CS 783 VISUAL RECOGNITION

Assignment 1

Multiple Instance Recognition

Kritik Soman-18104052
Darshan Ramesh Shet - 16104024

The problem statement for this assignment requires us to implement object detection and recognition of multiple objects under multiple instance case. This can be called as the multi-class multi-instance problem. Object detection can be achieved using Selective Search, a region proposal algorithm and object/instance recognition is done by finding cosine similarity of color histogram feature with the training image database. The detailed explanation of the all the steps that we followed is listed as under.

Part 1: Generation of Color Histogram Feature Database

To only capture the features corresponding to the object in each of the train images, we annotated the bounding boxes for all the images. For each of bounding box portion of the images in train database, we used extract_features() function to return color histogram descriptors by binning across 3 RGB color channel and it is then flattened to the one-dimensional column vector. The function batch_extractor() stores the entire features vector in pickle object.

```
def  extract_features( image_path ):
        descriptors   = []
        for  imagePath  in  imagePathList:
            image = cv2.imread(imagePath, cv2.COLOR_RGB2BGR)
            hist = cv2.calcHist(images=[image], channels=[0,1,2], mask=None,
                            histSize=[16, 16, 16], ranges=[0, 256] * 3)
            dsc = hist.flatten().astype(np.uint8)
        return descriptor
```

```
def  batch_extractor( pickled_db_path = "features.pck" ):
        for  dirindex  in  dirPathList :
              fileLoc =  os.getcwd() + trainPath + '/' + dirindex
              imagePathList = [f for f in os.listdir(fileLoc)]
              for imagePath in imagePathList:
                 f = os.path.join(fileLoc, imagePath)
                 name = dirindex + '_' +imagePath
                 result[name] = extract_features(f)

         with open(pickled_db_path, 'w') as fp:
                pickle.dump(result, fp)
```

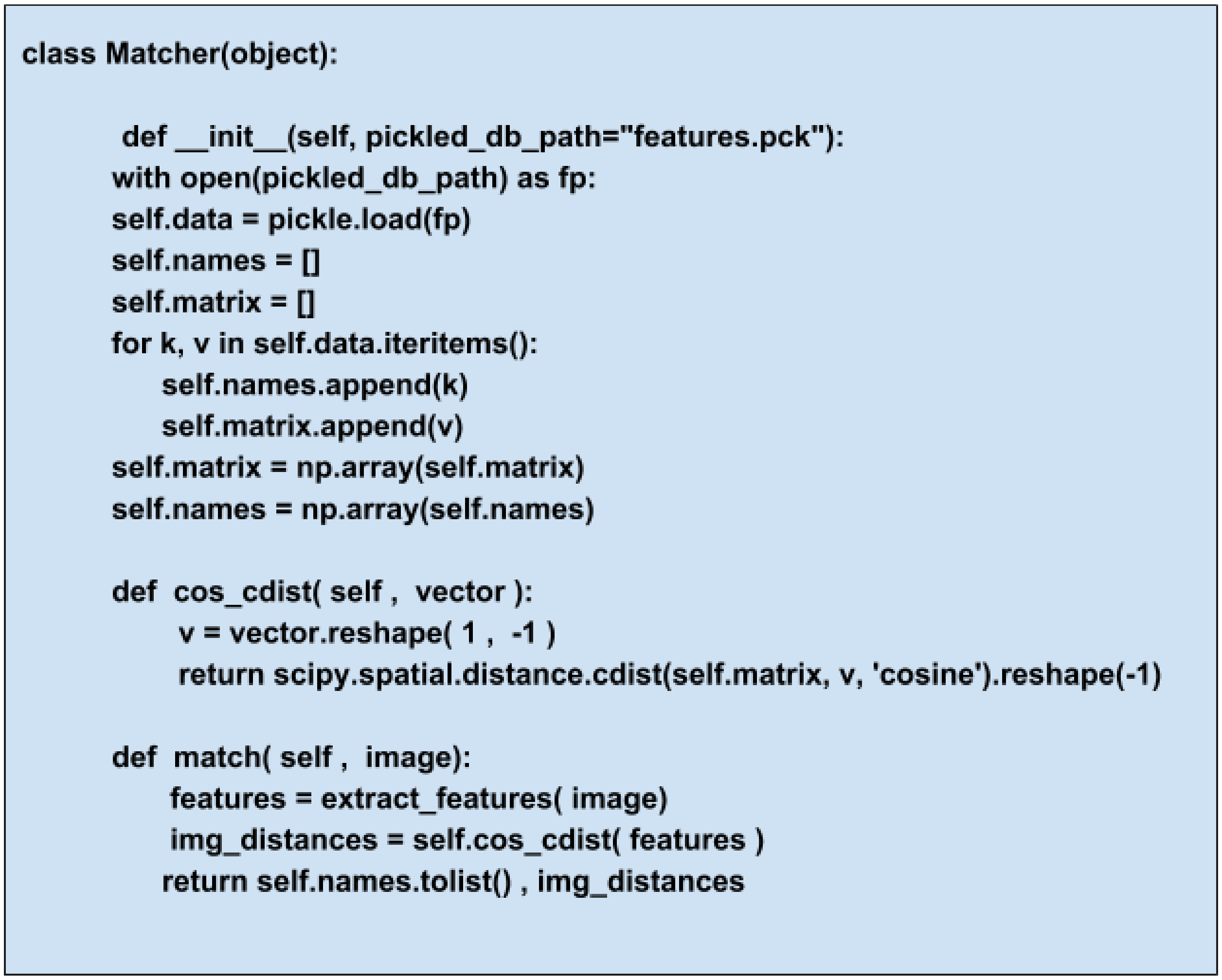
Part 2: Object Detection(Object Localization)

Object detection is done using selective search algorithm [1] which takes an image as the input and output bounding boxes corresponding to all patches in an image that is most likely to be objects. These region proposals can be noisy, overlapping and may not contain the object perfectly. It groups similar regions based on color, texture, size and shape compatibility.

To implement above method we used selective search code for object recognition" by J.R.R. Uijlings et al.

Part 3. Instance Recognition

Object or Instance recognition is done by finding cosine similarity between the test image and train image database. Color histogram feature is computed for bounding box of test image returned by object detection algorithm and given to cos_cdist to compute similarity.



```
class Matcher(object):

         def __init__(self, pickled_db_path="features.pck"):
        with open(pickled_db_path) as fp:
        self.data = pickle.load(fp)
        self.names = []
        self.matrix = []
        for k, v in self.data.iteritems():
            self.names.append(k)
            self.matrix.append(v)
        self.matrix = np.array(self.matrix)
        self.names = np.array(self.names)

        def  cos_cdist( self ,  vector ):
             v = vector.reshape( 1 ,  -1 )
             return scipy.spatial.distance.cdist(self.matrix, v, 'cosine').reshape(-1)

        def  match( self ,  image):
             features = extract_features( image)
             img_distances = self.cos_cdist( features )
            return self.names.tolist() , img_distances
```

4. Ranking

In this step, ranking function computes the color histogram descriptors for all the bounding boxes returned by selective search algorithm. Cosine similarity is used to rank all the training images for every bounding box. The best rank (minimum value of cosine similarity) of every training image is then picked and stored in a list. Finally, this list is sorted and that is used to rank all the training images. By using the minimum value of cosine similarity for a training image, we are also able to eliminate the error due to redundant (partly overlapping) bounding boxes returned by selective search.

```
# Find ranking of all images for every  bounding box
hist = cv2.calcHist([img], [0, 1, 2], None, [16, 16, 16], [0, 256, 0, 256,0, 256])
hist = hist.flatten().astype(np.uint8)
ma = Matcher('features.pck')

fnames=[]
N = len(regions)  #no. of bounding box
finalScore = np.zeros((3456,N))
For  i  in  range(len(regions)):
      b = regions[i]
      subimg = img[b[1]:b[3],b[0]:b[2],:]
      names, finalScore[:,i] = ma.match(subimg)
      fnames=names
rank = np.nanmin(finalScore,axis=1)
index = np.argsort(rank)
```

MD5 hash value for our training model

MD5 (features.pck) = **46ebcbf9e3f7c458e02ceadb8be7df42**

4. Other approaches that we tried

We initially started to address this problem by looking at local scale-invariant features such as sift, orb etc. The resolution of the part of the train image that contained the object was so poor so it could hardly detect strong distinctive feature (For example: the images containing “expo_marker_red” has only 3-4 SIFT keypoints, which are very less for matching). We tried implementing an image search engine which we build on sift feature, but it did not produce good results.

Next, we explored the visual vocabulary recognition tree which again builds on extracting sift features. This also failed because of lack of distinctive features and at some instance of objects had plane texture in which sift features were not created.

Then, we tried to implement pyramid match kernel (which was discussed in class) for matching SIFT descriptors, but we were unable to implement this method because it required 128 dimensional binning (The max dimension of histogram binning that can be done via numpy.histogramdd is 32). Implementing our own binning code would require memory management due to the large dimension of binning. Therefore, we dropped this idea.

After this, we tried to pass the training images through a pretrained network (using the imagenet model weights from keras.applications.VGG16) and use the output of the last fully connected layer as the descriptor of the object. We tried to find the cosine similarity of this with the same descriptor from the test images, but this failed because this descriptor that we generated was not rotation and scale invariant.

As all the above trails failed, we tried deep learning framework to solve this problem. So we attempted to train YOLOV2 model for object recognition. We spent a lot of time to do setup and install the prerequisites for the above model on a local machine. Since that machine was slow for training, we uploaded the data on google drive and tried to train the model on Google Colab. Finally, we ran the model but the weights it generated didn't recognize our objects and we were unable to troubleshoot it because of lack of time. We could not preprocess training dataset by adding a different background to objects, scale and rotate images.

5. Results on sample test

The results on the sample test dataset are present in the res folder in the zip file. We observed that for the hard test images, our ranking based on cosine similarity of color histogram descriptor was failing many times. For the simple test images, our method worked well both in single and multiple instance case.

References:

[1] Uijlings, Jasper RR, et al. "Selective search for object recognition." *International journal of computer vision* 104.2 (2013): 154-171.