Scene Recognition using Bag of Words and SVM

The first part was to load the images successfully. We loaded the images in a single dictionary as it gives the flexibility to access images category wise. The line [**for root, dirs, files in os.walk(path, topdown=True)**] gave us the flexibility to access each folder one-by-one. There was also an option to access folders in an order. We access the folders using the top-down order. This line of code gives us the access to all the images, in each folder, in an array and then applying another nested loop allowed us to access each image separately. If condition was used to ensure only a specific number of images are stored in the dictionary. There are two identical functions namely **load_images_training** and **load_images_test**. The first one is used to load training images and the second function is used to load test images. Both functions return dictionary containing images according to their category and another array called **labels_nd**, where nd represent that it is an nd_array. The bonus part of using spatial pyramids was also completed. Upon looking up the spatial pyramids, it was clear that it is done to extract features which might not be clearly visible otherwise in the image. We upsides the images by the ratio of 2 twice to generate three-levels pyramid. Then the features are extracted from each level of the pyramid.

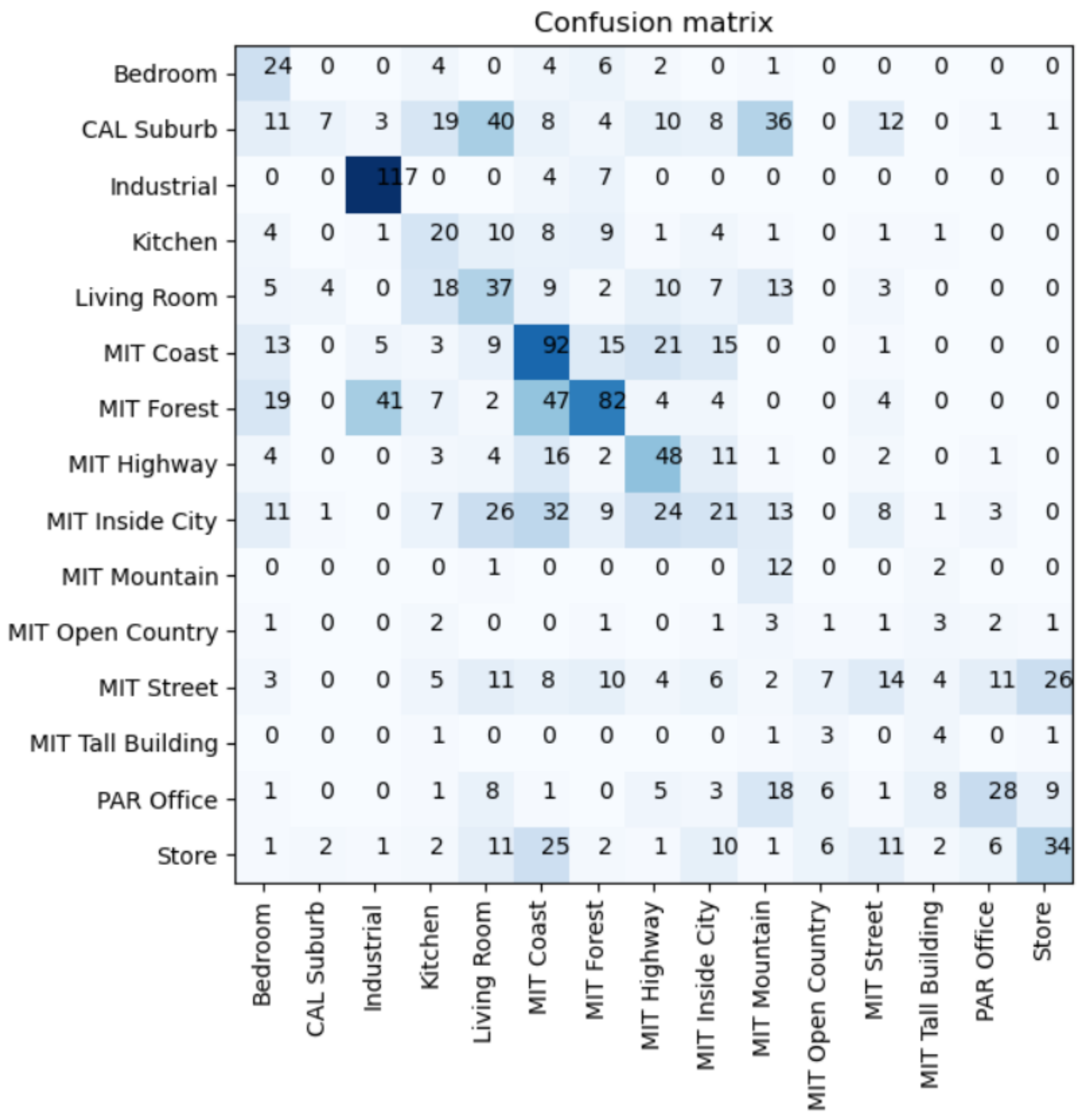
Once the training images are loaded, they are passed to the **features** function. This function generates the HOG descriptors for each image. We chose the HOG descriptors from the start because after reading a few articles in the internet it was evident that it worked the best for object detection applications. Cells per block in the HOG were set to 2x2 as it gave the best results. The HOG code line returns two things. Firstly, the HOG descriptors in a tuple and secondly HOG image that can be seen using cv2.imread command. **Transform_sqrt=True** as it further takes the square root of the descriptors and in turn gives the better descriptors. After generating descriptors, they are converted into the list to select 1000 descriptors randomly from each image. Then those randomly selected descriptors are converted to the 1-D array and float32 as **KMeans** then required to change values to the float32. Descriptors function returns two outputs namely descriptors in an array without any category labels and **features_dict** that contains the descriptors of all images stored according to their category. As it can be noticed to store all the descriptors, in a way so that each descriptor is in one single position of an array, the **.extend** command has been used instead of **.append** because **.append** simply stores all the elements in a single position of the array. Now comes the **clustering** function to the use. We used K-Means clustering function of the **sklearn** library as it ran the fastest, generated K=50 clusters for better accuracy and set 10 iterations as well. We pass the uncategorized array of descriptors to the clustering and then function returns the clusters centres.

Now come the part that takes most of the run time that is making the histograms. The **feature_dict** generated earlier in **features** functions and the centroids from the **clustering** function are passed to the **histograms** function. This function calculates the Euclidean distance of each descriptor from each centroid then assigns the centroid which is nearest to that descriptor. For calculating Euclidean distance, there is another function called inside the **histograms** function namely **indexing**. This function then calculates the frequency of each frequency by adding +1 to that feature frequency. Now the **hist_dict** is used to calculate store all the descriptors in **hist_list** and labels in **labels** array. It is done for the ease of use later in

the code as there were some issues using dictionary directly. After histograms. We then used **SVM** for supervised learning by passing the **hist_list** and **hist_labels**.
After spending some time looking for the easiest and simultaneously accurate way to do svm, we figured out to use svm using **rbf** kernel also known as Radial Basis Function. After doing some research on the internet it turned out that **rbf** is also an appropriate kernel among other for non-linear SVC. There is an extra parameter in the **svm.SVC** line code that is **cache_size=500**. It simply allows us to assign specific amount of memory to the svm. By default, the memory is 200 mb however we set it to 500 mb to speed up the process.

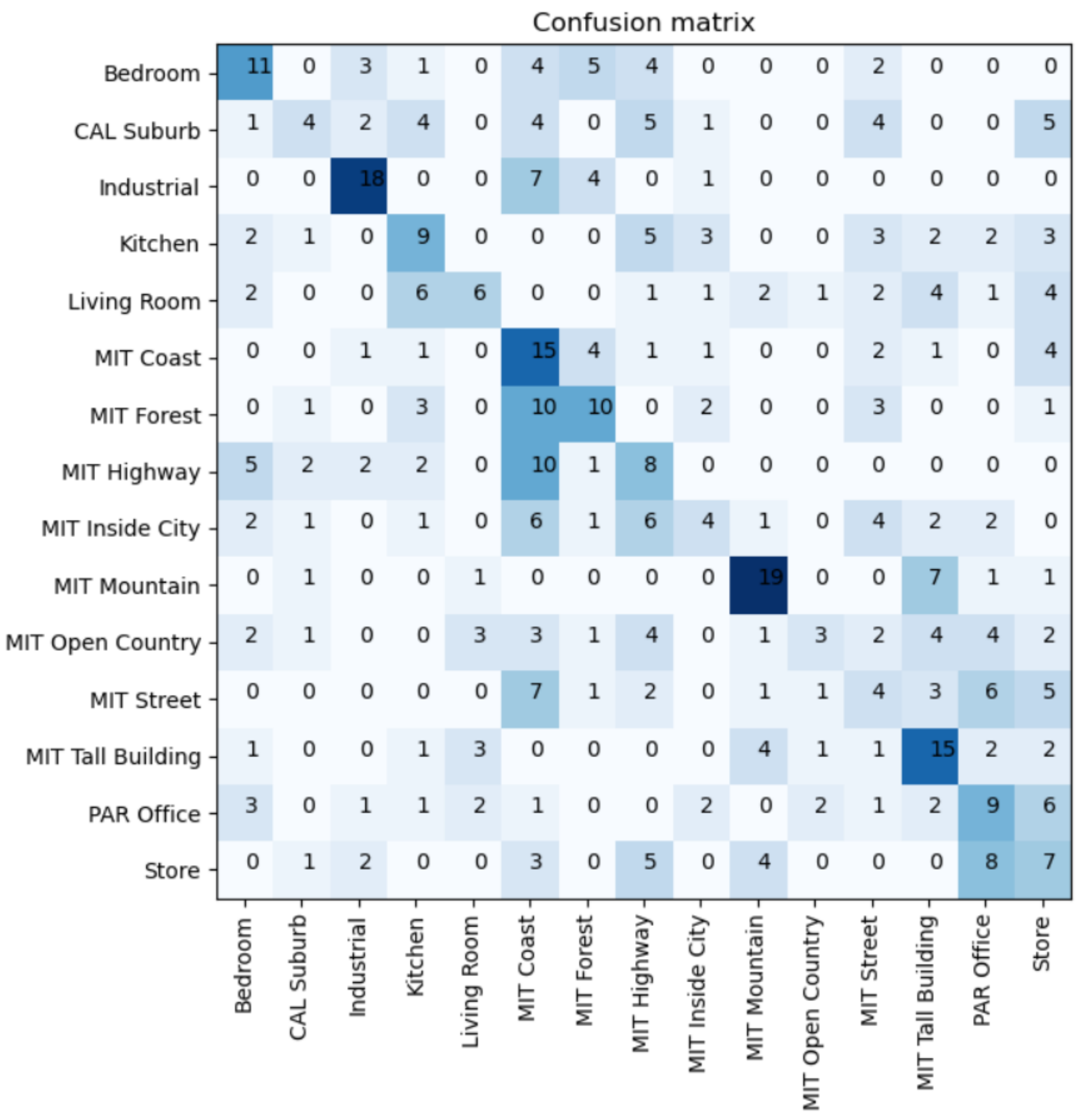
Then starts the testing phase of the algorithm. Just as before test images are loaded then then their descriptors are extracted. Then these descriptors are passed to the **histograms** function but with the centroids of the training data as to compare the test images descriptors with the training images descriptors. The output of this function, **hist_list_test** is then passed to the **clf.predit**. The **clf.predict** gives the predicted data which is then passed to the accuracy function along with the actual labels of the test data, that is **hist_labels_test**, to calculate accuracy of the images learned. Then same arrays are passed to the confusion matrix. And lastly the run time is calculated in order to observe time differences with different data sets and other variables.

Without spatial pyramids and training set=200
Test Accuracy: 0.31555555555555553
Run time: 03 hrs 05 mins


Confusion matrix

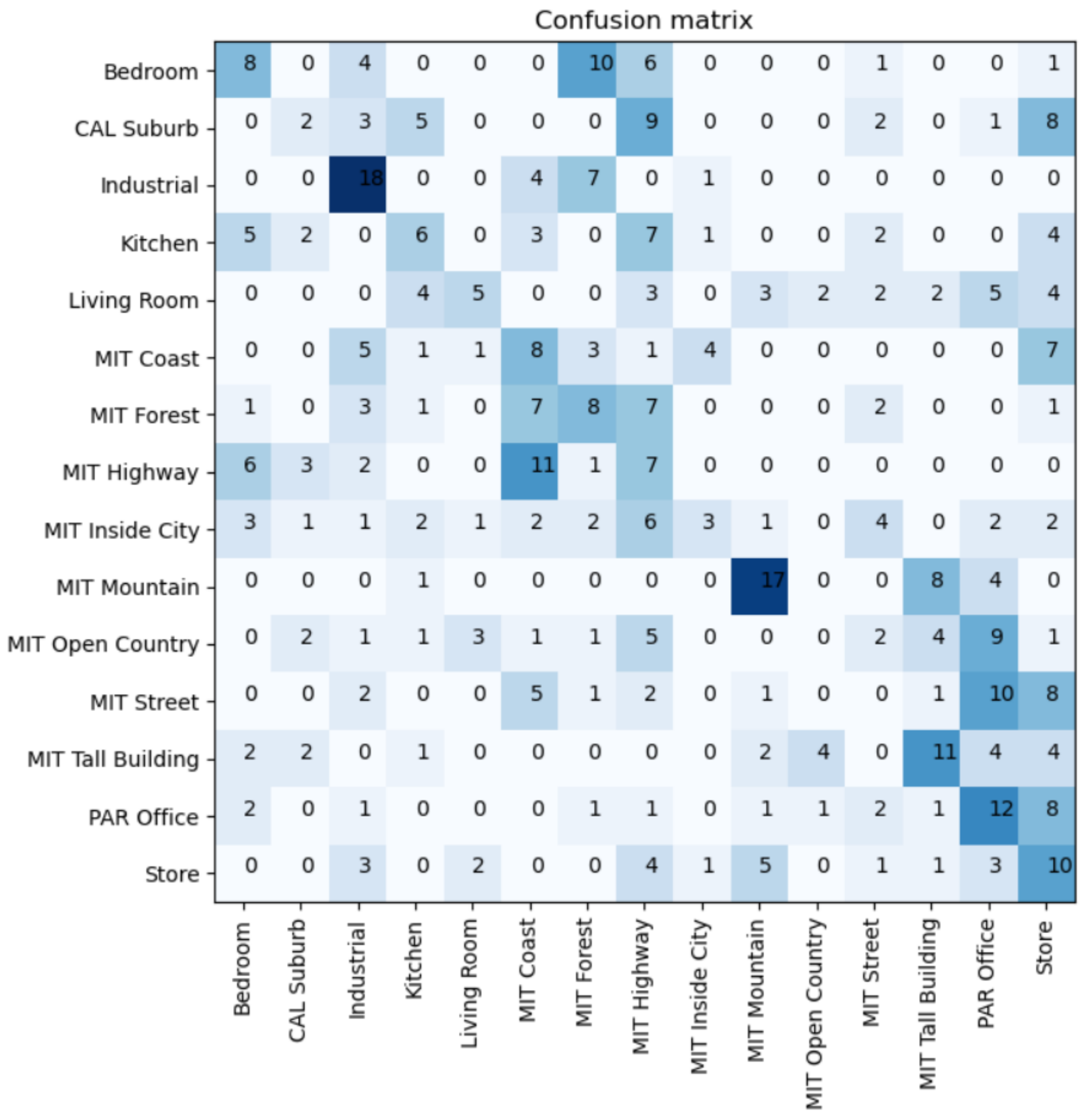
	Bedroom	CAL Suburb	Industrial	Kitchen	Living Room	MIT Coast	MIT Forest	MIT Highway	MIT Inside City	MIT Mountain	MIT Open Country	MIT Street	MIT Tall Building	PAR Office	Store
Bedroom	24	0	0	4	0	4	6	2	0	1	0	0	0	0	0
CAL Suburb	11	7	3	19	40	8	4	10	8	36	0	12	0	1	1
Industrial	0	0	117	0	0	4	7	0	0	0	0	0	0	0	0
Kitchen	4	0	1	20	10	8	9	1	4	1	0	1	1	0	0
Living Room	5	4	0	18	37	9	2	10	7	13	0	3	0	0	0
MIT Coast	13	0	5	3	9	92	15	21	15	0	0	1	0	0	0
MIT Forest	19	0	41	7	2	47	82	4	4	0	0	4	0	0	0
MIT Highway	4	0	0	3	4	16	2	48	11	1	0	2	0	1	0
MIT Inside City	11	1	0	7	26	32	9	24	21	13	0	8	1	3	0
MIT Mountain	0	0	0	0	1	0	0	0	0	12	0	0	2	0	0
MIT Open Country	1	0	0	2	0	0	1	0	1	3	1	1	3	2	1
MIT Street	3	0	0	5	11	8	10	4	6	2	7	14	4	11	26
MIT Tall Building	0	0	0	1	0	0	0	0	0	1	3	0	4	0	1
PAR Office	1	0	0	1	8	1	0	5	3	18	6	1	8	28	9
Store	1	2	1	2	11	25	2	1	10	1	6	11	2	6	34

With spatial pyramids and training set=200
Test Accuracy: 0.36430976430976433
Run time: 06 hrs 41 mins 04.20 sec



	Bedroom	CAL Suburb	Industrial	Kitchen	Living Room	MIT Coast	MIT Forest	MIT Highway	MIT Inside City	MIT Mountain	MIT Open Country	MIT Street	MIT Tall Building	PAR Office	Store
Bedroom	11	0	3	1	0	4	5	4	0	0	0	2	0	0	0
CAL Suburb	1	4	2	4	0	4	0	5	1	0	0	4	0	0	5
Industrial	0	0	18	0	0	7	4	0	1	0	0	0	0	0	0
Kitchen	2	1	0	9	0	0	0	5	3	0	0	3	2	2	3
Living Room	2	0	0	6	6	0	0	1	1	2	1	2	4	1	4
MIT Coast	0	0	1	1	0	15	4	1	1	0	0	2	1	0	4
MIT Forest	0	1	0	3	0	10	10	0	2	0	0	3	0	0	1
MIT Highway	5	2	2	2	0	10	1	8	0	0	0	0	0	0	0
MIT Inside City	2	1	0	1	0	6	1	6	4	1	0	4	2	2	0
MIT Mountain	0	1	0	0	1	0	0	0	0	19	0	0	7	1	1
MIT Open Country	2	1	0	0	3	3	1	4	0	1	3	2	4	4	2
MIT Street	0	0	0	0	0	7	1	2	0	1	1	4	3	6	5
MIT Tall Building	1	0	0	1	3	0	0	0	0	4	1	1	15	2	2
PAR Office	3	0	1	1	2	1	0	0	2	0	2	1	2	9	6
Store	0	1	2	0	0	3	0	5	0	4	0	0	0	8	7

With spatial pyramids and training set=50
Test Accuracy: 0.25555555555555554
Run time: 02 hrs 17 mins 03.65 sec


Confusion matrix

	Bedroom	CAL Suburb	Industrial	Kitchen	Living Room	MIT Coast	MIT Forest	MIT Highway	MIT Inside City	MIT Mountain	MIT Open Country	MIT Street	MIT Tall Building	PAR Office	Store
Bedroom	8	0	4	0	0	0	10	6	0	0	0	1	0	0	1
CAL Suburb	0	2	3	5	0	0	0	9	0	0	0	2	0	1	8
Industrial	0	0	18	0	0	4	7	0	1	0	0	0	0	0	0
Kitchen	5	2	0	6	0	3	0	7	1	0	0	2	0	0	4
Living Room	0	0	0	4	5	0	0	3	0	3	2	2	2	5	4
MIT Coast	0	0	5	1	1	8	3	1	4	0	0	0	0	0	7
MIT Forest	1	0	3	1	0	7	8	7	0	0	0	2	0	0	1
MIT Highway	6	3	2	0	0	11	1	7	0	0	0	0	0	0	0
MIT Inside City	3	1	1	2	1	2	2	6	3	1	0	4	0	2	2
MIT Mountain	0	0	0	1	0	0	0	0	0	17	0	0	8	4	0
MIT Open Country	0	2	1	1	3	1	1	5	0	0	0	2	4	9	1
MIT Street	0	0	2	0	0	5	1	2	0	1	0	0	1	10	8
MIT Tall Building	2	2	0	1	0	0	0	0	0	2	4	0	11	4	4
PAR Office	2	0	1	0	0	0	1	1	0	1	1	2	1	12	8
Store	0	0	3	0	2	0	0	4	1	5	0	1	1	3	10

Conclusions

Most of the time the algorithm took was when it generated the histograms. Besides that, making a confusion matrix was also a bit trickier than expected. After running the algorithm on different data sets and with varying number of clusters, it was evident that why people now prefer using deep neural networks as it gives considerably higher accuracy. The accuracy is considerably low in our case due to the fact that we have used randomly selected 1000 descriptors from each image. Using all the descriptors from each image gave better results but running it on 50 or 200 training set would increase the run time.