Computer Vision
Scene Recognition

Introduction

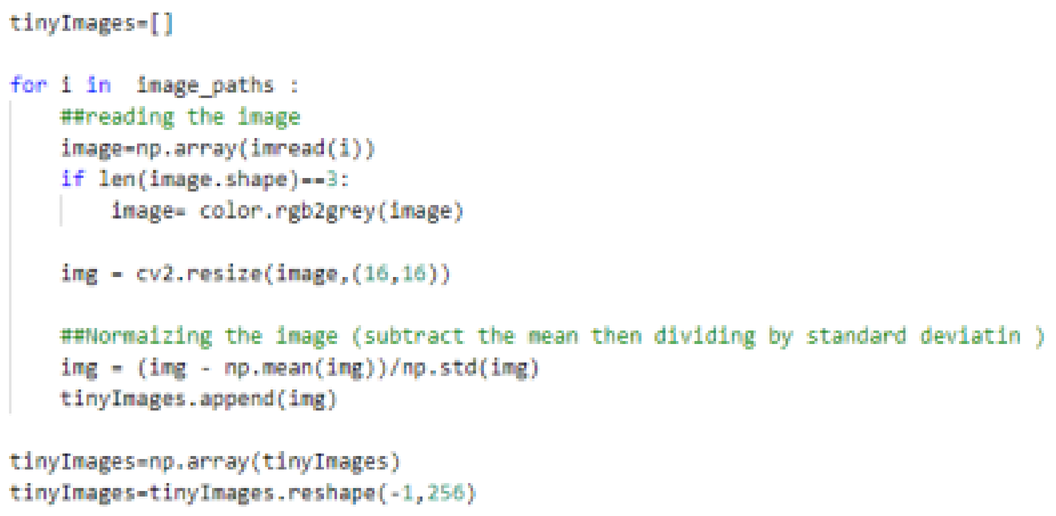
In this project , we are building a scene recognition model.Scene recognition is one of the classical computer vision problems in which the model tries to classify images of different scenes.
Using three different approaches to solve this problem which are
: 1. Tiny image with KNN classifier.
2. Bag of words with KNN classifier.
3. Bag of words with one vs.all linear SVM.

Functions Implemented

1. Tiny images Implementation

This function resizes the images to 16*16 resolution through this steps :

- Iterate through all the images
- Check if it is not grayscale ,turn into gray scale
- Resize to 16*16
- Normalize by subtraction the mean and dividing by std
- Return array of images with dim (n*256) where n is the number of images ((every tiny image in a row))

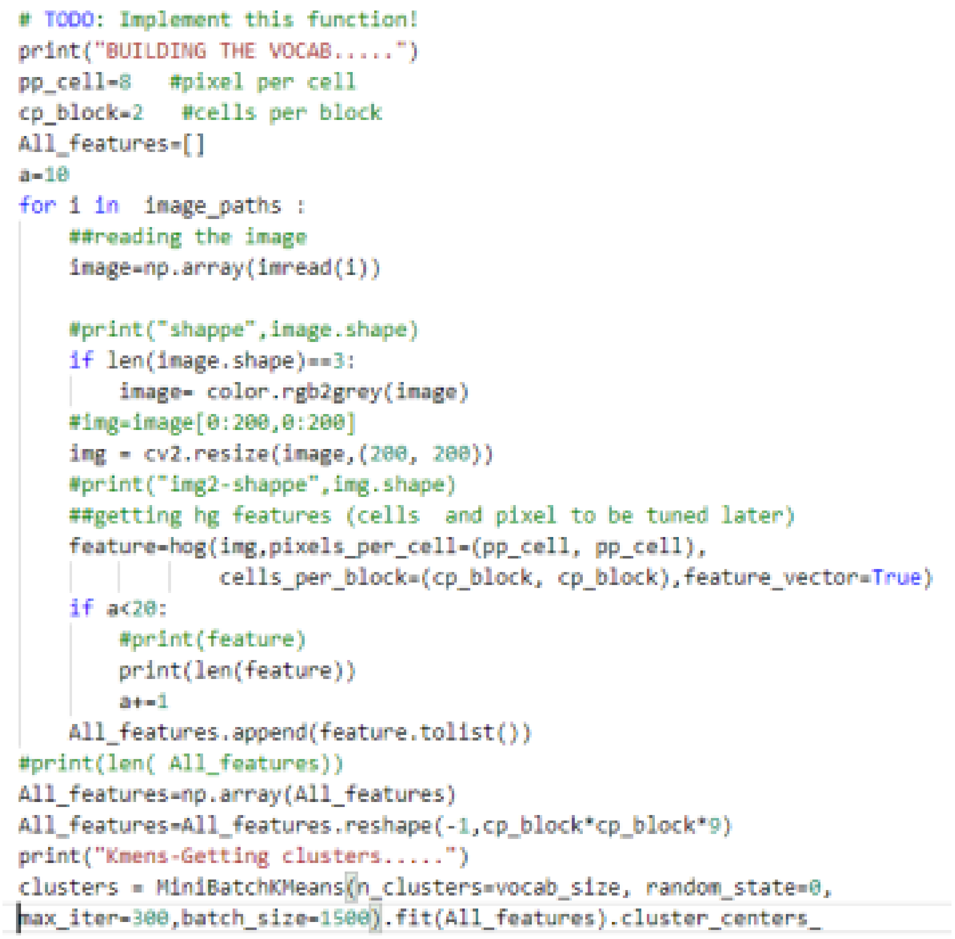


```
tinyImages=[]

for i in  image_paths :
    ##reading the image
    image=np.array(imread(i))
    if len(image.shape)==3:
        image= color.rgb2grey(image)

    img = cv2.resize(image,(16,16))

    ##Normaizing the image (subtract the mean then dividing by standard deviatin )
    img = (img - np.mean(img))/np.std(img)
    tinyImages.append(img)

tinyImages=np.array(tinyImages)
tinyImages=tinyImages.reshape(-1,256)
```

2. Build Vocabulary function

To do better than tiny images ,we now go to Bag of words approach but to do so ,we need to build a vocabulary of visual words first ,the ideas is as follow : ● First ,get the hog feature descriptor for each image so its equivalent to extract the local features from each image

- The above step will be done using skimage hog function
- This will return very massive sized feature vector
- So we cluster them we Kmeans Returning the vocabulary with cluster centers only
- We use Vocab we now have



```
# TODO: Implement this function!
print("BUILDING THE VOCAB.....")
pp_cell=8   #pixel per cell
cp_block=2   #cells per block
All_features=[]
a=10
for i in  image_paths :
    ##reading the image
    image=np.array(imread(i))

    #print("shappe",image.shape)
    if len(image.shape)==3:
        image= color.rgb2grey(image)
    #img=image[0:200,0:200]
    img = cv2.resize(image,(200, 200))
    #print("img2-shappe",img.shape)
    ##getting hg features (cells  and pixel to be tuned later)
    feature=hog(img,pixels_per_cell=(pp_cell, pp_cell),
                cells_per_block=(cp_block, cp_block),feature_vector=True)
    if a<20:
        #print(feature)
        print(len(feature))
        a+=1
    All_features.append(feature.tolist())
#print(len( All_features))
All_features=np.array(All_features)
All_features=All_features.reshape(-1,cp_block*cp_block*9)
print("Kmens-Getting clusters.....")
clusters = MiniBatchKMeans(n_clusters=vocab_size, random_state=0,
max_iter=300,batch_size=1500).fit(All_features).cluster_centers_
```

3. Best parameters tuning for hog function

Tunning with a lot of different values,we reached the following :

- Pixel per cell =8 ,cells per block =2
- K Means with following parameters :

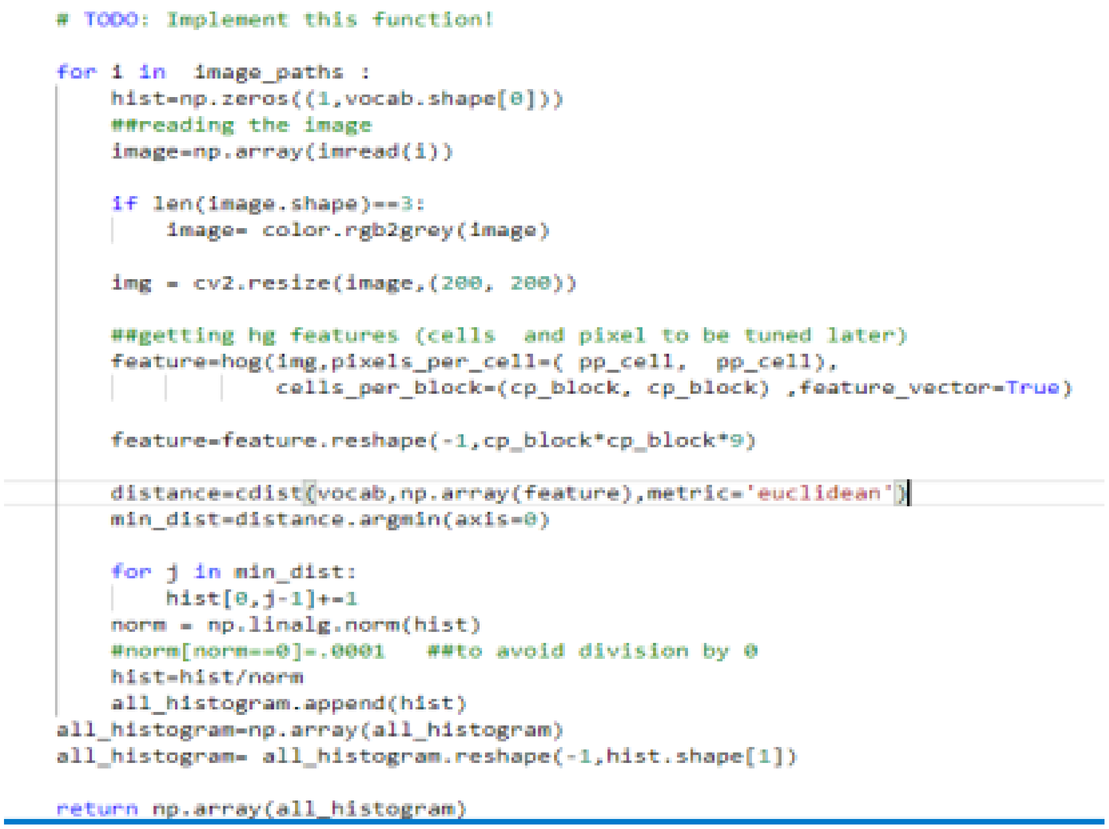
```
MiniBatchKMeans(n_clusters=vocab_size, random_state=0,
    max_iter=300,batch_size=1500).fit(All_features).cluster_centers_
```

4. Bag of words function

After building our Vocabulary of visual words (local features of all the images),we move now to build the bag of descriptors.

For each image ,we will count how many descriptor (feature) falls with in certain cluster by using the euclidean distance to get the closest cluster to that feature and then create histogram for each of them

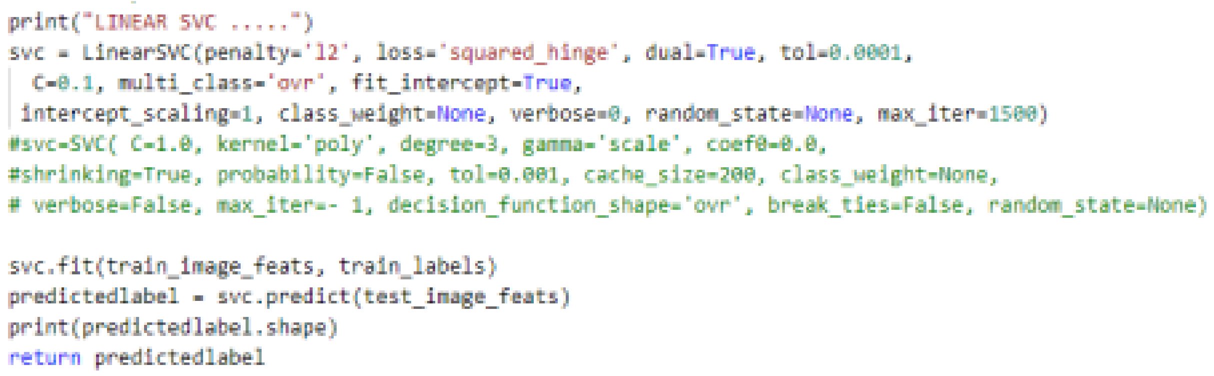
- For each image,we extracted the hog descriptors (image feature vector)
- Getting the closest vocab word for each image
- Building histogram representing the occurrence of each feature for each image corresponding to closest Vocab word



```
# TODO: Implement this function!

for i in  image_paths :
    hist=np.zeros((1,vocab.shape[0]))
    ##reading the image
    image=np.array(imread(i))

    if len(image.shape)==3:
        image= color.rgb2grey(image)

    img = cv2.resize(image,(200, 200))

    ##getting hg features (cells  and pixel to be tuned later)
    feature=hog(img,pixels_per_cell=( pp_cell,  pp_cell),
                cells_per_block=(cp_block, cp_block) ,feature_vector=True)

    feature=feature.reshape(-1,cp_block*cp_block*9)

    distance=cdist(vocab,np.array(feature),metric='euclidean')
    min_dist=distance.argmin(axis=0)

    for j in min_dist:
        hist[0,j-1]+=1
    norm = np.linalg.norm(hist)
    #norm[norm==0]=.0001   ##to avoid division by 0
    hist=hist/norm
    all_histogram.append(hist)
all_histogram=np.array(all_histogram)
all_histogram= all_histogram.reshape(-1,hist.shape[1])

return np.array(all_histogram)
```

5. SVM_classify function Implementation

- This function is used to predict a category for every test image by training 15 many versus one linear SVM classifiers on the training data, then using those learned classifiers on the testing data.
- This function uses LinearSVC from sklearn.svm, and then we use the arguments to call it as (penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=,1 multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1500)
- Then we fit the SVC model between(train_image_feats, train_labels), after multiple tuning to get the best C parameter and tol parameter, we used the model to predict test_image_feats.

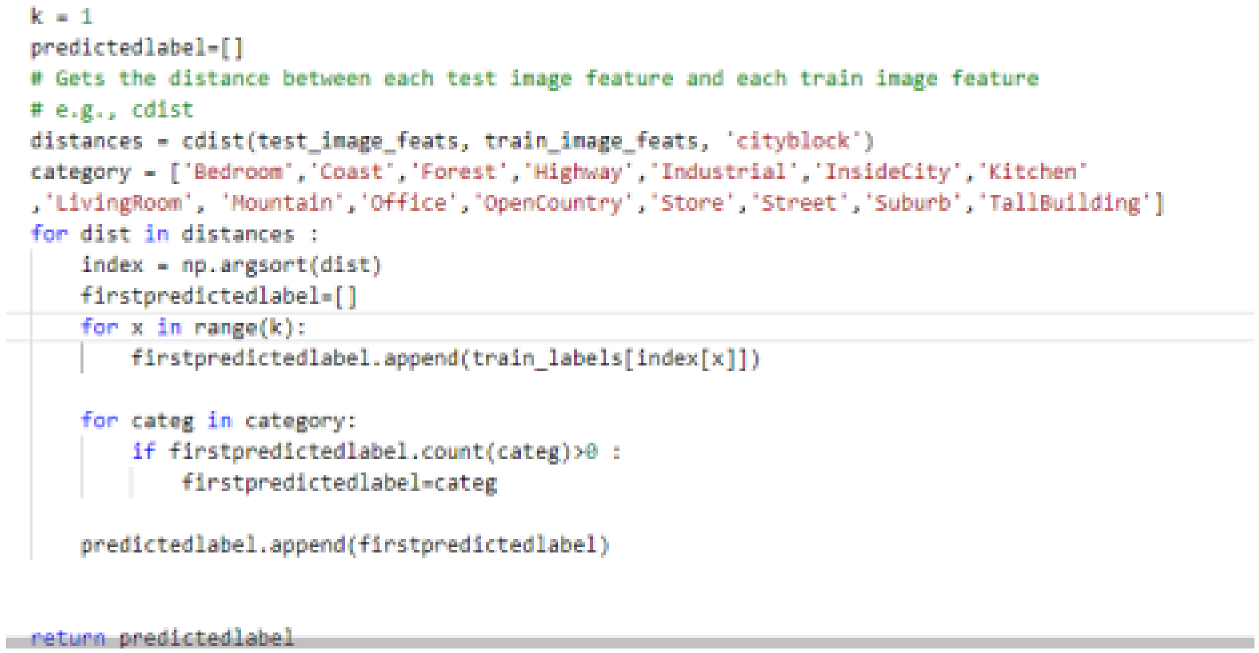


```
print("LINEAR SVC .....")
svc = LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001,
  C=0.1, multi_class='ovr', fit_intercept=True,
 intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1500)
#svc=SVC( C=1.0, kernel='poly', degree=3, gamma='scale', coef0=0.0,
#shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
# verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None)

svc.fit(train_image_feats, train_labels)
predictedlabel = svc.predict(test_image_feats)
print(predictedlabel.shape)
return predictedlabel
```

6. nearest_neighbor_classify function Implementation

This function will predict the category for every test image by finding the training image with most similar features, so we build the function as follow ● First we initialize the categories array and put the list of data names in it such as ‘Coast’, ‘Bedroom’.etc.

- We then calculate the euclidean distance between the test_image_feats and train_image_feats.
- We then loop in the distances array and we argsort the distance to find the index.
    1. Then we loop in the range of k by variable x and then append train_labels[index[x]] in the first predicted label
    2. Then we loop in the category, and if we found first predicted label.count(categ)> 0
- Then we append that category in the predicted label



```
k = 1
predictedlabel=[]
# Gets the distance between each test image feature and each train image feature
# e.g., cdist
distances = cdist(test_image_feats, train_image_feats, 'cityblock')
category = ['Bedroom','Coast','Forest','Highway','Industrial','InsideCity','Kitchen'
,'LivingRoom', 'Mountain','Office','OpenCountry','Store','Street','Suburb','TallBuilding']
for dist in distances :
    index = np.argsort(dist)
    firstpredictedlabel=[]
    for x in range(k):
        firstpredictedlabel.append(train_labels[index[x]])

    for categ in category:
        if firstpredictedlabel.count(categ)>0 :
            firstpredictedlabel=categ

    predictedlabel.append(firstpredictedlabel)


return predictedlabel
```

Results

- After tuning parameters that's what we got :
    1) Pixel per cell =8 ,cells per block =2
    2) K Means with following parameters :
       MiniBatchKMeans(n_clusters=vocab_size, random_state=0, max_iter=300,batch_size=1500).fit(All_features).cluster_centers_
    3) Linear SVC with c=1.0 (a lot of regularization values between .1 :5000)
    4) K=1 is the best one worked for us after trying k=1,2,3
    5) Vocab size =200 best one out of 8 different sizes as below

Extra credit:

Vocab size	KNN accuracy+bag of words SVM accuracy+bag of words
10	38.267% 47.533%
20	45.967% 55.460%
50	52.200% 62.200%

100	53.233 64.533%
200	53.733% 66.867%
400	45.233% 67.767%
1000	36.400% 68.533%
2000	22.667% 68.133%

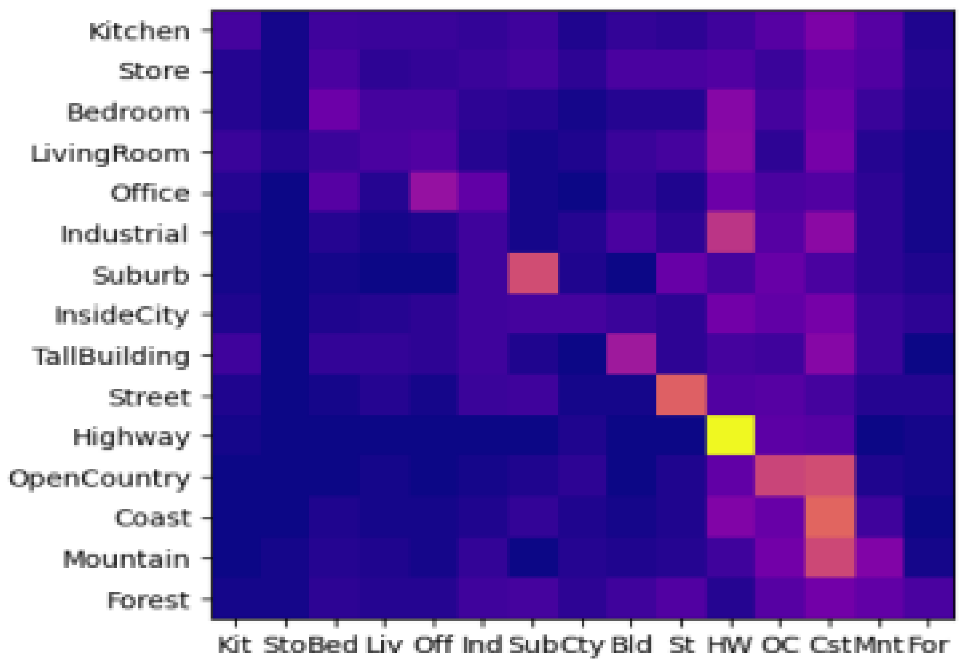
As obvious from the above table ,Increasing the vocab size increases the accuracy to certain point then the accuracy decreases as follows :

- For knn ,the accuracy increases and becomes maxima with Vocab size =200 then decreases heavily.
- For SVM , accuracy increases with increasing Vocab size but decreases slightly with size =2000 which suggests it will decrease more with bigger size.

So ,depending on the above results we chose Vocab size =200 as optimal size which gives highest accuracy for both LinearSVC and KNN Together

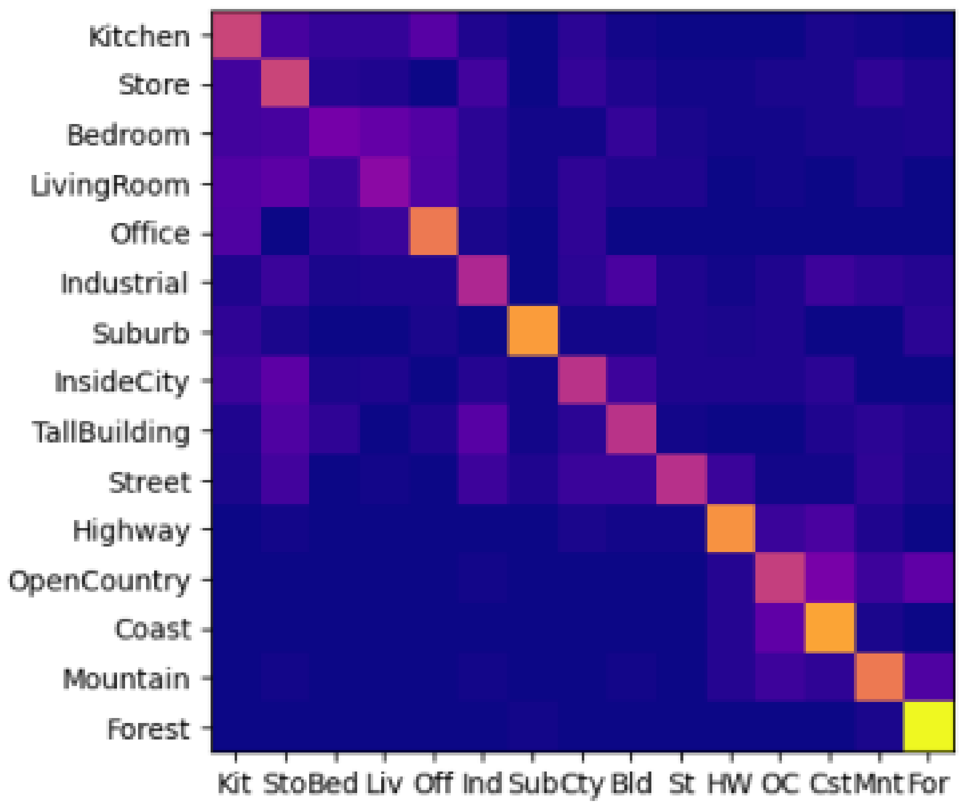
Performance

1. **Tiny images with knn**
    - Accuracy :23.867%
    - Comment : that is no surprising as we lost most of the information when resizing each image to small size



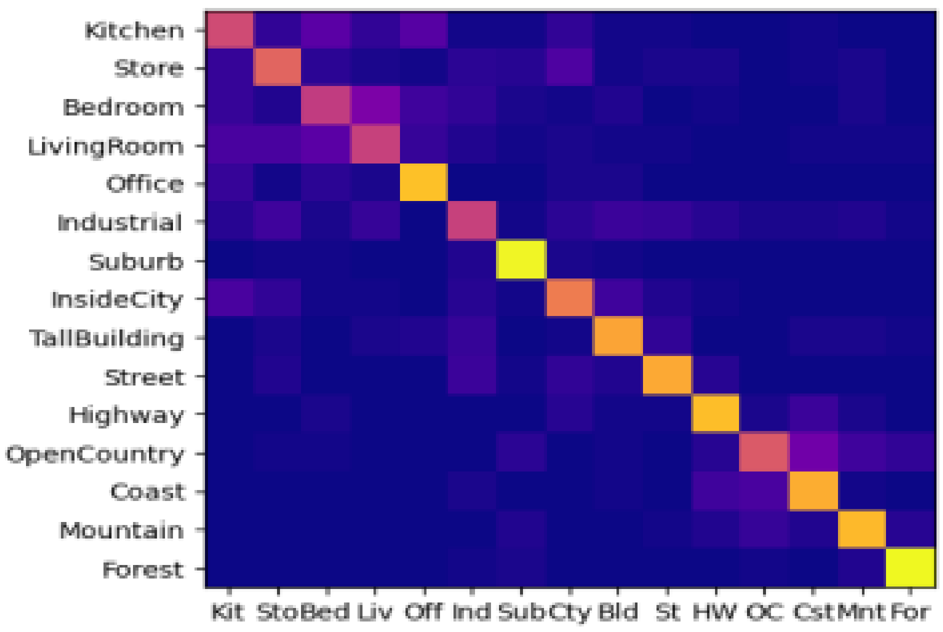
2.Bag of words with 1nn

- Accuracy : 53.733%
- Comment : Much better than tiny image which not surprising as with bag of words we now have more sophisticated and representative representation



3.Bag of words with Multiclass LinearSVM

- Accuracy :66.867% ~67%
- Comment : Linear svm gives best accuracy between them all as stated with the reason behind this in the project file description


Kitchen
Store
Bedroom
LivingRoom
Office
Industrial
Suburb
InsideCity
TallBuilding
Street
Highway
OpenCountry
Coast
Mountain
Forest
Kit
Sto
Bed
Liv
Off
Ind
Sub
Cty
Bld
St
HW
OC
Cst
Mnt
For