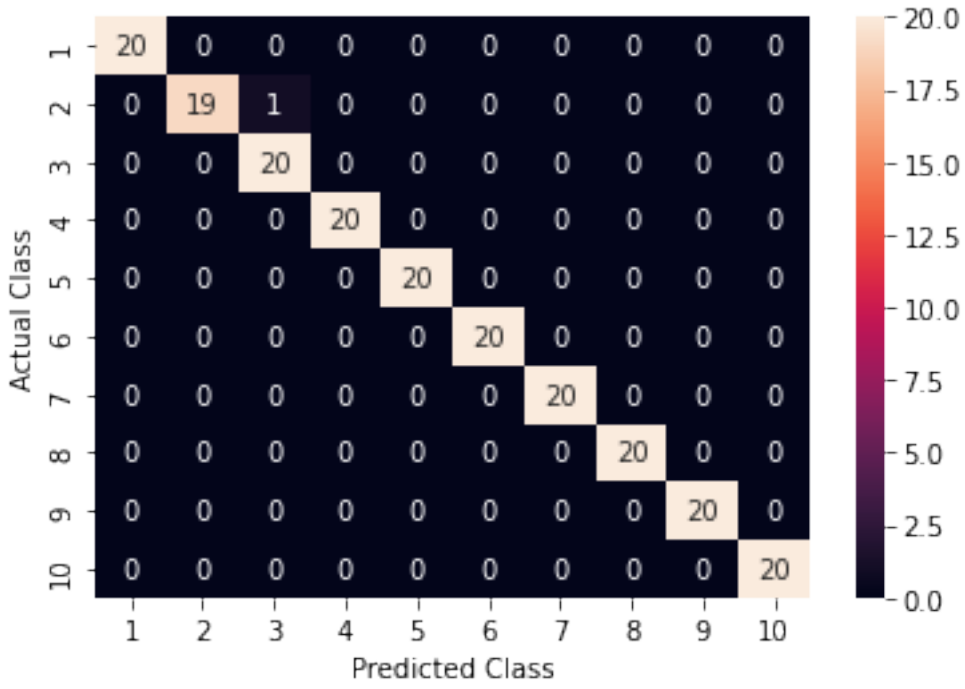
```
    knn = KNeighborsClassifier()  # KNN Initialization
    knn_cv = GridSearchCV(knn, param_grid, scoring='accuracy')

    # fitting the model for grid search
    knn_cv.fit(X, Y)
    knn = knn_cv.best_estimator_  # Best model parameters are stored
    print ('Best accuracy and parameters are: ', knn_cv.best_score_, knn_cv.
    ↪best_params_)

    print('\n Test Results \n')
    y_p=knn_cv.best_estimator_.predict(x)

    x_axis_labels = sorted(np.unique(Y)) # labels for x-axis
    y_axis_labels = sorted(np.unique(Y)) # labels for y-axis
    sns.heatmap(confusion_matrix(y, y_p), annot = True,␣
    ↪xticklabels=x_axis_labels, yticklabels=y_axis_labels)
    plt.xlabel("Predicted Class")
    plt.ylabel("Actual Class")
    print(classification_report(y, y_p))

    return y_p
```

Quantitative Results

[28]:
```
y_pred = KNN(train_hg_desc, y_train, test_hg_desc, y_test)
```

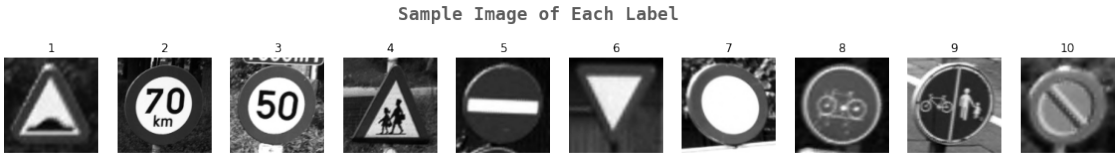
```
Best accuracy and parameters are:  0.99125 {'metric': 'minkowski',
'n_neighbors': 5}

 Test Results

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        20
           2       1.00      0.95      0.97        20
           3       0.95      1.00      0.98        20
           4       1.00      1.00      1.00        20
           5       1.00      1.00      1.00        20
           6       1.00      1.00      1.00        20
           7       1.00      1.00      1.00        20
           8       1.00      1.00      1.00        20
           9       1.00      1.00      1.00        20
          10       1.00      1.00      1.00        20

    accuracy                           0.99       200
   macro avg       1.00      0.99      0.99       200
```

```
weighted avg       1.00      0.99      0.99       200
```



Qualitative Results

[29]:
```
# Correctly Classified Test Images from each class
k=0
fig, ax = plt.subplots(1,10,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True:
        if y_test[j]==i and y_pred[j]==i:
            ax[k].imshow(X_test[j], 'gray')
            ax[k].set_title(y_pred[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```



[37]:
```
# Incorrectly Classified Test Images from each class
fig, ax = plt.subplots(1,1,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True:
        if j==200:
            break
        if y_test[j]==i and y_pred[j]!=i:
            ax.imshow(X_test[j], 'gray')
            ax.set_title(y_pred[j])
            ax.axis('off')
            break
        j+=1
```

3

SIFT Descriptors

October 16, 2022

0.1 Importing Libraries

[128]:
```
import os
import random
import pandas as pd
import numpy as np
import cv2
import time
from tqdm import tqdm
from cv2 import HOGDescriptor

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import LeaveOneOut, cross_validate,␣
↪cross_val_score, StratifiedShuffleSplit

import seaborn as sns
import matplotlib.pyplot as plt

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

0.2 Load and View Images

[129]:
```
folders = os.listdir("CS67_Fall2022_ A1_Dataset")
if ".DS_Store" in folders:
    folders.remove(".DS_Store")

folders = sorted(folders, key=int)

data_type = ["train", "test"] # txt Files in Dataset folders
df_list = [] # List that will contain the dataframe dict

# In each folder there are 2 txt files, train.txt and test.txt
# train.txt files from each folder are taken and image names in these files are␣
 ↪stored as filename in dict
# Same procedure is then followed for test.txt files
for i in data_type:
    for j in folders:
        with open(f"CS67_Fall2022_ A1_Dataset/{j}/{i}.txt") as f:
            for line in f:
                df_dict = {}
                df_dict["folder"] = j
                df_dict["filename"] = line
                df_dict["file_type"] = i    # Train or Test
                df_dict["file_path"] = f"CS67_Fall2022_ A1_Dataset/{j}/{line}".
 ↪strip()
                df_dict["label"] = int(j)   # Integer Label
                df_list.append(df_dict)

df = pd.DataFrame.from_dict(df_list)
df.head(10)
```

[129]:

	folder	filename	file_type	\
0	00001	00468_00002.ppm\n	train	
1	00001	00475_00000.ppm\n	train	
2	00001	00475_00001.ppm\n	train	
3	00001	00475_00002.ppm\n	train	
4	00001	00801_00000.ppm\n	train	
5	00001	00801_00001.ppm\n	train	
6	00001	00801_00002.ppm\n	train	
7	00001	00806_00000.ppm\n	train	
8	00001	00806_00001.ppm\n	train	
9	00001	00806_00002.ppm\n	train	

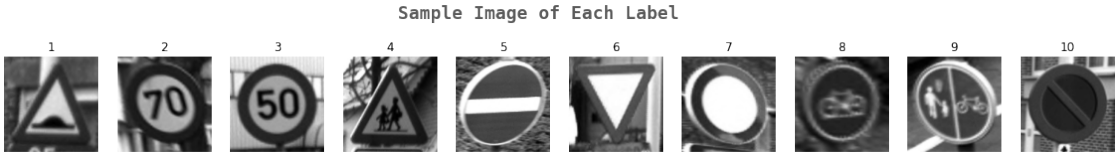
	file_path	label
0	CS67_Fall2022_ A1_Dataset/00001/00468_00002.ppm	1
1	CS67_Fall2022_ A1_Dataset/00001/00475_00000.ppm	1
2	CS67_Fall2022_ A1_Dataset/00001/00475_00001.ppm	1

```
3  CS67_Fall2022_ A1_Dataset/00001/00475_00002.ppm      1
4  CS67_Fall2022_ A1_Dataset/00001/00801_00000.ppm      1
5  CS67_Fall2022_ A1_Dataset/00001/00801_00001.ppm      1
6  CS67_Fall2022_ A1_Dataset/00001/00801_00002.ppm      1
7  CS67_Fall2022_ A1_Dataset/00001/00806_00000.ppm      1
8  CS67_Fall2022_ A1_Dataset/00001/00806_00001.ppm      1
9  CS67_Fall2022_ A1_Dataset/00001/00806_00002.ppm      1
```

[130]:
```python
# Initialize lists to store train and test images
X_train = []
y_train = []
X_test = []
y_test = []

# iterate over the dataframe rows. For each row check type of file that whether␣
 ↪it is train or test.
for _, row in tqdm(df.iterrows(), total=df.shape[0]):
    if row.file_type == "train":
        img = cv2.imread(row.file_path) # Read the file using filepath given in␣
 ↪dataframe.
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to grayscale
        img = cv2.resize(img, (128, 128))
        X_train.append(img) # Append the loaded image in training data list
        y_train.append(row.label)
    else:
        img = cv2.imread(row.file_path) # Read the file using filepath given in␣
 ↪dataframe.
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to grayscale
        img = cv2.resize(img, (128, 128))
        X_test.append(img) # Append the loaded image in test data list
        y_test.append(row.label)

# Convert train and test data to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)

X_test = np.array(X_test)
y_test = np.array(y_test)

print("Training Set Shape: ", X_train.shape)
print("Test Set Shape: ", X_test.shape)
```

```
100%|      | 1000/1000 [00:00<00:00, 3085.15it/s]

Training Set Shape:  (800, 128, 128)
Test Set Shape:  (200, 128, 128)
```

[131]:
```
# Visiualize 1 image from each of the class

k=0
fig, ax = plt.subplots(1,10,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True :
        if y_train[j]==i:
            ax[k].imshow(X_train[j], 'gray')
            ax[k].set_title(y_train[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```



0.3 SIFT Descriptor and Helper Functions

[122]:
```
def getDescriptors(sift, img):
    kp, des = sift.detectAndCompute(img, None)
    return des

def vstackDescriptors(descriptor_list):
    descriptors = np.array(descriptor_list[0])
    for descriptor in descriptor_list[1:]:
        descriptors = np.vstack((descriptors, descriptor))

    return descriptors

def clusterDescriptors(descriptors, no_clusters):
    kmeans = KMeans(n_clusters = no_clusters).fit(descriptors)
    return kmeans

def extractFeatures(kmeans, descriptor_list, image_count, no_clusters):
    im_features = np.array([np.zeros(no_clusters) for i in range(image_count)])
    for i in range(image_count):
        for j in range(len(descriptor_list[i])):
```

```
            feature = descriptor_list[i][j]
            feature = feature.reshape(1, 128)
            idx = kmeans.predict(feature)
            im_features[i][idx] += 1

    return im_features

def normalizeFeatures(scale, features):
    return scale.transform(features)

def plotHistogram(im_features, no_clusters):
    x_scalar = np.arange(no_clusters)
    y_scalar = np.array([abs(np.sum(im_features[:,h], dtype=np.int32)) for h in␣
 ↪range(no_clusters)])

    plt.figure(figsize=(20,10))
    plt.bar(x_scalar, y_scalar)
    plt.xlabel("Visual Word Index")
    plt.ylabel("Frequency")
    plt.title("Complete Vocabulary Generated")
    plt.xticks(x_scalar + 0.4, x_scalar)
    plt.show()

def svcParamSelection(X, y, kernel, nfolds):
    Cs = [0.1, 1, 10, 100]
    gammas = [0.1, 0.11, 0.095, 0.105]
    param_grid = {'C': Cs, 'gamma' : gammas}
    grid_search = GridSearchCV(SVC(kernel=kernel), param_grid, cv=nfolds)
    grid_search.fit(X, y)
    grid_search.best_params_
    return grid_search.best_params_

def findSVM(im_features, train_labels, kernel):
    features = im_features
    if kernel == "precomputed":
      features = np.dot(im_features, im_features.T)

    params = svcParamSelection(features, train_labels, kernel, 5)
    C_param, gamma_param = params.get("C"), params.get("gamma")
    print(C_param, gamma_param)

    svm = SVC(kernel = kernel, C =  C_param, gamma = gamma_param)
    svm.fit(features, train_labels)
    return svm
```

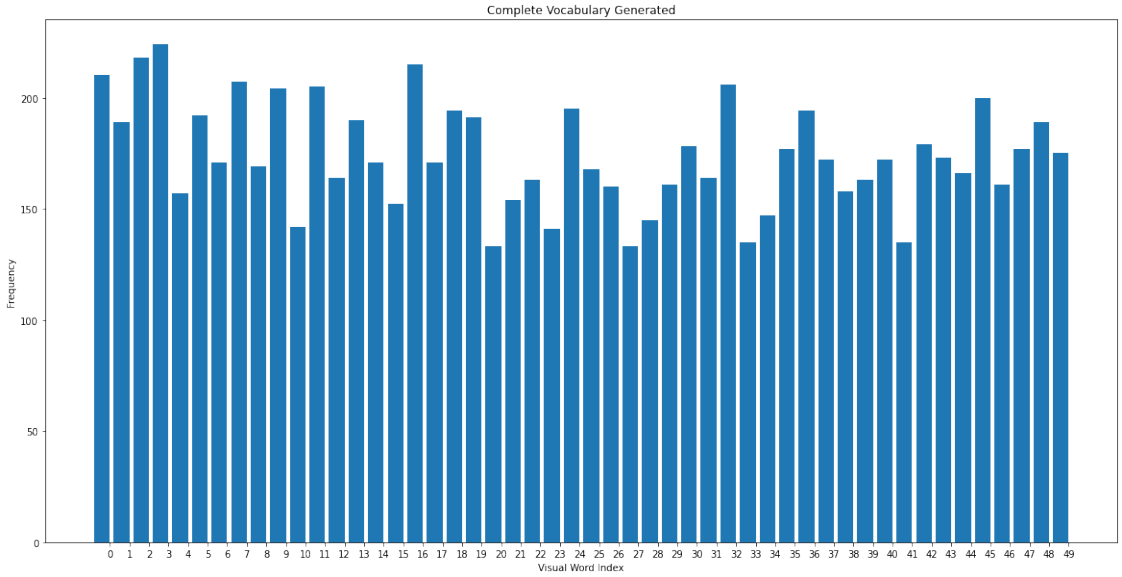
0.4 SVM with SIFT

0.4.1 Training

[123]:
```
def trainModel(trainingData, trainLabels, no_clusters, kernel):
    sift = cv2.SIFT_create()
    descriptor_list = []
    train_labels = trainLabels
    label_count = 10
    image_count = X_train.shape[0]

    for img in trainingData:
        des = getDescriptors(sift, img)
        descriptor_list.append(des)

    descriptors = vstackDescriptors(descriptor_list)
    print("Descriptors vstacked.")

    kmeans = clusterDescriptors(descriptors, no_clusters)
    print("Descriptors clustered.")

    im_features = extractFeatures(kmeans, descriptor_list, image_count,␣
 ↪no_clusters)
    print("Images features extracted.")

    scale = StandardScaler().fit(im_features)
    im_features = scale.transform(im_features)
    print("Train images normalized.")

    plotHistogram(im_features, no_clusters)
    print("Features histogram plotted.")

    svm = findSVM(im_features, train_labels, kernel)
    print("SVM fitted.")
    print("Training completed.")

    return kmeans, scale, svm, im_features
```

[124]:
```
#train_path = "inputs/dataset/train"
#test_path = "inputs/dataset/test"
no_clusters = 50
kernel = "precomputed"

start_time = time.time()
kmeans, scale, svm, im_features = trainModel(X_train, y_train, no_clusters,␣
 ↪kernel)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Descriptors vstacked.
Descriptors clustered.
Images features extracted.
Train images normalized.
```




```
Features histogram plotted.
0.1 0.1
SVM fitted.
Training completed.
--- 37.52240777015686 seconds ---
```

0.4.2 Testing SVM

[125]:
```python
def testModel(testData, kmeans, scale, svm, im_features, no_clusters, kernel):
    count = 0
    true = []
    descriptor_list = []

    sift = cv2.SIFT_create()

    for img in testData:
        des = getDescriptors(sift, img)

        if des is not None:
            count += 1
            descriptor_list.append(des)
```

```
    descriptors = vstackDescriptors(descriptor_list)

    test_features = extractFeatures(kmeans, descriptor_list, count, no_clusters)

    test_features = scale.transform(test_features)

    kernel_test = test_features
    if kernel == "precomputed":
        kernel_test = np.dot(test_features, im_features.T)

    predictions = svm.predict(kernel_test)
    print("Test images classified.")

    return predictions
```

[126]:
```
start_time = time.time()
y_pred = testModel(X_test, kmeans, scale, svm, im_features, no_clusters, kernel)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Test images classified.
--- 4.098567962646484 seconds ---
```

0.5 Results

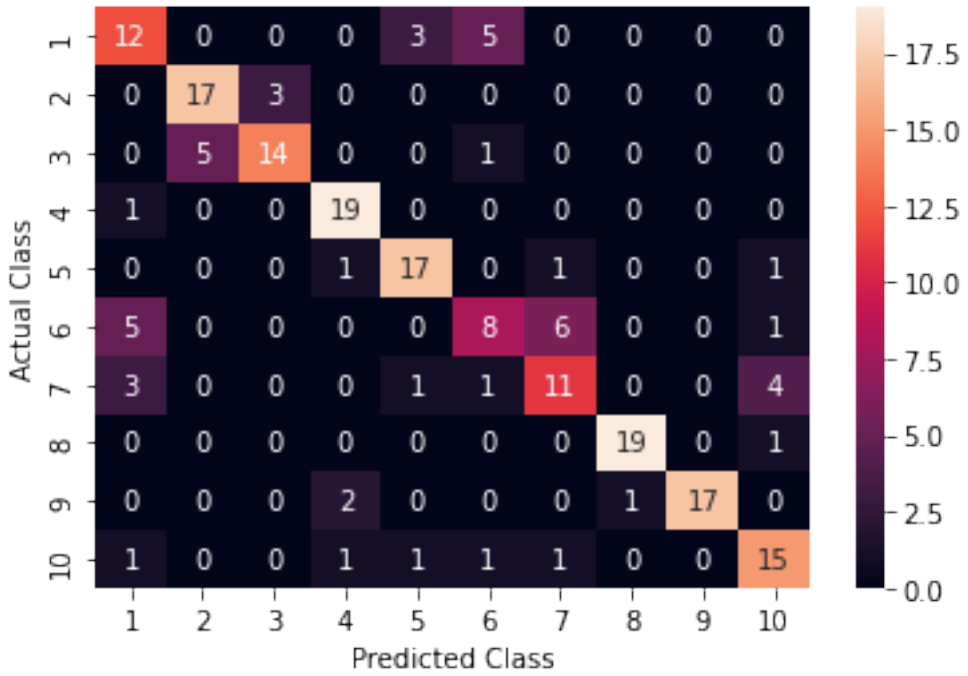
0.5.1 Quantitative Results

[165]:
```
print(classification_report(y_test, y_pred))

x_axis_labels = sorted(np.unique(y_test)) # labels for x-axis
y_axis_labels = sorted(np.unique(y_test)) # labels for y-axis

sns.heatmap(confusion_matrix(y_test, y_pred), annot = True,␣
 ↪xticklabels=x_axis_labels, yticklabels=y_axis_labels)
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
```

	precision	recall	f1-score	support
1	0.55	0.60	0.57	20
2	0.77	0.85	0.81	20
3	0.82	0.70	0.76	20
4	0.83	0.95	0.88	20
5	0.77	0.85	0.81	20
6	0.50	0.40	0.44	20
7	0.58	0.55	0.56	20
8	0.95	0.95	0.95	20

```
           9       1.00      0.85      0.92        20
          10       0.68      0.75      0.71        20

    accuracy                           0.74       200
   macro avg       0.75      0.74      0.74       200
weighted avg       0.75      0.74      0.74       200
```

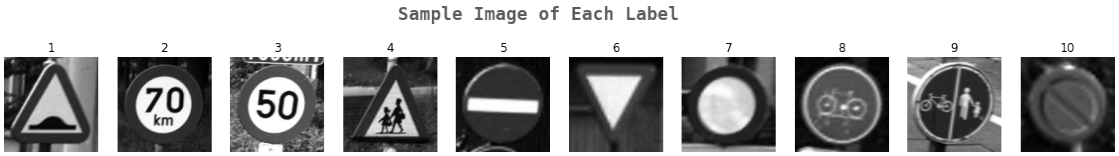
[165]: `Text(33.0, 0.5, 'Actual Class')`



0.5.2 Qualitative Results

[160]:
```
# Correctly Classified Test Images from each class
k=0
fig, ax = plt.subplots(1,10,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True:
        if y_test[j]==i and y_pred[j]==i:
            ax[k].imshow(X_test[j], 'gray')
            ax[k].set_title(y_pred[j])
            ax[k].axis('off')
```

```
            k+=1
            break
        j+=1
```



[163]:
```
# Incorrectly Classified Test Images from each class
k=0
fig, ax = plt.subplots(1,10,figsize=(20,10))     # Grid of images
fig.text(s='Sample Image of Each Label',size=18,fontweight='bold',
             fontname='monospace',color='#313131',y=0.62,x=0.4,alpha=0.8)
for i in df["label"].unique():  # unique labels are used to visiualize images
    j=0
    while True:
        if y_test[j]==i and y_pred[j]!=i:
            ax[k].imshow(X_test[j], 'gray')
            ax[k].set_title(y_pred[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```