Assignment #2

EunSu Yeo

March 5, 2025

1 Scene Recognition

(a) Implement Code

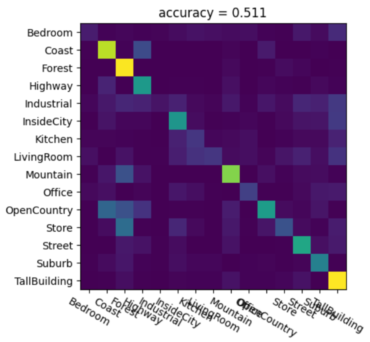
- The function `predict_knn` is implemented to accept `k` as a variable, as required by the assignment slides. However, We need to find the best k that gives the accurate result in the `classify_knn_bow` function.
- For the `build_visual_dictionary` function, we considered the number of iterations used in the `KMeans` function from `sklearn.cluster`.
- In `predict_svm`, the `lambda` regularization is controlled by the argument `C` in `LinearSVC`. We compared the performance at `C = 0.1`, `C = 1`, and `C = 10`, and determined at which value of `C` the model achieved the best accuracy.

(b) Write Report

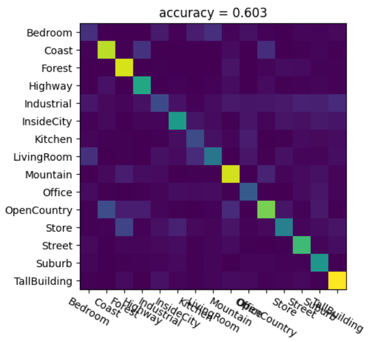
The hyperparameters we aim to optimize are the number of iterations in KMeans, the value of C used to compute lambda, number of k in KMeans and the dictionary size (dic_size).

(1) First, Let's think about the number of iterations in KMenas function. We know that the more iterations KMeans performs, the more accurate the resulting centroids are likely to be. With Scikit-learn 1.2 or higher, n_init='auto' is used by default in KMeans, which performs one initialization using k-means++. I relied on this default behavior before tuning other parameters to improve clustering accuracy.

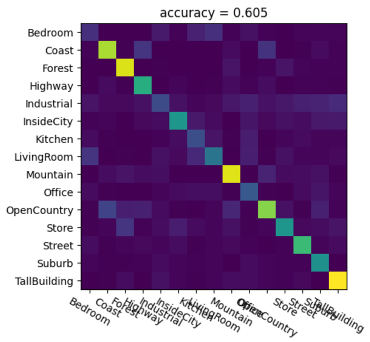
(2) We need to get the C and dic_size which gives us the highest accuracy of SVM BoW. Let's check the C value first when dic_size is constant. To check the difference between C values, I set the dic_size=50 and changed the C values.
As the image below, We can see that C=1 and C=10 gives the simliar accuracy. So there might be value of C that maximize the accuracy in between 1 and 10.



(a) C=0.1, $\lambda = 10$



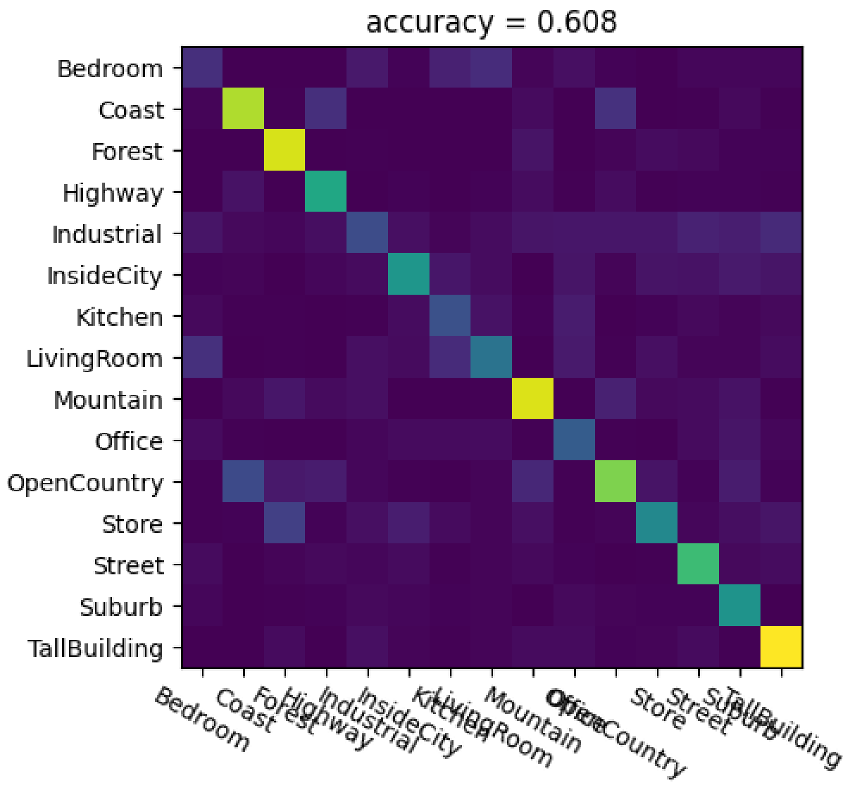
(b) C=1, $\lambda = 1$



(c) C=10, $\lambda = 0.1$

Figure 1: Confusion matrices for SVM BoW with different lambda size

So let's briefly set the C=2 which is the mean value of lambda 1 and 0.1. Below is the image of the result. This shows highest accuracy among the results as we predicted.

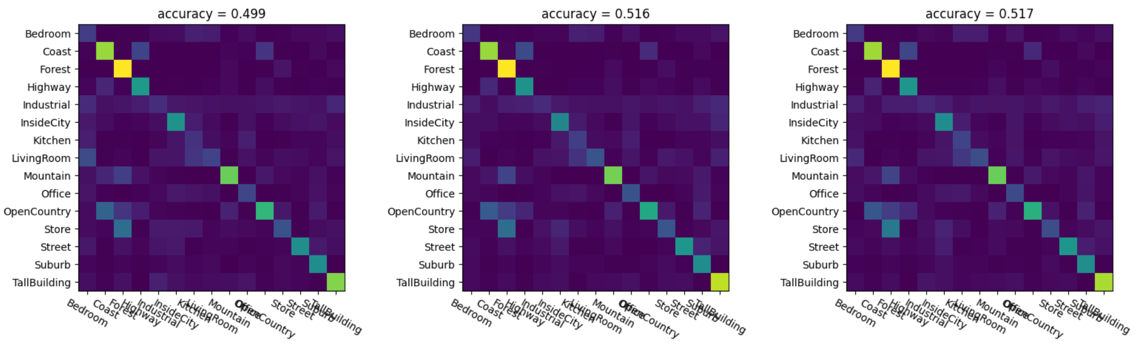


(a) C=2, $\lambda = 0.5$

Figure 2: Confusion matrices for SVM BoW with Adjusted C value

(3) We need to determine the best value of K (the number of nearest neighbors) which gives us the highest accuracy of SVM BoW. To check the effect of K, I fixed dic_size = 50 and varied K values. After the size of k=20 KMean might give us too brief outputs so I compared 3

cases of k. As shown in the image below, we can see the result of k = 5, k=10, k=15.

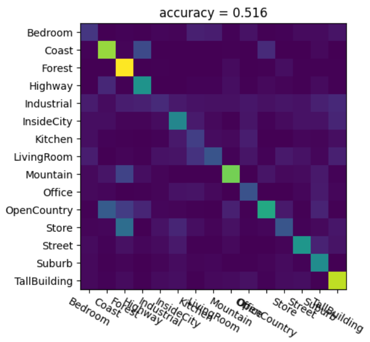


(a) k=5 (b) k=10 (c) k=15

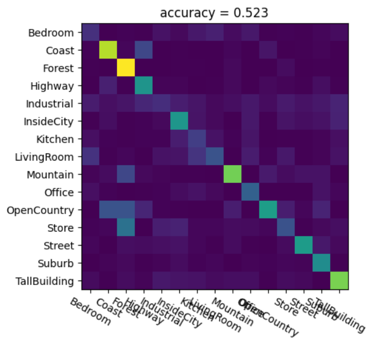
Figure 3: Confusion matrices for KNN BoW with different k size

As we can see from the result, when k is 15 it gives the highest accuracy. So let's briefly set the k=15 to get the best value of BoW.

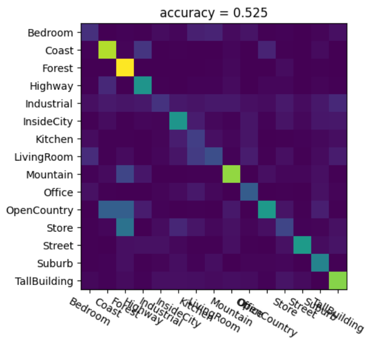
(4) Now, Let's find the best dictionary size that gives the highest accuracy when other parameters are fixed. Image below is the result of KNN and SVM BoW with different dictionary size.



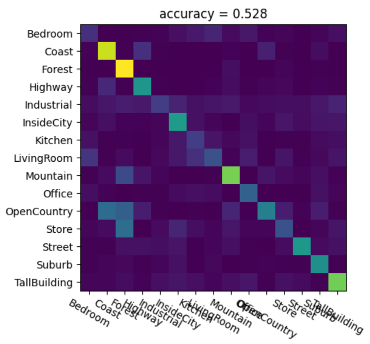
(a) KNN, dic_size=50



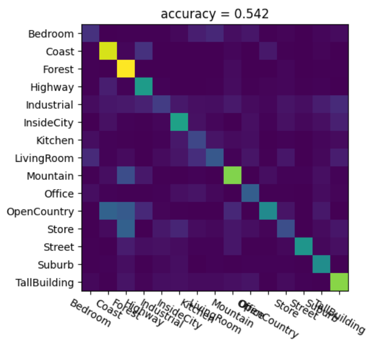
(b) KNN, dic_size=100



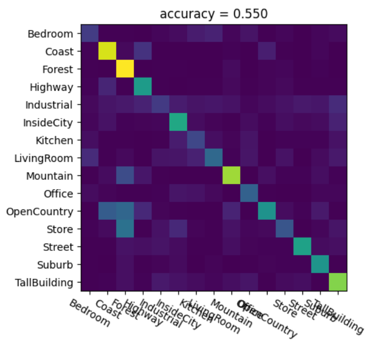
(c) KNN, dic_size=150



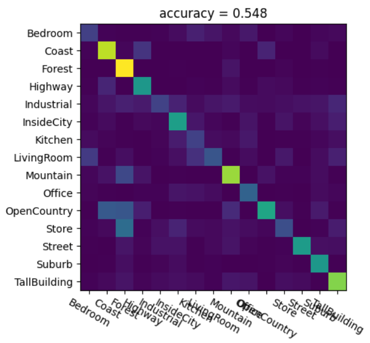
(d) KNN, dic_size=200



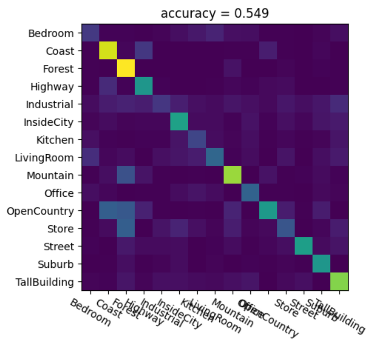
(e) KNN, dic_size=250



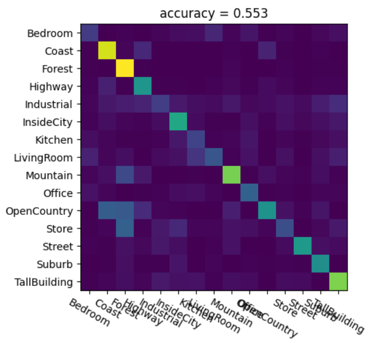
(f) KNN, dic_size=300



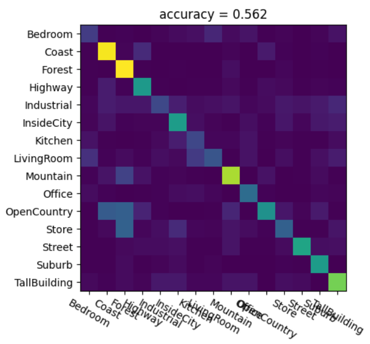
(g) KNN, dic_size=350



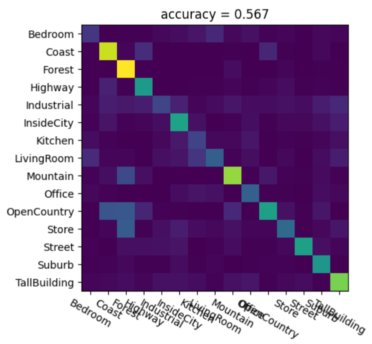
(h) KNN, dic_size=400



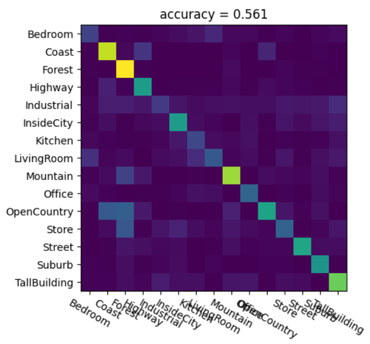
(i) KNN, dic_size=450



(j) KNN, dic_size=500

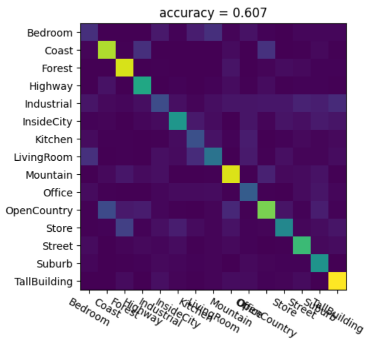


(k) KNN, dic_size=550

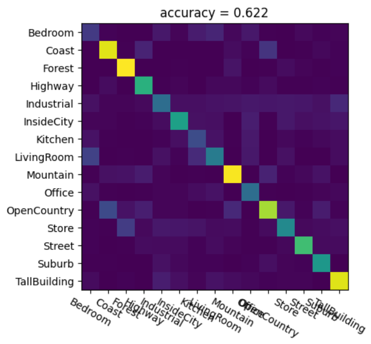


(l) KNN, dic_size=600

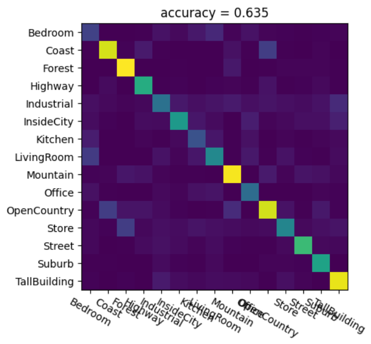
Figure 4: Confusion matrices for KNN BoW with different dictionary sizes



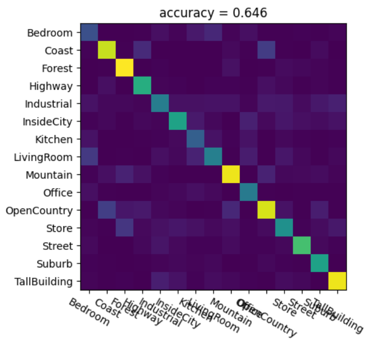
(a) SVM, dic_size=50



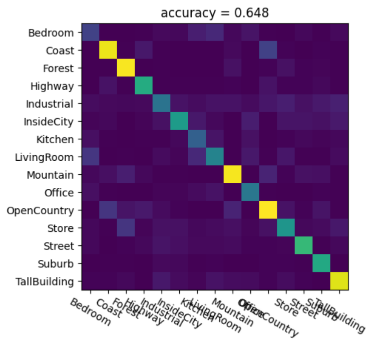
(b) SVM, dic_size=100



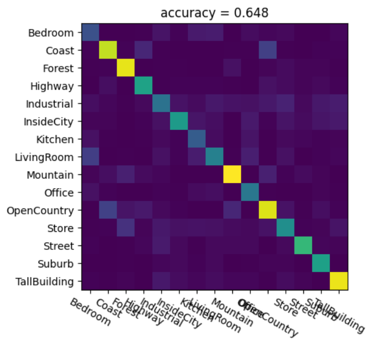
(c) SVM, dic_size=150



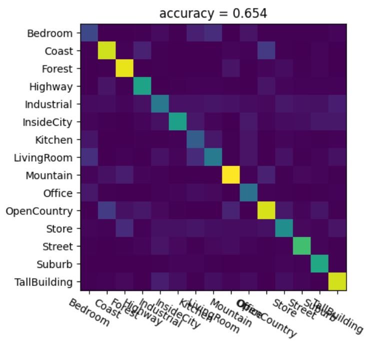
(d) SVM, dic_size=200



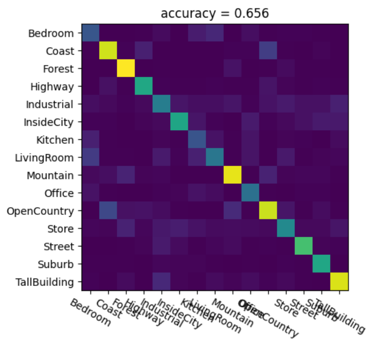
(e) SVM, dic_size=250



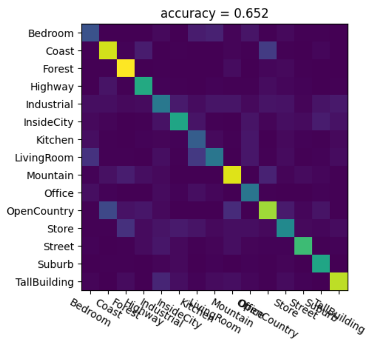
(f) SVM, dic_size=300



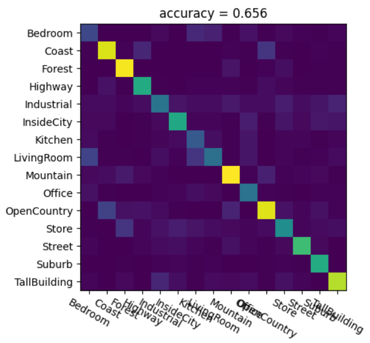
(g) SVM, dic_size=350



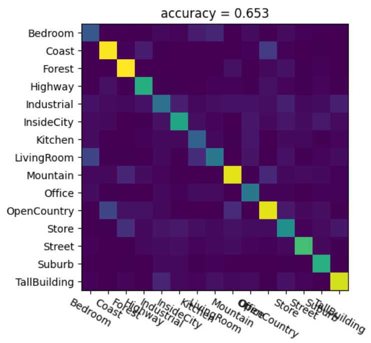
(h) SVM, dic_size=400



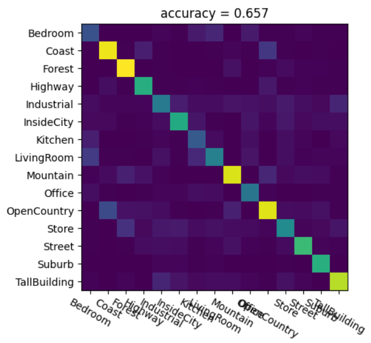
(i) SVM, dic_size=450



(j) SVM, dic_size=500



(k) SVM, dic_size=550



(l) SVM, dic_size=600

Figure 5: Confusion matrices for SVM BoW with different dictionary sizes

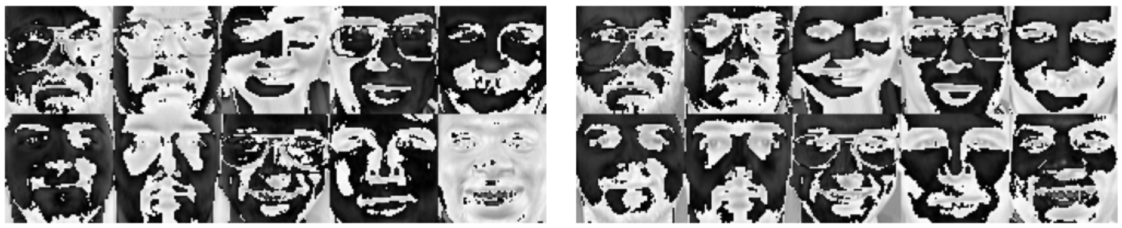
As the running time for large `dic_size` values became too long, we incrementally increased it by 50 starting from 50, aiming to roughly find the point where reaches the highest accuracy.

From the results in (a) through (l) in Figure 4, it can be reasonably inferred that the best dictionary size for KNN is 550, as the accuracy reaches 0.567 at dic_size=550 and slightly decreases at dic_size=600.

From the results in (a) through (l) in Figure 5, it can be reasonably inferred that the best dictionary size for SVM is 600, as the accuracy reaches 0.657 at dic_size=600 and slightly decreases before dic_size=600.

From these results we can find out the best hyperparameters are k=15, C=2. For KNN the best dic_size = 550 and for SVM the best dic_size = 600.

2 Eigen Face



(a) faces_centered_1 (b) faces_centered_2

Figure 6: Result of different normalization

(1) As we can see from the image above, the first normalization subtracts mean of total face from the given face. This makes the output by removing common features and enabling PCA to find directions of variance. This works as the standard preprocessed step for eigenface decomposition in the code.

The second normalization subtracts the mean of each individual image from the image itself. This helps reduceing illumination variation because it removes specific global intensity. You can see this from the black part of image. For 2nd and Last image, second method gives more darker part than the first method. Subtracting it's own images highlights the difference with the bright part more. Due to this property it's less suitable for PCA because it gives less important global structures than the previous normalization model. Instead, it is useful when comparing images using similarity metrics.

(2) Representing a face using eigen-coefficients gives the output. In the process, each image is projected to ordered eigenvector space. By using only top-k eigenvectors, the dimension of the data decreases while still having most of the essential information. This decrease gives us the result of suppression that removes noise and improves computational efficiency. Due to

this property its used for similar tasks in face comparison.

However, decreased number of eigen coefficients comes with tradeoffs. This may also remove important discriminative features, especially when the discarded components contain subtle but relevant identity cues.

In our experiment using cosine similarity, we compared the eigenface representations of two different image pairs. The first pair consisted of two visually similar female faces and yielded a relatively high cosine similarity of 0.4597. The second pair, consisting of clearly different male faces, had a much lower similarity of 0.0408. This result supports the idea that even with reduced eigen-coefficients, the top principal components are effective at capturing major identity-related features. However, finer distinctions such as between similar individuals or expressions may require retaining more components to avoid information loss.

References

[1] scikit-learn developers. *sklearn.cluster.KMeans – scikit-learn 1.4.2 documentation.* Accessed: May 5, 2025.
`https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`

[2] scikit-learn developers. *sklearn.svm.LinearSVC – scikit-learn 1.4.2 documentation.* Accessed: May 5, 2025.
`https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`

[3] scikit-learn developers. *sklearn.neighbors.NearestNeighbors – scikit-learn 1.4.2 documentation.* Accessed: May 5, 2025.
`https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html`