Assignment 2 Report

Visual Recognition

Instructors : Prof. Dinesh B J

K. Yashovardhan Reddy
IMT2019097

1 2a

1.1 How SURF is different from SIFT

In SIFT the scale space extrema is calculated by convolving different sized images with constant sized filter. Here a difference of Gaussian is used for convolution. In SURF the varying sized filter is convoluted with the constant sized image. Here a box filter(Laplacian of Gaussian) is used for convolution.The convolution can be done in parallel for different scales in SURF which is not possible in SIFT. Key point detection in SIFT is done using local extrema detection whereas in SURF keypoints are detected using the hessian matrix and non maxima suppression only.
The orientations and gradient magnitudes are sampled around the keypoints by using the scale of key point in SIFT, in SURF a sliding window is used to get the dominant orientation near the key point. The orientation histogram is calculated over $4 * 4$ sample regions near the keypoint for SIFT. In SURF $5 * 5$ sample regions are used.
The resulting descriptor is 128 bits in SIFT and 64 bits long in SURF. SURF promises better results in rotation invariant, blurr and warp transform than SIFT, but SIFT is better than SURF in different image scales.

1.2 Principles of FLANN matching and RANSAC

- FLANN is a collection of algorithms which are optimized for searching nearest neighbours in large datasets with high dimensions. This method projects the high dimensional features to a lower dimensional space and generates the compact binary codes, which can be used to carry out fast image searches. Usage of compact binary codes reduces the computational costs and the time consumed. Flann uses K Trees to represent K dimensional data i.e. randomized kd-tree algorithm is used if the dataset is of low dimensionality and hierarchial k-means tree is used otherwise generally.
- RANSAC(RANdom SAmple Consensus) is a parameter estimation approach designed to cope with significant number of outliers in the input data. This involves selecting a subset of the data and estimate the best line that fits the subset of points. The correctness is estimated by calculating the total number of points that lie on the estimated line in the whole data set. This process is repeated large number of times so that the probability of selecting a subset which does not include an outlier is minimized and we get the parameters required.

1.3 Results for the panorama image stiching:

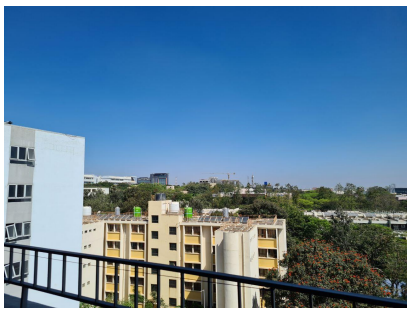

Figure 1: Image 1

Figure 2: Image 2

Figure 3: Image3

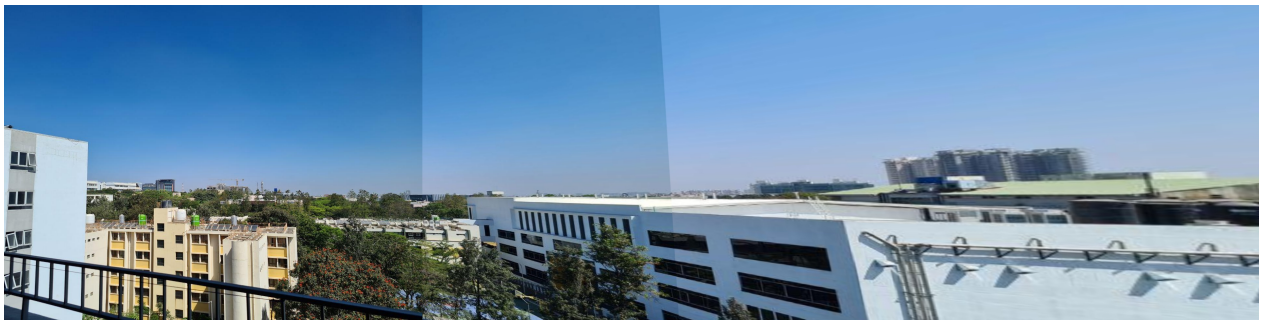

Figure 4: Resulting Image

2 2b

2.1 Code structure, Approach and Observations

Modeled the bag of words approach as a class with certain attributes needed for the implementation. The code on a high level can be divided in to 3 sections; to get the images for the classification, create the class for the prediction and driver code for running it for the given inputs.

2.1.1 Code Structure

Initially all the required libraries are imported and the images are loaded along with their class labels with the help of the following function (fig:5).

```
def getImagesList(path):
    # path = "./Assignment2_BikeHorses"
    trainImages = []
    trainLabels = []
    categorys = []
    label = 0
    for each in os.listdir(path):
        categorys.append(str(each).lower())
        for file in os.listdir(path+"/"+each):
            img = cv2.imread(path +"/"+ each +"/"+file)
            trainImages.append(img)
            trainLabels.append(label)
        label+=1
    return trainImages,trainLabels,categorys
```

Figure 5: Image Loader Function

The bag of visual words approach for classification of images has been implemented as a class whose constructor(fig:6) takes the number of clusters that will be used to classify the descriptors of features of the images. The Class has the following helper functions :

- getDescriptorStack(trainImages,trainLabels,classes): Calculates the descriptors and stacks them verti-

cally such that the number of columns correspond to the number of bins in the descriptors of the image. It creates a SIFT or a SURF object(fig: 6) to get the descriptors based on the parameters passed.

```
class BOVW:
    def __init__(self,number_clusters,):
            self.getClass = []
            self.n_clusters = number_clusters
            self.sift_or_surf_object = None
            self.descriptorStack = None
            self.descriptorList = None
            self.trainLabels = np.array([])
            self.clusterObject = None
            self.clustersList = None
            self.clusterHistogram = None
            self.scaler = None
            self.classifier = None

    def getdescriptorStack(self,TrainImages,trainImageLabels,getClassList,siftOrSurf="sift"):
        if siftOrSurf == "sift":
            self.sift_or_surf_object = cv2.xfeatures2d.SIFT_create()
        else :
            self.sift_or_surf_object = cv2.xfeatures2d.SURF_create()
        self.descriptorList = []
        self.getClass = getClassList
        for i in range(len(TrainImages)):
            self.trainLabels = np.append(self.trainLabels,trainImageLabels[i])
            kp,des = self.sift_or_surf_object.detectAndCompute(TrainImages[i],None)
            self.descriptorList.append(des)
        self.descriptorStack = np.array(self.descriptorList[0])
        for rem in self.descriptorList[1:]:
            self.descriptorStack = np.vstack((self.descriptorStack,rem))

    def getClusters(self):
        # print(self.descriptorList.shape)
        # print(self.descriptorStack.shape)
        self.clusterObject = KMeans(n_clusters=self.n_clusters,n_init=15,max_iter=400)
        self.clustersList = self.clusterObject.fit_predict(self.descriptorStack)

        print("clusterList.shape - ",self.clustersList.shape)
```

(variable) clusterHistogram: NDArray[_SCT@array] | Any | None

Figure 6: Class for Bag of Visual Words and the helper functions

- getClusters():Creates a KMeans object(fig: 6) with required number of clusters and gets the cluster labels for the descriptors in the train data.
- makeHistogram(): The vocabulary is generated using the cluster id of each of the descriptor in each image in this function(fig: 7).
- printHistogram(): A function to visualize the distribution of the descriptors in each of the clusters(fig:7).
- scaleHistogram(): Accomplishes the scaling of the values in the histogram so that the algorithms can infer without any bias from the train data(fig:7).
- fitClassifier(): Creates a classifier object based on the arguements passed, can create a KNN, LogisticRegression or a SVC classifier. The classifier is trained with the train labels and the normalized vocabulary histogram for each image (fig:7).
- predict(image): Used to predict the the class of the image passed as arguement(fig:7).

Creation of test and train data, creation of the bovw class object, calling of the predict function and all other functionalities are handled by the driver code.

```
def makeHistogram(self):
    self.clusterHistogram = np.array([np.zeros(self.n_clusters) for i in range(len(self.descriptorList))])
    total = 0
    for i in range(len(self.descriptorList)):
        l = len(self.descriptorList[i])
        for j in range(l):
            idx = self.clustersList[total+j]
            self.clusterHistogram[i][idx] += 1
        total+=l

def printHistogram(self):
    x = np.arange(self.n_clusters)
    y = np.array([abs(np.sum(self.clusterHistogram[:,h],dtype=np.int32)) for h in range(self.n_clusters)])

    plt.bar(x,y)
    plt.xlabel("cluster index")
    plt.ylabel("Frequency")
    plt.show()

def scaleHistogram(self,):
    print("max in histogram = ",np.amax(self.clusterHistogram))
    print("min in histogram = ",np.amin(self.clusterHistogram))
    self.scaler =  StandardScaler().fit(self.clusterHistogram)
    # self.scaler =  MinMaxScaler().fit(self.clusterHistogram)
    self.clusterHistogram = self.scaler.transform(self.clusterHistogram)
    print("min in histogram = ",np.amin(self.clusterHistogram))
    print("max in histogram = ",np.amax(self.clusterHistogram))



def fitClassifier(self,method = "svc"):
    if method == "svc":
        self.classifier = SVC()
        self.classifier.fit(self.clusterHistogram,self.trainLabels)
    elif method == "lr":
        self.classifier = LogisticRegression().fit(self.clusterHistogram,self.trainLabels)
    elif method == "knn":
        self.classifier = KNeighborsClassifier(n_neighbors=8).fit(self.clusterHistogram,self.trainLabels)
    print("classifier trained")

def predict(self,img,):
    # gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    kp,des = self.sift_or_surf_object.detectAndCompute(img,None)
    freq = np.array([[0 for i in range(self.n_clusters)]])
    cluster_values = self.clusterObject.predict(des)
    for cluster in cluster_values:
        freq[0][cluster]+=1
    freq = self.scaler.transform(freq)
    prediction = self.classifier.predict(freq)
    # print(prediction)
    if type(prediction) == list:
        # print("this belongs to class ",self.getClass[int(prediction[0])])
        return int(prediction[0])
    else:
        return int(prediction)
```

Figure 7: Helper functions in the class

2.1.2 Approach

The images are loaded and split into test and train sets initially. Then a BOVW object is created for each cluster size value in a loop. In the loop the descriptors are calculated and classified into clusters, a vocabulary histogram is generated and scaled. Then the scaled vocabulary histogram is used to learn from the train labels using a classifier object(KNN, SVC or LogisticRegression). The predict function is called for all the images in the test image set and the accuracy of the prediction is calculated.

2.1.3 Observations

The following observations were made when sift object was used with :

1. The accuracy vs cluster size when using a KNN classifier is shown in Fig:10 and the vocabulary for various cluster sizes is also shown below.

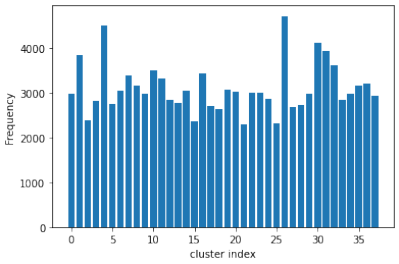

Figure 8: Vocabulary for 38 clusters in KNN

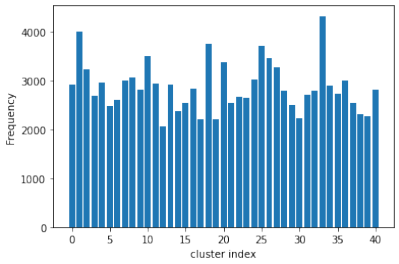

Figure 9: Vocabulary for 40 clusters in KNN

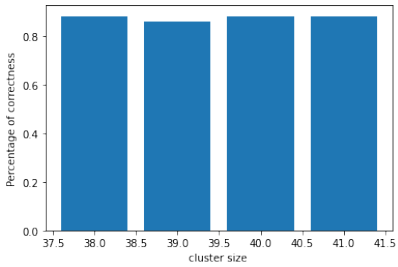

Figure 10: Accuracy vs cluster size for KNN

2. The accuracy vs cluster size when using a SVC classifier is shown in Fig:13 and the vocabulary for various cluster sizes is also shown below.

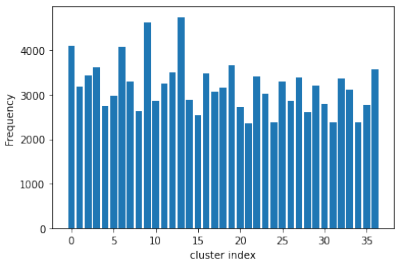

Figure 11: Vocabulary for 36 clusters in SVC

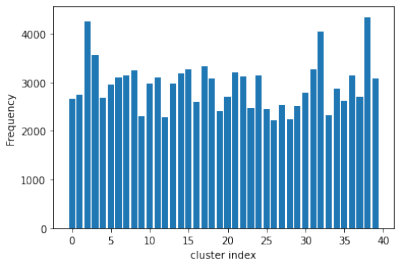

Figure 12: Vocabulary for 40 clusters in SVC

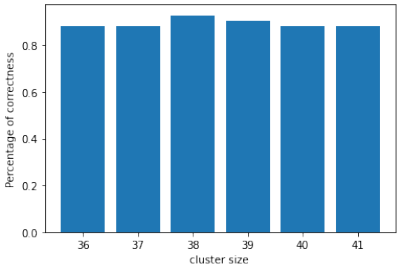

Figure 13: Accuracy vs cluster size for SVC

3. The accuracy vs cluster size when using a Logistic regression classifier is shown in Fig:16 and the vocabulary for various cluster sizes is also shown below.

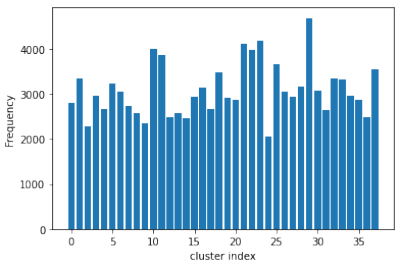

Figure 14: Vocabulary for 36 clusters in Logistic Regression

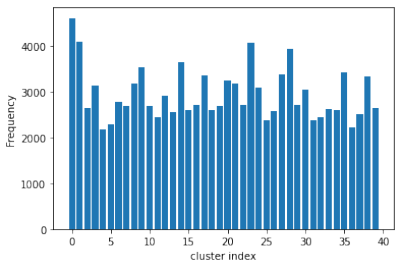

Figure 15: Vocabulary for 40 clusters in Logistic Regression

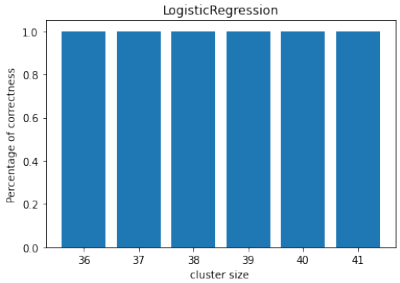

Figure 16: Accuracy vs cluster size for Logistic Regression

An accuracy of 100% was obtained using a Logistic regression classifier and 88% accuracy was obtained while using KNN and SVC classifiers as well.

3 CIFAR 10

An approach similar to the above one was used to predict the labels of the CIFAR 10 dataset. An accuracy of 0.137 was obtained.