ENSE 885AY

Application of Deep Learning in Computer Vision

Assignment A04

Scene recognition with bag of words

Instructed by

Dr. Kin-Choong Yow

Student:

Marzieh Zamani Alavijeh

List of Sections and Sub-sections

1. Introduction

1.1. Overview (Key points from the assignment description) [1]

2. Student Code

2.1. Represent images using tiny images (get_tiny_images())

Algorithm of get_tiny_images() [1]:

2.2. Represent images using bags_of_sifts (build_vocabulary() & get_bags_of_sifts())

Algorithm of build_vocabulary() [1]:

Algorithm of get_bags_of_sifts() [1]:

2.3. Classify images using KNN (nearest_neighbor_classify())

Algorithm of nearest_neighbor_classify() [1]:

2.4. Classify images using 1 vs. All SVM (svm_classify())

Algorithm of svm_classify() [1]:

3. Experiment Design

3.1. Cross Validation with Different Vocabulary Sizes

Cross validation experiment summary:

Cross validation experiment algorithm:

3.2. Validation with Different Vocabulary Sizes and Different SVM Lambda Values

Validation experiment summary:

Validation experiment algorithm:

4. Results and Discussion

4.1. Results for KNN on tiny_images

4.2. Results for KNN on bags_of_sifts

4.3. Results for SVM on bags_of_sifts

4.4. Results for Cross Validation with Different Vocabulary Sizes

4.5. Results for Validation with Different Vocabulary Sizes and Different SVM Lambda Values

Extra Works

References

1. Introduction

1.1. Overview (Key points from the assignment description) [1]

Assignment Subject:

Scene recognition with bag of words

Assignment objectives:

- The goal of this project is to get familiar with image recognition.

Steps to local feature matching between two images (image1 & image 2):

1. Represent images using:
    - ⇨ get_tiny_images()
    - ⇨ build_vocabulary()
    - ⇨ get_bags_of_sifts()

2. Classify images using:
    - ⇨ nearest_neighbor_classify()
    - ⇨ svm_classify()

2. Student Code

2.1. Represent images using tiny images (get_tiny_images())

Algorithm of get_tiny_images() [1]:

feats = get_tiny_images(image_paths)

1. Define feature dimension:
    a. feat_dim = 16

for each image path in {image_paths}:

2. Load image as grayscale
3. Resize image to 16x16 (according to feat_dim)
4. Normalize image

    a. Norm_img = (img - np.mean(img))/np.std(img)
5. Return resized & normalized image as tiny_image feature

2.2. Represent images using bags_of_sifts (build_vocabulary() & get_bags_of_sifts())

Algorithm of build_vocabulary() [1]:

vocab = build_vocabulary(image_paths, vocab_size)

for each image path in {image_paths}:

1. Load image as grayscale
2. Normalize image
    a. Norm_img = (img - np.mean(img))/np.std(img)
3. Obtain descriptor using "vlfeat.sift.dsift" function
    a. _, descriptors = vlfeat.sift.dsift(img, step=[50,50], fast=True)
4. Append image descriptors to bag_of_features
5. Cluster bag_of_features using "vlfeat.kmeans.kmeans"
    a. vocab = vlfeat.kmeans.kmeans(bag_of_features, vocab_size, initialization="PLUSPLUS")
6. Return cluster centroids as vocab

Remark on step size:

The choice of step=[50,50] is through trying different step sizes. This value results in about 42000 descriptor which is a reasonable number.

Algorithm of get_bags_of_sifts() [1]:

feats = get_bags_of_sifts(image_paths, vocab_filename):

1. Load vocab from saved files

    for each image path in {image_paths}:

2. Load image as grayscale
3. Normalize image
    a. Norm_img = (img - np.mean(img))/np.std(img)

4. Obtain descriptor using “vlfeat.sift.dsift” function
    a. _, descriptors = vlfeat.sift.dsift(img, step=[9,9], fast=True)
5. Assign each descriptor vector to nearest cluster center using “vlfeat.kmeans.kmeans_quantize” function
    a. assignments = vlfeat.kmeans.kmeans_quantize(descriptors.astype(np.float32), vocab)
6. Obtain histogram of features using “np.histogram” function
    a. histo, _ = np.histogram(assignments, range(len(vocab)+1))
7. Normalize histogram and append to feats
    a. feats.append(histo / np.linalg.norm(histo))
8. Return histograms of descriptor as feats

Remark on step size:

The choice of step=[9, 9] is through trying different step sizes. This value results in about 0.1 of the step size which was used in build_vocabulary() function.

2.3. Classify images using KNN (nearest_neighbor_classify())

Algorithm of nearest_neighbor_classify() [1]:

test_labels = nearest_neighbor_classify(train_image_feats, train_labels, test_image_feats, metric='euclidean'):

1. Define k for KNN
    a. k = 16
2. Computes the distance matrix D between all pairs of test & train images
    a. D = sklearn_pairwise.pairwise_distances(test_image_feats, train_image_feats, 'euclidean')
3. Find the K nearest features (train images) to each test image feature by sorting D matrix along every row
    a. Row [i] of matrix D is corresponding to distances between test image [i] and all train images

For each test image [i]:

4. Obtain the labels for K nearest train images to test image [i]
5. Predict the label of test image [i] by voting among K labels
6. Return predicted labels as test_labels

Remark on K:

The choice of K = 16 is through trying different values. This value ensures that labels will be predicted by obtaining at least 2 votes (not by chance).

2.4. Classify images using 1 vs. All SVM (svm_classify())

Algorithm of svm_classify() [1]:

test_labels = svm_classify(train_image_feats, train_labels, test_image_feats, lambd=3)

1. Obtain list of categories
2. Construct 1 vs all SVMs for each category:
    a. svms = {cat: LinearSVC(random_state=0, tol=1e-3, loss='hinge', C=lambd, max_iter=10000) for cat in categories}

One vs. All SVC:

for each category[k]:

3. Obtain binary train labels (by setting the category label to "1" and all other labels to "0")
4. Fit SVC on binary train data
    a. svc.fit(train_image_feats, train_labels_binary)
5. Obtain prediction confidence for test images using the trained svc
    a. pred_conf[:,cat_idx] = svc.decision_function(test_image_feats)

for each test image[i]:

6. Obtain the highest prediction confidence and its corresponding category
7. Predict the label according to the label with highest confidence
8. Return predicted labels as test_labels

3. Experiment Design

3.1. Cross Validation with Different Vocabulary Sizes

Cross validation experiment summary:

- Number of experiments: exp_num = 10
- Dataset:

    - Train dataset: 150 random images from original train dataset
    - Test dataset: 150 random images from original test dataset
- Varying parameters:
    - vocab_zize = [10, 20, 50, 100, 200, 400, 1000, 10000]
- Approaches:
    - KNN on tiny_images
    - KNN on bags_of_sifts
    - SVM on bags_of_sifts
- Total number of experiments: 10 * (1 + 8 + 8) = 170
- Results:
    - Accuracy average per approach, per vocab_size
    - Accuracy standard deviation, per approach per vocab_size

Cross validation experiment algorithm:

1. Define constant and varying parameters

   exp_number = 10

   cv_size = 150

2. Define output matrices

   CV_acc_knn_tiny: Prediction accuracy for KNN on tiny_images

   CV_acc_knn_sift: Prediction accuracy for KNN on bags_of_sifts

   CV_acc_svm_sift: Prediction accuracy for SVM on bags_of_sifts

Experiment on random train & test datasets:

for exp_idx = [0 10):

3. Generating random indexes for trainCV & testCV datasets
4. Obtain trainCV & testCV dataset by indexing

KNN on tiny_images:

5. Obtain tiny_images features for trainCV & testCV dataset
6. Apply KNN on tiny_images
7. Save accuracy

KNN on bags_of_sifts:

for vocab_size = [10, 20, 50, 100, 200, 400, 1000, 10000]:

8. Obtain bags_of_sifts features for trainCV & testCV dataset
9. Apply KNN on bags_of_sifts
10. Save accuracy

SVM on bags_of_sifts:

for vocab_size = [10, 20, 50, 100, 200, 400, 1000, 10000]:

11. Obtain bags_of_sifts features for trainCV & testCV dataset
12. Apply KNN on bags_of_sifts
13. Save accuracy

Obtain average accuracy among experiments:

CV_acc_knn_tiny_mean

CV_acc_knn_sift_mean

CV_acc_svm_sift_mean

Obtain standard deviation of accuracy among experiments:

CV_acc_knn_tiny_std

CV_acc_knn_sift_std

CV_acc_svm_sift_std

3.2. Validation with Different Vocabulary Sizes and Different SVM Lambda Values

Validation experiment summary:

- Number of experiments: 1
- Dataset:
  - Train dataset: 1050 images from original train dataset (70 images per class)
  - validation dataset: 450 images from original train dataset (30 images per class)

- Varying parameters:
  - vocab_zize = [10, 20, 50, 100, 200, 400, 1000, 10000]
  - svm_lambda = [0.1, 0.5, 1, 3, 5, 7]
- Approaches:
  - KNN on tiny_images
  - KNN on bags_of_sifts
  - SVM on bags_of_sifts
- Total number of experiments: exp_num = 1 * (1 + 8 + 8*6) = 57
- Results:
  - Accuracy per approach, per vocab_size, per svm_lambda

Validation experiment algorithm:

1. Define constant and varying parameters
   N_trainV = 1050 images from original train dataset (70 images per class)

   N_validation = 450 images from original train dataset (30 images per class)

2. Generating indexes for trainV & validation datasets (uniform number of samples per class)
3. Obtain trainV & validation dataset by indexing
4. Define output matrices

   val_knn_tiny: Prediction accuracy for KNN on tiny_images

   val_knn_sift: Prediction accuracy for KNN on bags_of_sifts

   val_svm_sift: Prediction accuracy for SVM on bags_of_sifts

Experiment on train & validation datasets:

KNN on tiny_images:

5. Obtain tiny_images features for trainV & validation dataset
6. Apply KNN on tiny_images
7. Save accuracy

KNN on bags_of_sifts:

for vocab_size = [10, 20, 50, 100, 200, 400, 1000, 10000]:

8. Obtain bags_of_sifts features for trainV & validation dataset
9. Apply KNN on bags_of_sifts
10. Save accuracy

SVM on bags_of_sifts:

for vocab_size = [10, 20, 50, 100, 200, 400, 1000, 10000]:

for lambda = [0.1, 0.5, 1, 3, 5, 7]:

11. Obtain bags_of_sifts features for trainV & validation dataset
12. Apply KNN on bags_of_sifts
13. Save accuracy

Return validation accuracy:

val_knn_tiny

val_knn_sift

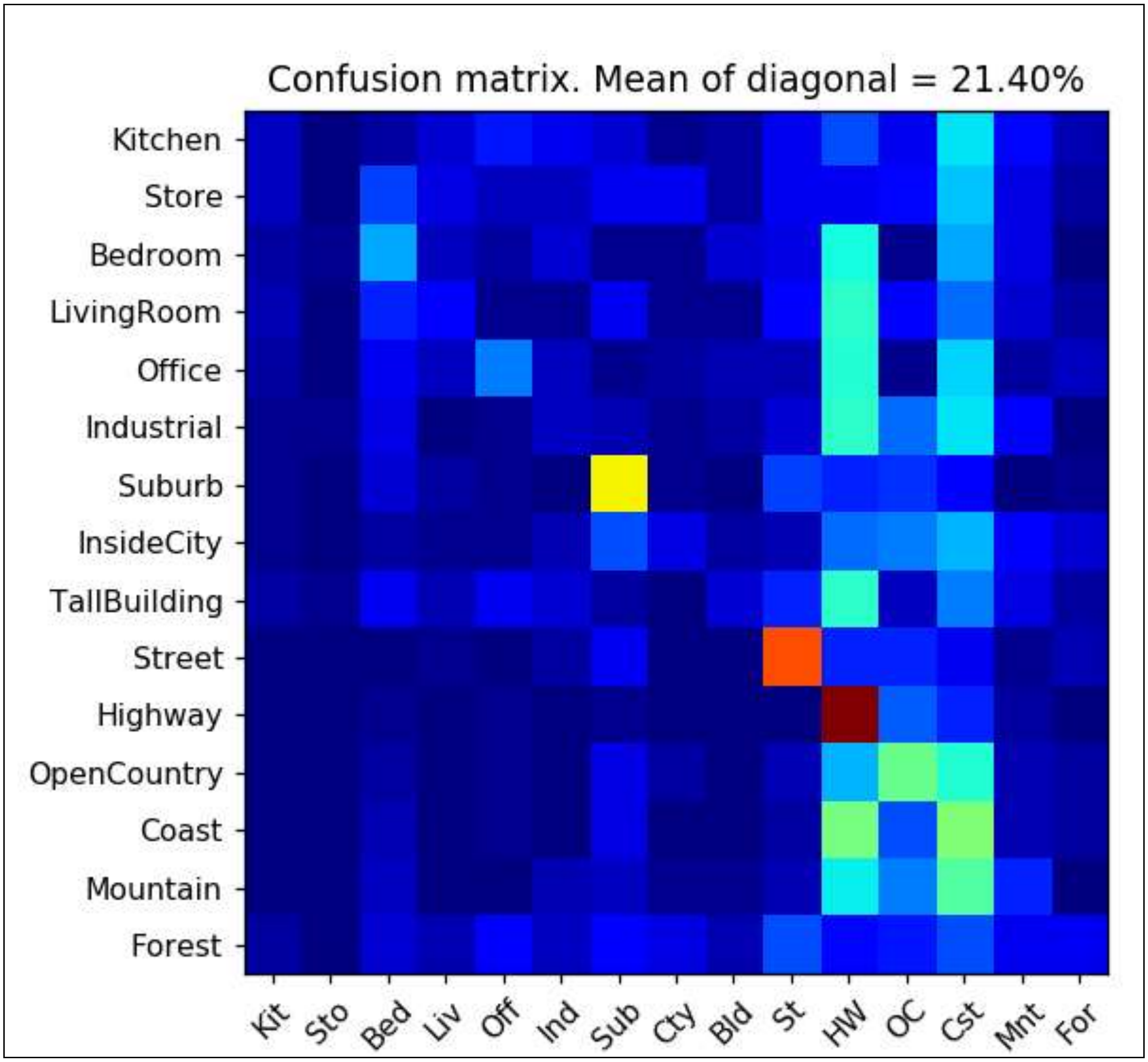
val_svm_sift

4. Results and Discussion

4.1. Results for KNN on tiny_images

Table 1: KNN on tiny_images (K = 16)

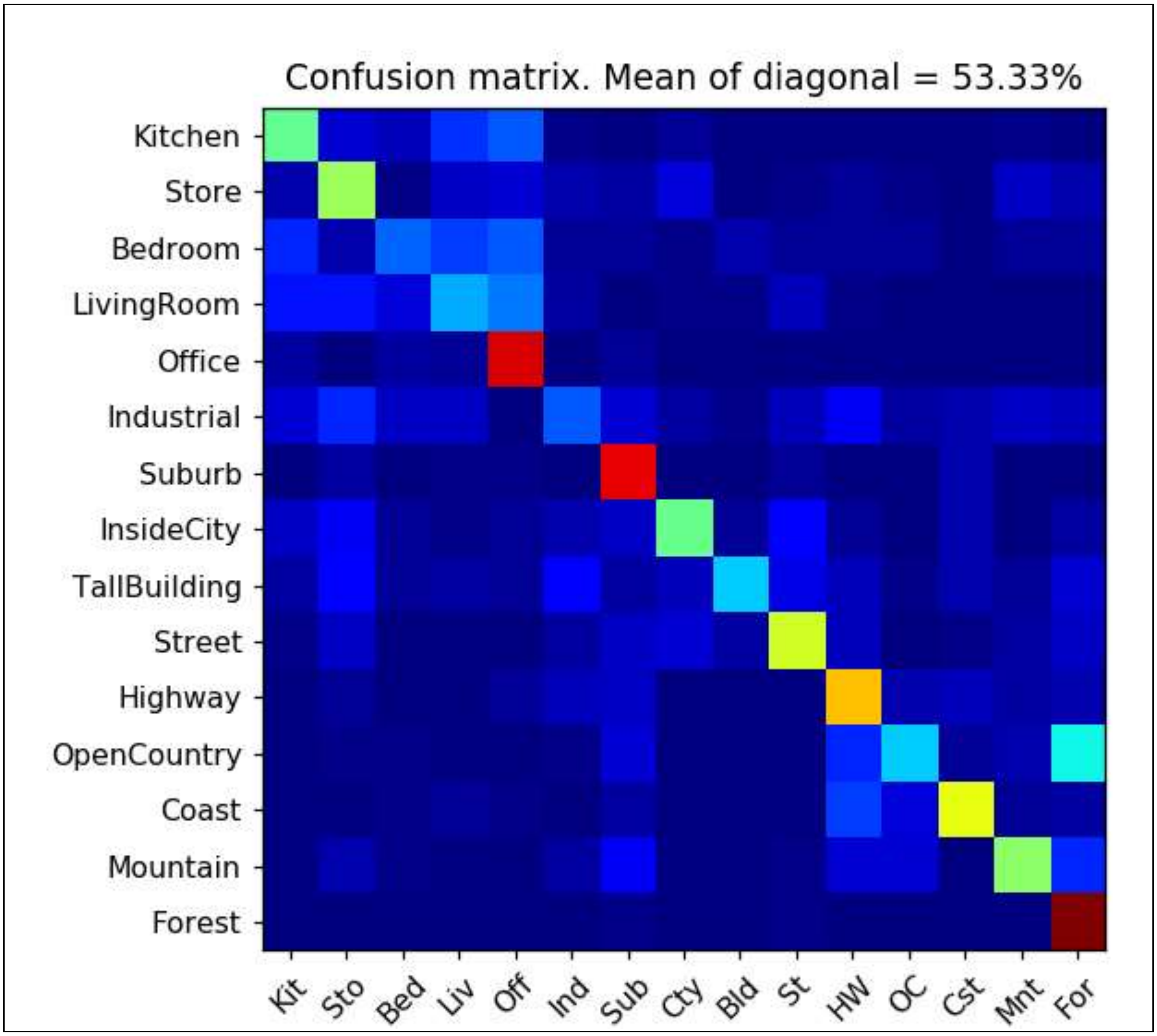
KNN on tiny_images
K = 16
Accuracy = 21.4 %



4.2. Results for KNN on bags_of_sifts

Table 2: KNN on bags_of_sifts (K = 16)

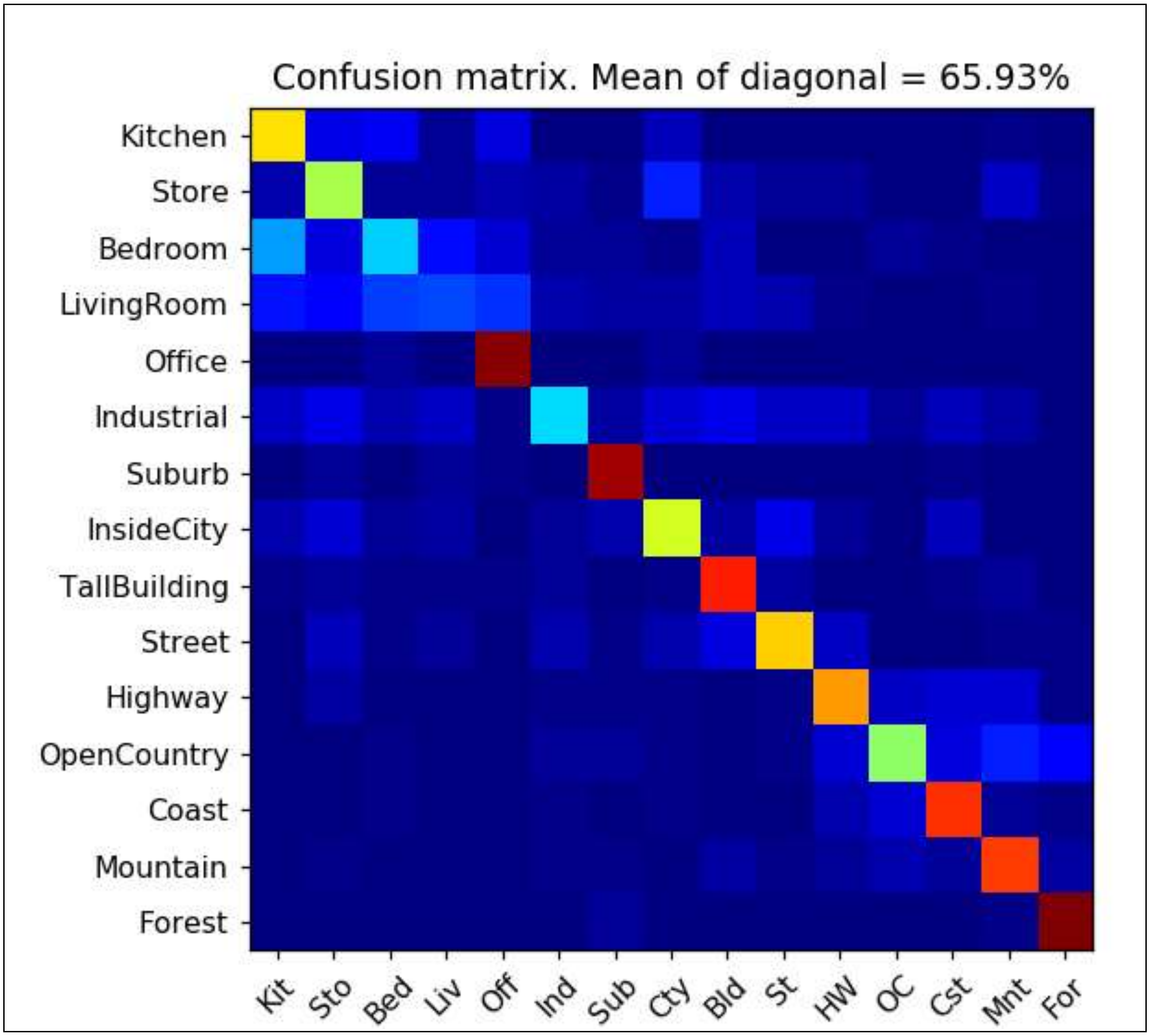
KNN on bags_of_sifts
Vocab_size = 200 & K = 16
Accuracy = 53.33%



4.3. Results for SVM on bags_of_sifts

Table 3: SVM on bags_of_sifts (K = 16)

SVM on bags_of_sifts
Vocab_size = 200 & SVM_Lambda = 3
Accuracy = 65.93%



4.4. Results for Cross Validation with Different Vocabulary Sizes

Table 4: Results for Cross Validation with Different Vocabulary Sizes

Cross validation results		Vocab size							
		10	**20**	**50**	**100**	**200**	**400**	**1000**	**10,000**
KNN on tiny_images	**Accuracy Average**	0.13							
	Accuracy Std.	0.011							

KNN on bags_of_sifts	**Accuracy Average**	0.30	0.31	0.33	0.32	0.34	0.36	0.33	0.33
	Accuracy Std.	0.051	0.044	0.050	0.032	0.052	0.039	0.033	0.034
SVM on bags_of_sifts	**Accuracy Average**	0.30	0.37	0.41	0.44	0.46	0.47	0.48	0.46
	Accuracy Std.	0.030	0.014	0.037	0.050	0.045	0.050	0.049	0.067

Table 5: Cross Validation Results: Average Prediction Accuracy vs. Vocab Size & Approach

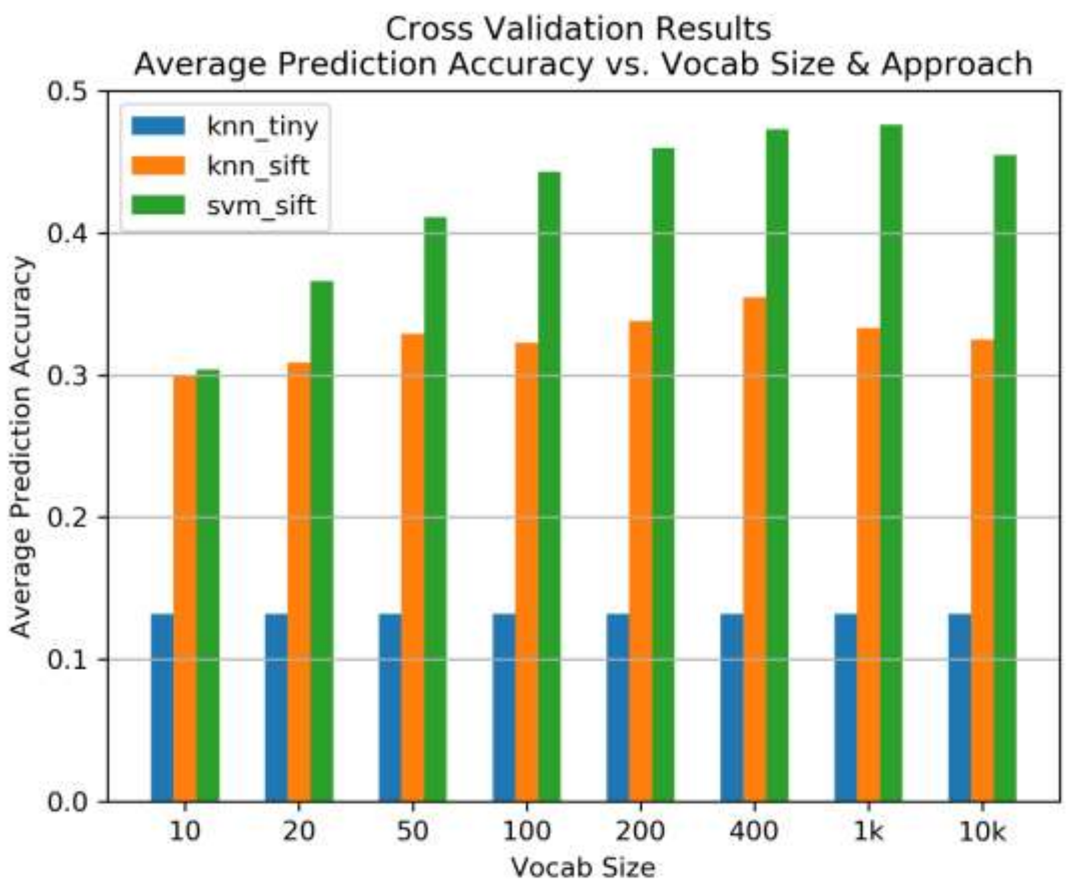
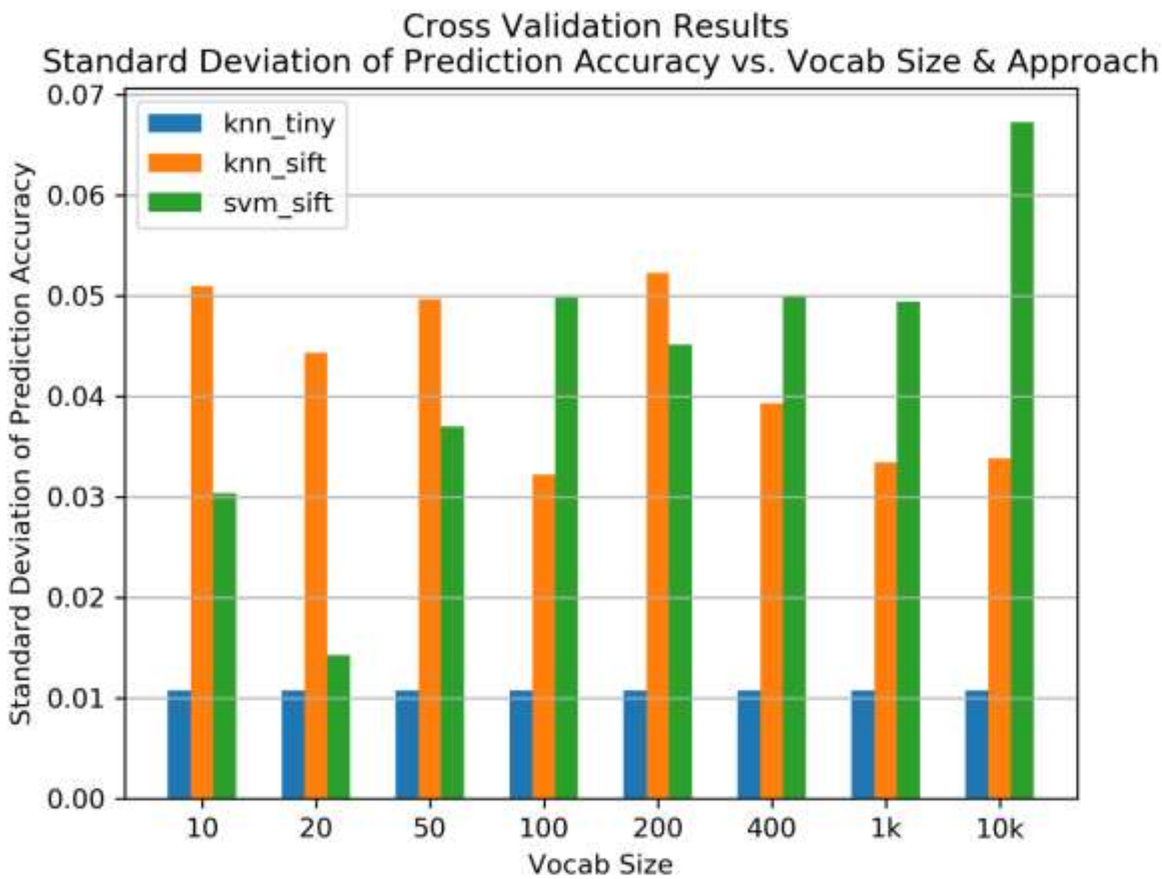


Table 6: Cross Validation Results: Standard Deviation of Prediction Accuracy vs. Vocab Size & Approach



Remarks on the cross-validation results:

Experiment design summary (rewritten here for convenience of reader):

- Number of experiments: exp_number = 10
- Dataset:
  - Train dataset: 150 random images from original train dataset
  - Test dataset: 150 random images from original test dataset
- Varying parameters:
  - vocab_size = [10, 20, 50, 100, 200, 400, 1000, 10000]
- Approaches:
  - KNN on tiny_images
  - KNN on bags_of_sifts
  - SVM on bags_of_sifts
- Total number of experiments: 10 * (1 + 8 + 8) = 170
- Results:
  - Accuracy average per approach, per vocab_size

- Accuracy standard deviation, per approach per vocab_size

Accuracy average vs. approach:

- It is worth mentioning that for KNN on tiny_images, accuracy is a single value, independent of vocab_size (all blue bars in the plot are identical).
- As expected, the average accuracy for KNN on tiny_images is the least value (13% ±1%);
- Next accuracy is obtained by KNN on bags_of_sifts (average: 33% ± 4%).
- The highest accuracy is obtained by SVM on bags_of_sifts (average: 42% ± 4%).

Accuracy average vs. vocab_size:

- When using bags_of_sifts features, the accuracy might also depend on vocab_size:
- For SVM on bags_of_sifts, average cross validation accuracy is increasing proportional to vocab_size until vocab_size = 1k; and then slightly decreases for vocab_size = 10k.
- However, for KNN on bags_of_sifts, average cross validation accuracy is almost independent of vocab_size.

Accuracy standard deviation vs. approach:

- As expected, the standard deviation for KNN on tiny_images is the least value (±1%);
- The average standard deviation is almost equal for KNN/SVM on bags_of_sifts (± 4%).

Accuracy standard deviation vs. vocab_size:

- While standard deviation is varying for different vocab_size, there does not seem to be a significant relationship between standard deviation of accuracy and vocab_size.

Overall conclusion from cross-validation results:

- bags_of_sifts are stronger features than tiny_images.
- SVM is stronger approach than KNN.
- For SVM on bags_of_sifts, accuracy is almost proportional to vocab_size (up to vocab_size = 1k).

4.5. Results for Validation with Different Vocabulary Sizes and Different SVM Lambda Values

Table 7: Results for Validation with Different Vocabulary Sizes and Different SVM Lambda Values

Validation results		**Vocab size**							
	Lambda	**10**	**20**	**50**	**100**	**200**	**400**	**1000**	**10,000**
KNN on tiny_images	**NA**	0.21							
KNN on bags_of_sifts	**NA**	0.40	0.47	0.48	0.49	**0.52**	0.51	0.50	0.43
SVM on bags_of_sifts	**0.1**	0.25	0.41	0.47	0.53	0.60	0.60	0.63	0.61
	0.5	0.29	0.42	0.50	0.56	0.60	0.61	0.65	0.62
	1	0.33	0.45	0.51	0.58	0.61	0.65	0.67	0.66
	3	0.40	0.48	0.54	0.60	0.66	0.67	0.70	0.67
	5	0.31	0.47	0.55	0.62	0.64	0.65	0.68	0.68
	7	0.38	0.49	0.56	0.63	0.64	0.64	0.68	0.68
	Average	**0.33**	**0.45**	**0.52**	**0.59**	**0.63**	**0.64**	**0.67**	**0.65**

Table 8: Validation Results: KNN on tiny_images & bags_of_sifts | Prediction Accuracy vs. Vocab Size

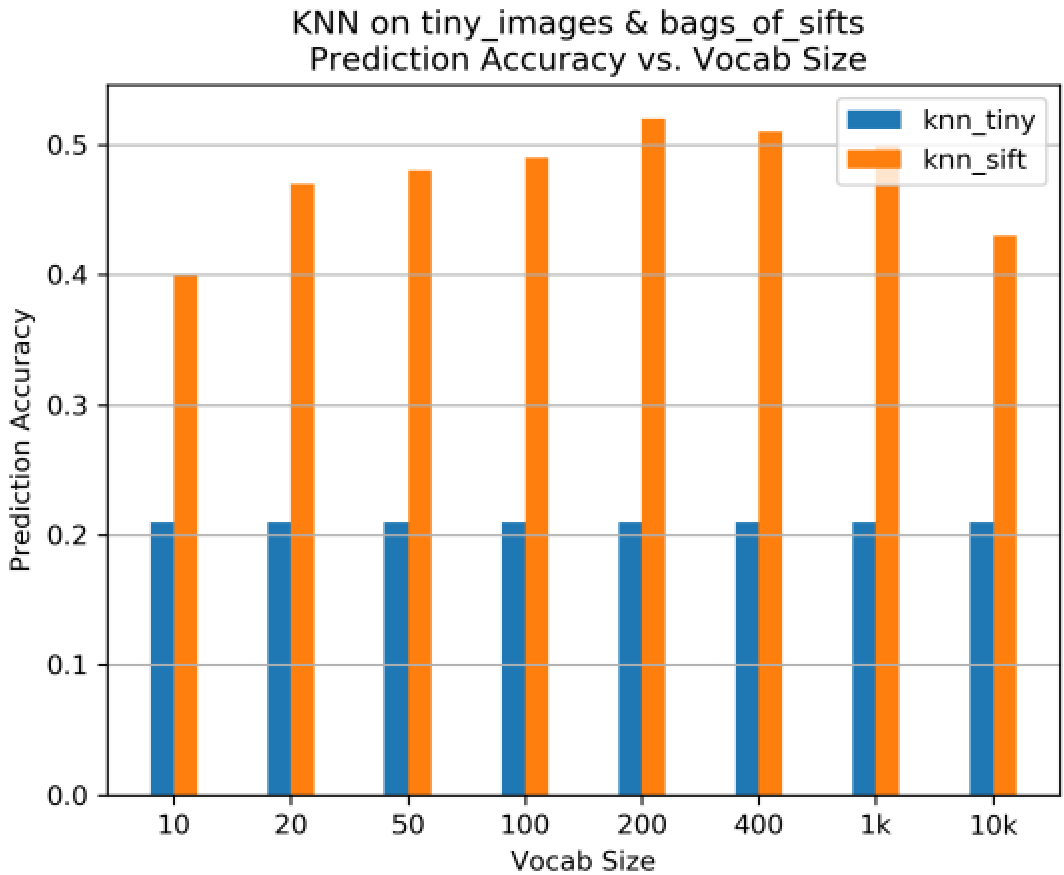
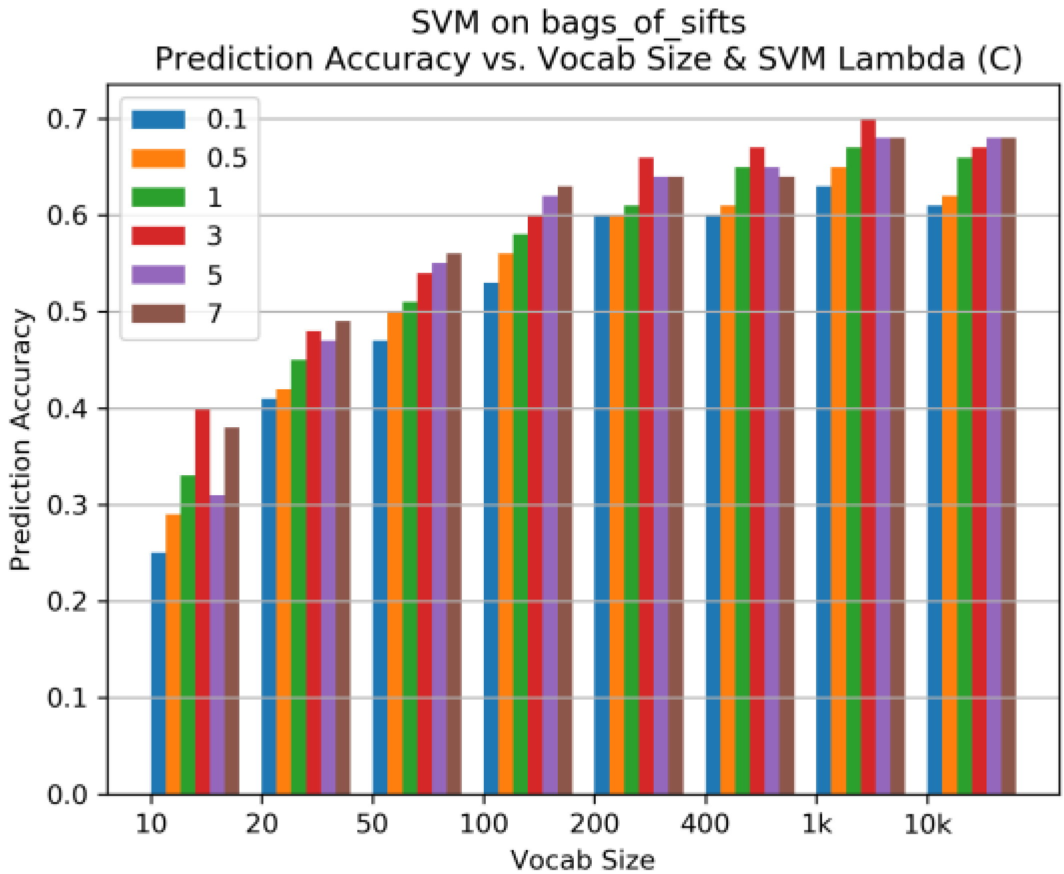


Table 9: Validation Results: SVM on bags_of_sifts | Prediction Accuracy vs. Vocab Size & SVM Lambda (C)



Remarks on the validation results:

Experiment design summary (rewritten here for convenience of reader):

- Number of experiments: 1
- Dataset:
  - Train dataset: 1050 images from original train dataset (70 images per class)
  - validation dataset: 450 images from original train dataset (30 images per class)
- Varying parameters:
  - vocab_zize = [10, 20, 50, 100, 200, 400, 1000, 10000]
  - svm_lambda = [0.1, 0.5, 1, 3, 5, 7]
- Approaches:
  - KNN on tiny_images
  - KNN on bags_of_sifts
  - SVM on bags_of_sifts
- Total number of experiments: exp_num = 1 * (1 + 8 + 8*6) = 57
- Results:

- Accuracy per approach, per vocab_size, per svm_lambda

Accuracy vs. approach:

- It is worth mentioning that for KNN on tiny_images, accuracy is a single value, independent of vocab_size (all blue bars in the plot are identical).
- As expected, the average accuracy for KNN on tiny_images is the least value (21%);
- Next accuracy is obtained by KNN on bags_of_sifts (average: 48%).
- The highest accuracy is obtained by SVM on bags_of_sifts when using appropriate vocab_size (accuracy from 33% (vocab_size = 10) to 67% (vocab_size =1000)).

Accuracy vs. vocab_size:

- When using bags_of_sifts features, the accuracy might also depend on vocab_size:
- For SVM on bags_of_sifts, validation accuracy is increasing proportional to vocab_size until vocab_size = 1k; and then slightly decreases for vocab_size = 10k.
- For KNN on bags_of_sifts, validation accuracy is also related to vocab_size but not as significant as SVM. The accuracy increases with vocab_size until reaches its maximum (52%) at vocab_size = 200 and then slowly decreases.

Accuracy vs. SVM lambda:

- When using SVM on bags_of_sifts, the accuracy might also depend on lambda:
- For most cases (vocab_sizes), validation accuracy is increasing proportional to lambda.
- For those cases where validation accuracy is not increasing proportional to lambda, it is mostly maximum at lambda = 3

Overall conclusion from validation results:

- bags_of_sifts are stronger features than tiny_images.
- SVM is stronger approach than KNN.
- For KNN on bags_of_sifts, accuracy is maximum for vocab_size = 200.
- For SVM on bags_of_sifts, accuracy is almost proportional to vocab_size (up to vocab_size = 1k).
- SVM accuracy is maximum with lambda = 5 for smaller vocab_size and lambda = 3 for higher vocab_size.

Extra Works

Following functions and code were done outside student_code.py predefined functions and proj4.ipynb:

- Function show_results_R1 saved in student_code.py
- Experiment code for cross validation and validation saved in proj4_experiment.ipynb

References

[1] Assignment 04 description by Dr. Kin-Choong Yow

[2] Szeliski, R. (2010). Computer vision: algorithms and applications. Springer Science & Business Media.