

CSE527 Homework 3

Due date: 23:59 on Oct 22, 2019 (Tuesday)

In this semester, we will use Google Colab for the assignments, which allows us to utilize resources that some of us might not have in their local machines such as GPUs. You will need to use your Stony Brook (*.stonybrook.edu) account for coding and Google Drive to save your results.

Google Colab Tutorial

Go to <https://colab.research.google.com/notebooks/>, you will see a tutorial named "Welcome to Colaboratory" file, where you can learn the basics of using google colab.

Settings used for assignments: *Edit -> Notebook Settings -> Runtime Type (Python 3)*.

Using SIFT in OpenCV 3.x.x in Colab

The default version of OpenCV in Colab is 3.4.3. If we use SIFT method directly, typically we will get this error message:

```
error: OpenCV(3.4.3) /io/opencv_contrib/modules/xfeatures2d/src/sift.cpp:1207: error: (-213:The function/feature is not implemented) This algorithm is patented and is excluded in this configuration; Set OPENCV_ENABLE_NONFREE CMake option and rebuild the library in function 'create'
```

One simple way to use the OpenCV in-built function SIFT in Colab is to switch the version to the one from 'contrib'. Below is an example of switching OpenCV version:

- 1. Run the following command in one section in Colab, which has already been included in this assignment:

```
pip install opencv-contrib-python==3.4.2.16
```

- 2. Restart runtime by

```
Runtime -> Restart Runtime
```

Then you should be able to use use `cv2.xfeatures2d.SIFT_create()` to create a SIFT object, whose functions are listed at http://docs.opencv.org/3.0-beta/modules/xfeatures2d/doc/nonfree_features.html

Some Resources

In addition to the tutorial document, the following resources can definitely help you in this homework:

- http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html
- http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html
- http://docs.opencv.org/3.0-beta/modules/xfeatures2d/doc/nonfree_features.html?highlight=sift#cv2.SIFT
- http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html

Description

In this homework, we will examine the task of scene recognition starting with very simple methods: tiny images and nearest neighbor classification, and then move on to more advanced methods: bags of quantized local features and linear classifiers learned by support vector machines.

Bag of words models are a popular technique for image classification inspired by models used in natural language processing. The model ignores or downplays word arrangement (spatial information in the image) and classifies based on a histogram of the frequency of visual words. The visual word "vocabulary" is established by clustering a large corpus of local features. See Szeliski chapter 14.4.1 for more details on category recognition with quantized features. In addition, 14.3.2 discusses vocabulary creation and 14.1 covers classification techniques.

For this homework you will be implementing a basic bag of words model. You will classify scenes into one of 16 categories by training and testing on the 16 scene database (introduced in [Lazebnik et al. 2006](#), although built on top of previously published datasets). [Lazebnik et al. 2006](#) is a great paper to read, although we will be implementing the baseline method the paper discusses (equivalent to the zero level pyramid) and not the more sophisticated spatial pyramid. For an excellent survey of pre-deep-learning feature encoding methods for bag of words models, see [Chatfield et al, 2011](#).

You are required to implement 2 different image representations: tiny images and bags of SIFT features, and 2 different classification techniques: nearest neighbor and linear SVM. There are 3 problems plus a performance report in this homework with a total of 100 points. 1 bonus question with extra 10 points is provided under problem 3. The maximum points you may earn from this homework is 100 + 10 = 110 points. Be sure to read **Submission Guidelines** below. They are important.

Dataset

The starter code trains on 150 and tests on 50 images from each category (i.e. 2400 training examples total and 800 test cases total). In a real research paper, one would be expected to test performance on random splits of the data into training and test sets, but the starter code does not do this to ease debugging.

Save the [dataset\(click me\)](#) into your working folder in your Google Drive for this homework. Under your root folder, there should be a folder named "data" (i.e. XXX/Surname_Givenname_SBUID/data) containing the images. **Do not upload** the data subfolder before submitting on blackboard due to size limit. There should be only one .ipynb file under your root folder Surname_Givenname_SBUID.

Starter Code

To make your task a little easier, below we provide some starter code which randomly guesses the category of every test image and achieves about 6.25% accuracy (1 out of 16 guesses is correct).

```
pip install opencv-contrib-python==3.4.2.16

❏ Requirement already satisfied: opencv-contrib-python==3.4.2.16 in /usr/local/lib/python3.6/dist-packages (3.4.2.16)
  Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packages (from opencv-contrib-python==3.4.2.16) (1.17.3)

# import packages here
import cv2
import numpy as np
import matplotlib.pyplot as plt
import glob
import itertools
import time
import zipfile
import torch
import torchvision
import gc
import pickle
from sklearn import svm
from skimage import color
from skimage import io
from torch.utils.data import Dataset, DataLoader
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import accuracy_score
```

```
print(cv2.__version__) # verify OpenCV version
```

```
3.4.3
```

```
# Mount your google drive where you've saved your assignment folder
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
%pwd
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&s
Enter your authorization code:
.....
Mounted at /content/gdrive
'/content'
```

```
# Set your working directory (in your google drive)
# Note that 'gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/hw2' is just an example,
# change it to your specific homework directory.
cd '/content/gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/Paratkar_Shreyash_112673930_hw3'
```

```
/content/gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/Paratkar_Shreyash_112673930_hw3
```

```
%pwd
%ls -lrt
```

```
total 522412
dr-x----- 2 root root      4096 Oct  4 16:38 data/
-rw----- 1 root root       127 Oct 21 16:20 sample.txt
-rw----- 1 root root 198806830 Oct 24 20:39 train.pkl
-rw----- 1 root root 33075269 Oct 24 20:39 test.pkl
-rw----- 1 root root 136559864 Oct 24 20:47 train_des_flattened_all_imgs.pkl
-rw----- 1 root root 123020990 Oct 24 20:47 train_des_all_imgs.pkl
-rw----- 1 root root 22759464 Oct 24 20:47 test_des_flattened_all_imgs.pkl
-rw----- 1 root root 20502986 Oct 24 20:47 test_des_all_imgs.pkl
-rw----- 1 root root   218079 Oct 24 20:49 Paratkar_Shreyash_112673930_hw3.ipynb
```

Data Preparation

```
class_names = [name[13:] for name in glob.glob('./data/train/*')]
class_names = dict(zip(range(len(class_names)), class_names))
print("class_names: %s " % class_names)
n_train_samples_per_class = 150
n_test_samples_per_class = 50
```

```
def load_dataset(path, num_per_class=-1):
    data = []
    labels = []
```

```
labels = []
for id, class_name in class_names.items():
    print("Loading images from class: %s" % id)
    img_path_class = glob.glob(path + class_name + '/*.jpg')
    if num_per_class > 0:
        img_path_class = img_path_class[:num_per_class]
    labels.extend([id]*len(img_path_class))
    for filename in img_path_class:
        data.append(cv2.imread(filename, 0))
return data, labels

# load training dataset
# train_data, train_label = load_dataset('./data/train/')
train_data, train_label = load_dataset('./data/train/', n_train_samples_per_class)
n_train = len(train_label)
print("n_train: %s" % n_train)

# load testing dataset
# test_data, test_label = load_dataset('./data/test/')
test_data, test_label = load_dataset('./data/test/', n_test_samples_per_class)
n_test = len(test_label)
print("n_test: %s" % n_test)
```



```
class_names: {0: 'Forest', 1: 'Industrial', 2: 'Flower', 3: 'Coast', 4: 'InsideCity', 5: 'Office', 6: 'Bedroom', 7: 'Highway', 8: 'Street', 9: 'TallBuilding', 10: 'LivingRoom', 11: 'Suburb', 12: 'OpenCoun
Loading images from class: 0
Loading images from class: 1
Loading images from class: 2
Loading images from class: 3
Loading images from class: 4
Loading images from class: 5
Loading images from class: 6
Loading images from class: 7
Loading images from class: 8
Loading images from class: 9
Loading images from class: 10
Loading images from class: 11
Loading images from class: 12
Loading images from class: 13
Loading images from class: 14
Loading images from class: 15
n_train: 2400
Loading images from class: 0
Loading images from class: 1
Loading images from class: 2
Loading images from class: 3
Loading images from class: 4
Loading images from class: 5
Loading images from class: 6
Loading images from class: 7
Loading images from class: 8
Loading images from class: 9
Loading images from class: 10
Loading images from class: 11
Loading images from class: 12
Loading images from class: 13
Loading images from class: 14
Loading images from class: 15
n_test: 400
```

```
# As loading the data from the source for the first time is time consuming, so you can pickle or save the data in a compact way such that subsequent data loading is faster
```

```
# Save intermediate image data into disk
```

```
file = open('train.pkl','wb')
pickle.dump(train_data, file)
pickle.dump(train_label, file)
file.close()
```

```
file = open('test.pkl','wb')
pickle.dump(test_data, file)
pickle.dump(test_label, file)
file.close()
```

```
# Load intermediate image data from disk
```

```
file = open('train.pkl', 'rb')
train_data = pickle.load(file)
train_label = pickle.load(file)
file.close()
```

```
file = open('test.pkl', 'rb')
test_data = pickle.load(file)
test_label = pickle.load(file)
file.close()

print(len(train_data), len(train_label)) # Verify number of training samples
print(len(test_data), len(test_label))   # Verify number of testing samples

# 2400 2400
# 400 400

# plt.imshow(train_data[1], cmap='gray') # Verify image
img_new_size = (240, 240)

train_data = list(map(lambda x: cv2.resize(x, img_new_size), train_data))
train_data = np.stack(train_data)
train_label = np.array(train_label)

test_data = list(map(lambda x: cv2.resize(x, img_new_size), test_data))
test_data = np.stack(test_data)
test_label = np.array(test_label)

# Verify image
# plt.imshow(cv2.resize(train_data[1], img_new_size), cmap='gray')
# print(train_data[0].dtype)

n_train = len(train_label)
n_test = len(test_label)

# feature extraction
def extract_feat(raw_data):
    print(len(raw_data))
    feat_dim = 1000
    feat = np.zeros((len(raw_data), feat_dim), dtype=np.float32)
    for i in np.arange(feat.shape[0]):
        feat[i] = np.reshape(raw_data[i], (raw_data[i].size))[:feat_dim] # dummy implemtation
    print("feat",len(feat))

    return feat

train_feat = extract_feat(train_data)
test_feat = extract_feat(test_data)

# model training: take feature and label, return model
def train(X, Y):
    return 0 # dummy implementation

# prediction: take feature and model, return label
def predict(model, x):
    return np.random.randint(16) # dummy implementation

# evaluation
```

```
# evaluation
predictions = [-1]*len(test_feat)
for i in np.arange(n_test):
    predictions[i] = predict(None, test_feat[i])

accuracy = sum(np.array(predictions) == test_label) / float(n_test)

print("The accuracy of my dummy model is {:.2f}%".format(accuracy*100))
```

```
2400
feat 2400
400
feat 400
The accuracy of my dummy model is 5.75%
```

Problem 1: Tiny Image Representation + Nearest Neighbor Classifier

{25 points} You will start by implementing the tiny image representation and the nearest neighbor classifier. They are easy to understand, easy to implement, and run very quickly for our experimental setup.

The "tiny image" feature is one of the simplest possible image representations. One simply resizes each image to a small, fixed resolution. You are required to **resize the image to 16x16**. It works slightly better if the tiny image is made to have zero mean and unit length (normalization). This is not a particularly good representation, because it discards all of the high frequency image content and is not especially invariant to spatial or brightness shifts. We are using tiny images simply as a baseline.

The nearest neighbor classifier is equally simple to understand. When tasked with classifying a test feature into a particular category, one simply finds the "nearest" training example (L2 distance is a sufficient metric) and assigns the label of that nearest training example to the test example. The nearest neighbor classifier has many desirable features — it requires no training, it can learn arbitrarily complex decision boundaries, and it trivially supports multiclass problems. It is quite vulnerable to training noise, though, which can be alleviated by voting based on the K nearest neighbors (but you are not required to do so). Nearest neighbor classifiers also suffer as the feature dimensionality increases, because the classifier has no mechanism to learn which dimensions are irrelevant for the decision.

Report your classification accuracy on the test sets and time consumption.

Hints:

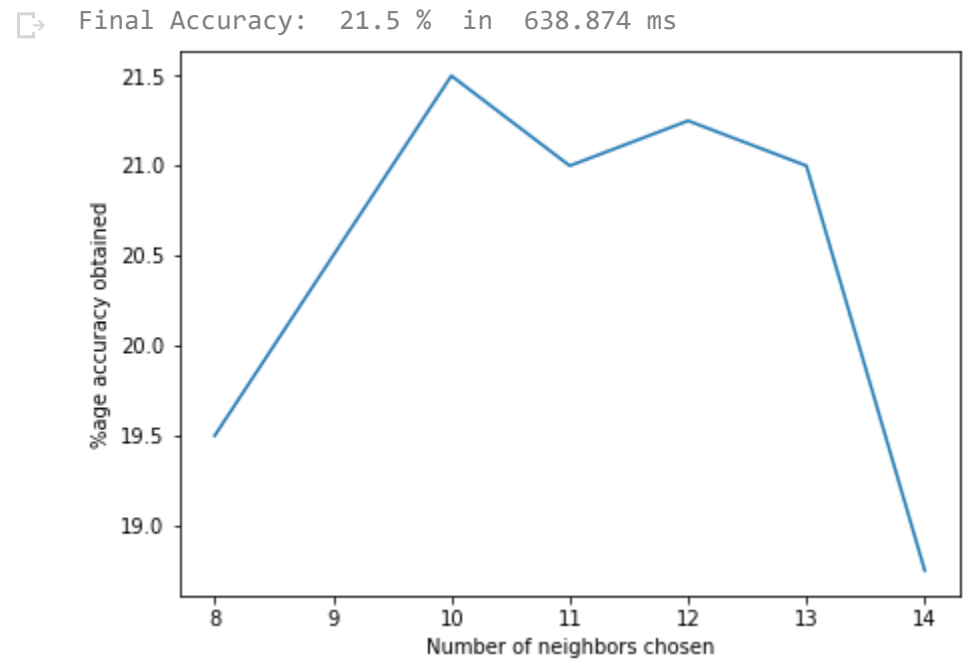
- Use [cv2.resize\(\)](#) to resize the images;
- Use [NearestNeighbors in Sklearn](#) as your nearest neighbor classifier.

```
# Write your codes here
from sklearn.metrics import accuracy_score
import datetime
start_time = datetime.datetime.now()
img_size_16 = (16, 16)
train_data_16x16 = list(map(lambda x: cv2.resize(x, img_size_16), train_data))
train_data_16x16 = list(map(lambda x: (x - x.mean()) / (x.max() - x.min()), train_data_16x16))
train_data_16x16 = list(map(lambda x: x.flatten(), train_data_16x16))
train_data_16x16 = np.stack(train_data_16x16)
test_data_16x16 = list(map(lambda x: cv2.resize(x, img_size_16), test_data))
test_data_16x16 = list(map(lambda x: (x - x.mean()) / (x.max() - x.min()), test_data_16x16))
test_data_16x16 = list(map(lambda x: x.flatten(), test_data_16x16))
test_data_16x16 = np.stack(test_data_16x16)
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(train_data_16x16, np.array(train_label))
pred1 = knn.predict(test_data_16x16)
label1 = np.array(test_label)
```

```
end_time = datetime.datetime.now()
knn_duration = end_time - start_time
print('Final Accuracy: ', accuracy_score(pred1, label1)*100, '%', ' in ', knn_duration.microseconds/1000, 'ms')
```

```
n_neighbors = [8, 9, 10, 11, 12, 13, 14]
accuracy_obtained = [19.5, 20.5, 21.5, 21, 21.25, 21, 18.75]
```

```
plt.subplots(figsize = (7,5))
plt.plot(n_neighbors, accuracy_obtained)
plt.xlabel("Number of neighbors chosen")
plt.ylabel("%age accuracy obtained")
plt.show()
```



References:

- <https://ashokharnal.wordpress.com/tag/kneighborsclassifier-explained/>
- <https://stats.stackexchange.com/questions/349842/why-do-we-need-to-fit-a-k-nearest-neighbors-classifier>

Problem 2: Bag of SIFT Representation + Nearest Neighbor Classifier

{35 points} After you have implemented a baseline scene recognition pipeline it is time to move on to a more sophisticated image representation — bags of quantized SIFT features. Before we can represent our training and testing images as bag of feature histograms, we first need to establish a vocabulary of visual words. We will form this vocabulary by sampling many local features from our training set (10's or 100's of thousands) and then cluster them with k-means. The number of k-means clusters is the size of our vocabulary and the size of our features. For example, you might start by clustering many SIFT descriptors into k=50 clusters. This partitions the continuous, 128 dimensional SIFT feature space into 50 regions. For any new SIFT feature we observe, we can figure out which region it belongs to as long as we save the centroids of our original clusters. Those centroids are our visual word vocabulary. Because it can be slow to sample and cluster many local features, the starter code saves the cluster centroids and avoids recomputing them on future runs.

Now we are ready to represent our training and testing images as histograms of visual words. For each image we will densely sample many SIFT descriptors. Instead of storing hundreds of SIFT descriptors, we simply count how many SIFT descriptors fall into each cluster in our visual word vocabulary. This is done by finding the nearest neighbor k-means centroid for every SIFT feature. Thus, if we have a vocabulary of 50 visual words, and we detect 220 distinct SIFT features in an image, our bag of SIFT representation will be a histogram of 50 dimensions where each bin counts how many times a SIFT descriptor was assigned to that cluster. The total of all the bin-counts is 220. The histogram should be normalized so that image size does not dramatically change the bag of features magnitude.

After you obtain the Bag of SIFT feature representation of the images, you have to train a KNN classifier in the Bag of SIFT feature space and report your test set accuracy and time consumption.

Note:

- Instead of using SIFT to detect invariant keypoints which is time-consuming, you are recommended to densely sample keypoints in a grid with certain step size (sampling density) and scale.
- There are many design decisions and free parameters for the bag of SIFT representation (number of clusters, sampling density, sampling scales, SIFT parameters, etc.) so accuracy might vary from 50% to 60%.
- Indicate clearly the parameters you use along with the prediction accuracy on test set and time consumption.

Hints:

- Use [KMeans in Sklearn](#) to do clustering and find the nearest cluster centroid for each SIFT feature;
- Use `cv2.xfeatures2d.SIFT_create()` to create a SIFT object;
- Use `sift.compute()` to compute SIFT descriptors given densely sampled keypoints ([cv2.Keypoint](#)).
- Be mindful of RAM usage. Try to make the code more memory efficient, otherwise it could easily exceed RAM limits in Colab, at which point your session will crash.
- If your RAM is going to run out of space, use [gc.collect\(\)](#) for the garbage collector to collect unused objects in memory to free some space.
- Store data or features as NumPy arrays instead of lists. Computation on NumPy arrays is much more efficient than lists.

```
# Write your codes here
siftDetector = cv2.xfeatures2d.SIFT_create()
step_size = 25

def get_all_descriptors(data):
    des_flattened_all_imgs = []
    des_all_imgs = []
    data_copy = data
    for image in data_copy:
        img_keypoints = []
        img_rows = image.shape[0]
        img_cols = image.shape[1]
        for img_x in range(0,img_rows,step_size):
            for img_y in range(0,img_cols,step_size):
                key_point = cv2.KeyPoint(img_x,img_y,step_size)
                img_keypoints.append(key_point)
        kp, des = siftDetector.compute(image,np.array(img_keypoints))
        des_all_imgs.append(des)
        for des_flattened in des:
            des_flattened_all_imgs.append(des_flattened)
    return des_flattened_all_imgs, des_all_imgs

train_des_flattened_all_imgs = []
train_des_all_imgs = []
train_des_flattened_all_imgs, train_des_all_imgs = get_all_descriptors(train_data)

test_des_flattened_all_imgs = []
test_des_all_imgs = []
test_des_flattened_all_imgs, test_des_all_imgs = get_all_descriptors(test_data)

print(len(train_des_flattened_all_imgs), len(train_des_all_imgs))
print(len(test des flattened allimgs), len(test des all imgs))
```

```
file = open('train_des_flattened_all_imgs.pkl','wb')
pickle.dump(train_des_flattened_all_imgs, file)
file.close()
```

```
file = open('train_des_all_imgs.pkl','wb')
pickle.dump(train_des_all_imgs, file)
file.close()
```

```
file = open('test_des_flattened_all_imgs.pkl','wb')
pickle.dump(test_des_flattened_all_imgs, file)
file.close()
```

```
file = open('test_des_all_imgs.pkl','wb')
pickle.dump(test_des_all_imgs, file)
file.close()
```

```
file = open('train_des_flattened_all_imgs.pkl', 'rb')
train_des_flattened_all_imgs = pickle.load(file)
file.close()
file = open('train_des_all_imgs.pkl', 'rb')
train_des_all_imgs = pickle.load(file)
file.close()
file = open('test_des_flattened_all_imgs.pkl', 'rb')
test_des_flattened_all_imgs = pickle.load(file)
file.close()
file = open('test_des_all_imgs.pkl', 'rb')
test_des_all_imgs = pickle.load(file)
file.close()
print(len(train_des_flattened_all_imgs), len(train_des_all_imgs))
print(len(test_des_flattened_all_imgs), len(test_des_all_imgs))
```

```
240000 2400
40000 400
```

```
import datetime
start_time = datetime.datetime.now()
kmeans = KMeans(n_clusters=60).fit(train_des_flattened_all_imgs)
end_time = datetime.datetime.now()
print('Time taken for fitting Kmeans = ', (end_time - start_time).microseconds/1000)
```

```
Time taken for fitting Kmeans = 584.704
```

```
import datetime
start_time = datetime.datetime.now()
hist_bins = np.arange(1,61)
def convert_img_to_hist(des_all_imgs):
    img_hists = []
    for img_des in des_all_imgs:
        clusters = kmeans.predict(img_des)
        img_hists.append(np.histogram(clusters, hist_bins)[0])
    return img_hists
```

```
        return img_nlists

train_histograms = convert_img_to_hist(train_des_all_imgs)
test_histograms = convert_img_to_hist(test_des_all_imgs)

end_time = datetime.datetime.now()
print('Time taken to build histograms of images = ', (end_time - start_time).microseconds/1000, 'ms')

file = open('train_histograms.pkl','wb')
pickle.dump(train_histograms, file)
file.close()

file = open('test_histograms.pkl','wb')
pickle.dump(test_histograms, file)
file.close()
```

```
➤ Time taken to build histograms of images = 121.681 ms
```

```
file = open('train_histograms.pkl', 'rb')
train_histograms = pickle.load(file)
file.close()
file = open('test_histograms.pkl', 'rb')
test_histograms = pickle.load(file)
file.close()
print(len(train_histograms))
print(len(test_histograms))
```

```
➤ 2400
   400
```

```
import datetime
start_time = datetime.datetime.now()

knn = KNeighborsClassifier(n_neighbors=25)
knn.fit(train_histograms, np.array(train_label))
```

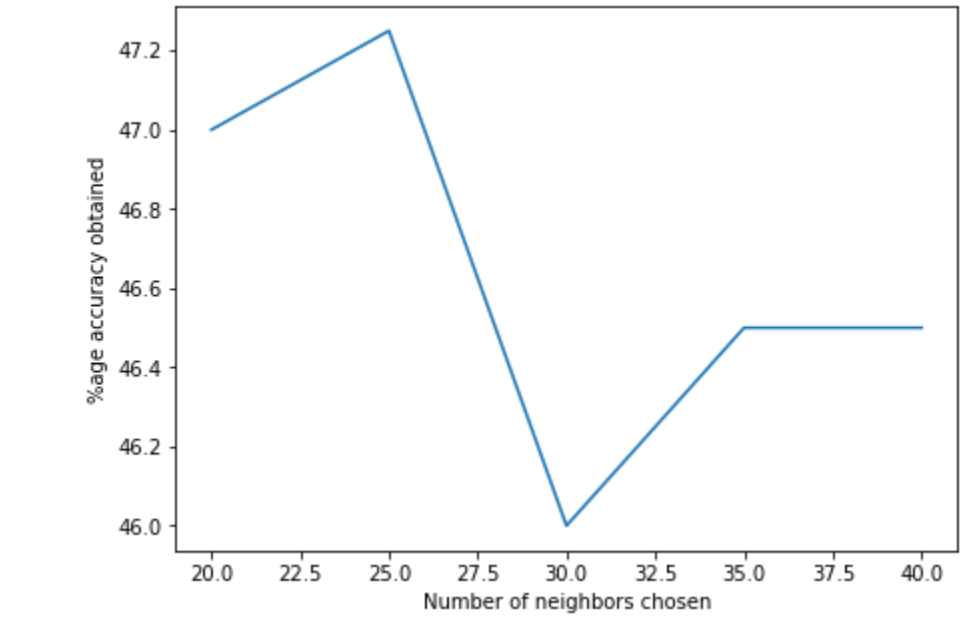
```
pred2 = knn.predict(test_histograms)
label2 = np.array(test_label)
```

```
end_time = datetime.datetime.now()
print('Time taken for classification and prediction = ', (end_time - start_time).microseconds/1000, 'ms')
print('Accuracy: ', accuracy_score(pred2, label2)*100, '%', ' in ', (end_time - start_time).microseconds/1000, 'ms')
n_neighbors = [20, 25, 30, 35, 40]
accuracy_obtained = [47, 47.25, 46, 46.5, 46.5]
```

```
plt.subplots(figsize = (7,5))
plt.plot(n_neighbors, accuracy_obtained)
plt.xlabel("Number of neighbors chosen")
plt.ylabel("%age accuracy obtained")
plt.show()
```

```
➤
```

Time taken for classification and prediction = 128.539 ms
Accuracy: 47.25 % in 128.539 ms



References:

<https://towardsdatascience.com/bag-of-visual-words-in-a-nutshell-9ceea97ce0fb>

Problem 3.a: Bag of SIFT Representation + one-vs-all SVMs

{15 points} The last task is to train one-vs-all linear SVMs to operate in the bag of SIFT feature space. Linear classifiers are one of the simplest possible learning models. The feature space is partitioned by a learned hyperplane and test cases are categorized based on which side of that hyperplane they fall on. Despite this model being far less expressive than the nearest neighbor classifier, it will often perform better.

You do not have to implement the support vector machine. However, linear classifiers are inherently binary and we have a 16-way classification problem (the library has handled it for you). To decide which of 16 categories a test case belongs to, you will train 16 binary, one-vs-all SVMs. One-vs-all means that each classifier will be trained to recognize 'forest' vs 'non-forest', 'kitchen' vs 'non-kitchen', etc. All 16 classifiers will be evaluated on each test case and the classifier which is most confidently positive "wins". E.g. if the 'kitchen' classifier returns a score of -0.2 (where 0 is on the decision boundary), and the 'forest' classifier returns a score of -0.3, and all of the other classifiers are even more negative, the test case would be classified as a kitchen even though none of the classifiers put the test case on the positive side of the decision boundary. When learning an SVM, you have a free parameter λ (lambda) which controls how strongly regularized the model is. Your accuracy will be very sensitive to λ , so be sure to try many values.

Indicate clearly the parameters you use along with the prediction accuracy on test set and time consumption.

Bonus {10 points}: For this question, you need to generate class prediction for the images in **test2** folder using your best model. The prediction file(**Surname_Givenname_SBUID_Pred.txt**) should follow the exact format as given in the **sample.txt** file.10 points will be given to students whose accuracy ranks top 3 in this homework.

Hints:

- Use SVM in [Sklearn](#) (recommended) or [OpenCV](#) to do training and prediction.

```
# Write your codes here
import warnings
warnings.filterwarnings('ignore')
from sklearn.svm import LinearSVC
import datetime
```

```
start_time = datetime.datetime.now()

linearSVCClassifier = LinearSVC(C=0.2)
linearSVCClassifier.fit(train_histograms, np.array(train_label))

label3 = np.array(test_label)
pred3 = linearSVCClassifier.predict(test_histograms)

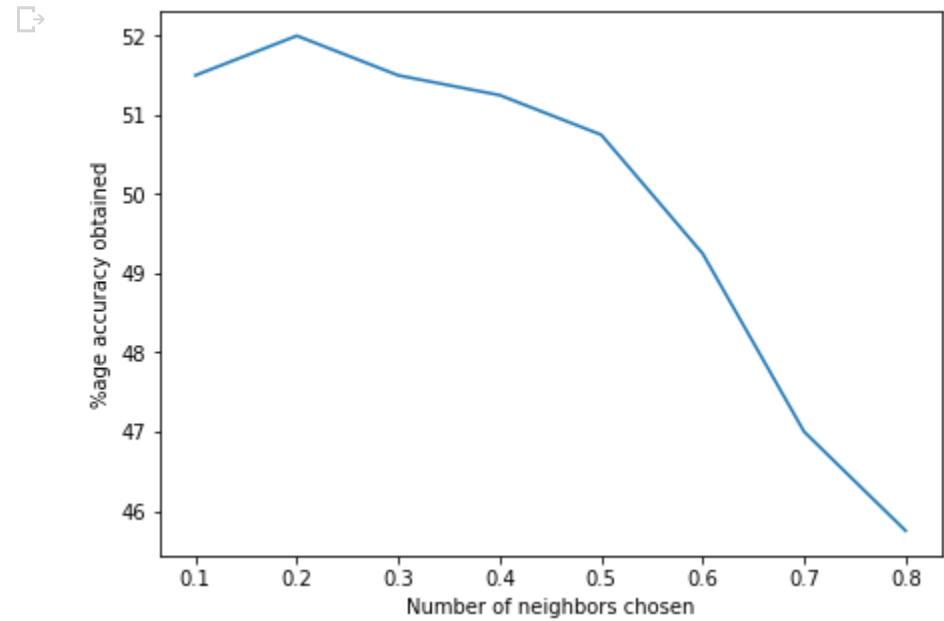
end_time = datetime.datetime.now()
knn_duration = end_time - start_time

print('Accuracy: ', accuracy_score(pred3, label3)*100, '%', ' in ', knn_duration.microseconds/1000, 'ms')
```

➤ Accuracy: 52.0 % in 182.307 ms

```
c_vals = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
accuracy_obtained = [51.5, 52, 51.5, 51.25, 50.75, 49.25, 47, 45.75]
```

```
plt.subplots(figsize = (7,5))
plt.plot(c_vals, accuracy_obtained)
plt.xlabel("Chosen value of regularity")
plt.ylabel("%age accuracy obtained")
plt.show()
```



▼ Problem 3.b

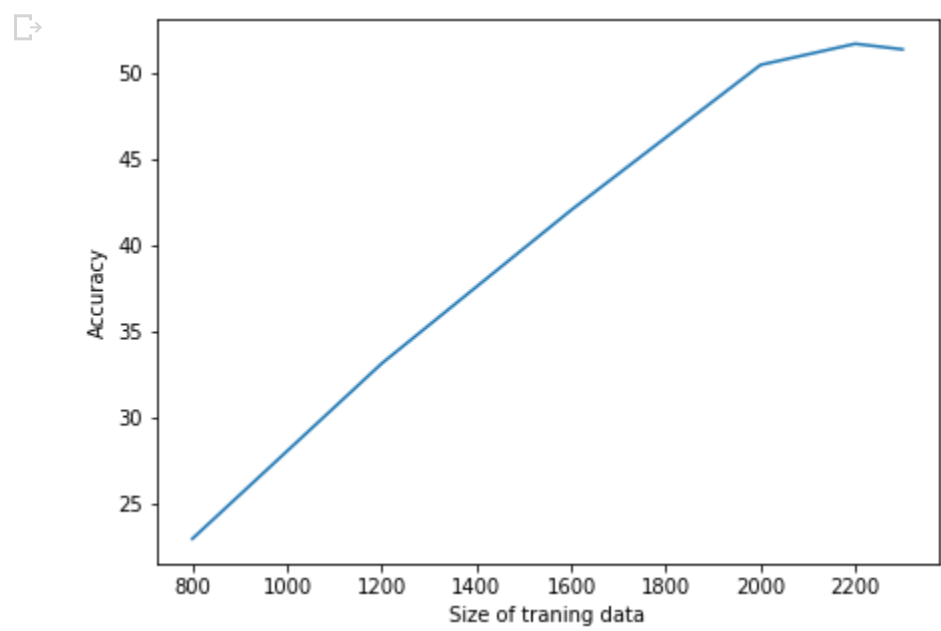
{5 points} Repeat the evaluation above for different sizes of training sets and draw a plot to show how the size of the training set affects test performace. Do this for training set sizes of 800, 1200, 1600, 2000, 2200, and 2300 images. Randomly sample the images from the original training set and evaluate accuracy. Repeat this process 10 times for each training set size and report the average prediction accuracy. How does performance variability change with training set size? How does performance change? Give reason for your observations.

```
# Write your codes here
from sklearn.utils import shuffle

training_sizes = [800,1200,1600,2000,2200,2300]
accuracy_scores = []

for training_size in training_sizes:
    sum_accuracy = 0
    for i in range(10):
        shuffle(train_histograms, np.array(train_label))
        linearSVCClassifier = LinearSVC(C = 0.2)
        linearSVCClassifier.fit(train_histograms[:training_size], np.array(train_label)[:training_size])
        pred = linearSVCClassifier.predict(test_histograms)
        sum_accuracy = sum_accuracy + accuracy_score(pred,np.array(test_label))*100
    accuracy_scores.append(sum_accuracy/10)

plt.subplots(figsize = (7,5))
plt.plot(training_sizes,accuracy_scores)
plt.xlabel('Size of traning data')
plt.ylabel('Accuracy')
plt.show()
```



▼ Performance Report

{20 points} Please report the performance of the following combinations **in the given order** in terms of the time consumed and classification accuracy. Describe your algorithm, any decisions you made to write your algorithm in your particular way, and how different choices you made affect it. Compute and draw a (normalized) [confusion matrix](#), and discuss where the method performs best and worse for each of the combination. Here is an [example](#) of how to compute confusion matrix.

- 1st: Tiny images representation and nearest neighbor classifier (accuracy of about 18-25%).
- 2nd: Bag of SIFT representation and nearest neighbor - classifier (accuracy of about 40-50%).
- 3rd: Bag of SIFT representation and linear SVM classifier (accuracy of about 50-70%).

```
from sklearn.metrics import confusion_matrix
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    fig, ax = plt.subplots(figsize=(8,8))
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
          yticks=np.arange(cm.shape[0]),
          # ... and label them with the respective list entries
          xticklabels=classes, yticklabels=classes,
          title=title,
          ylabel='True label',
          xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
              rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```

```
c_names = [name[13:] for name in glob.glob('./data/train/*')]
```

```
#First combination:
# Confusion matrix
cm1 = confusion_matrix(pred1, label1)
plt.figure(figsize=(12,12))
plot_confusion_matrix(cm1, c_names, normalize=True)
plt.show()
```

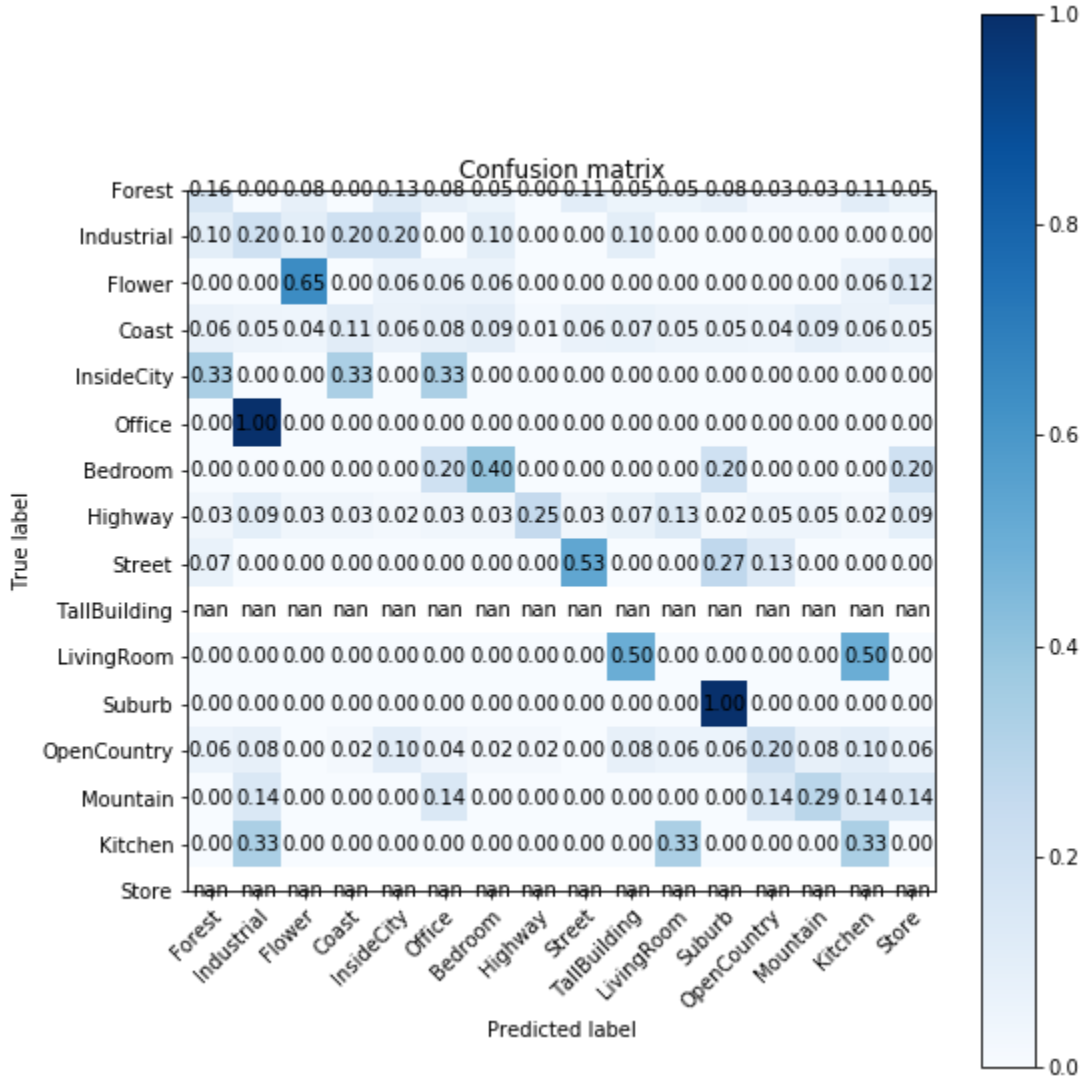
```
#Second combination:
# Confusion matrix
cm2 = confusion_matrix(pred2, label2)
```

```
plt.figure(figsize=(12,12))
plot_confusion_matrix(cm2, c_names, normalize=True)
plt.show()
```

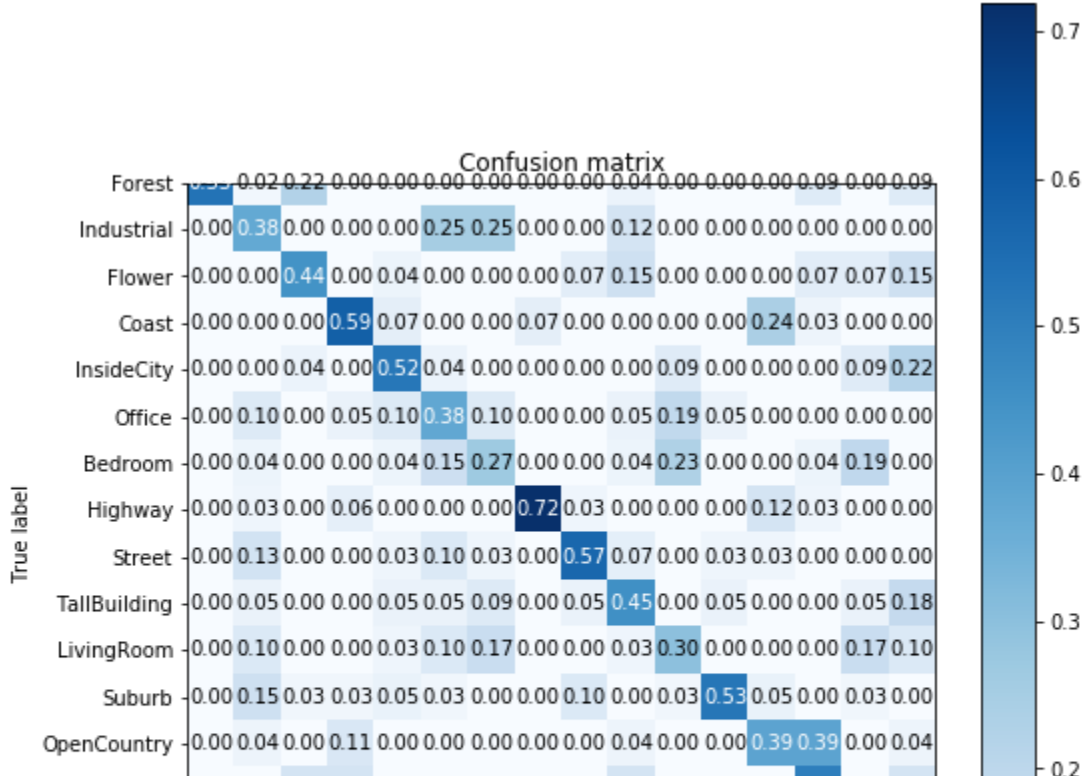
```
#Third combination:
# Confusion matrix
cm3 = confusion_matrix(pred3, label3)
plt.figure(figsize=(12,12))
plot_confusion_matrix(cm3, c_names, normalize=True)
plt.show()
```

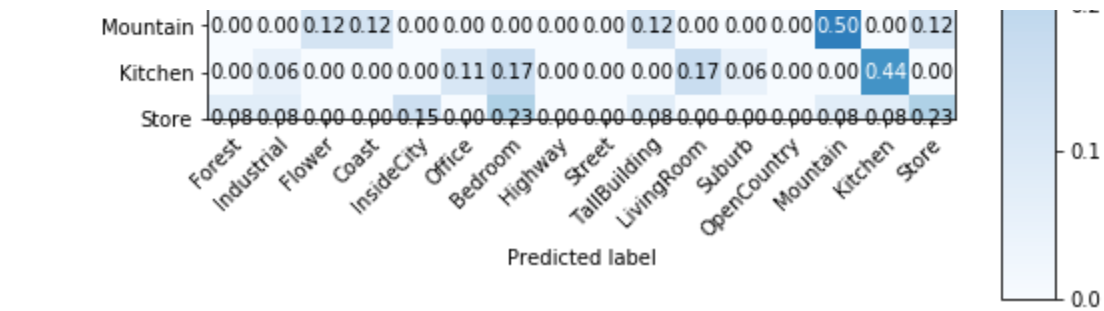


Normalized confusion matrix
<Figure size 864x864 with 0 Axes>

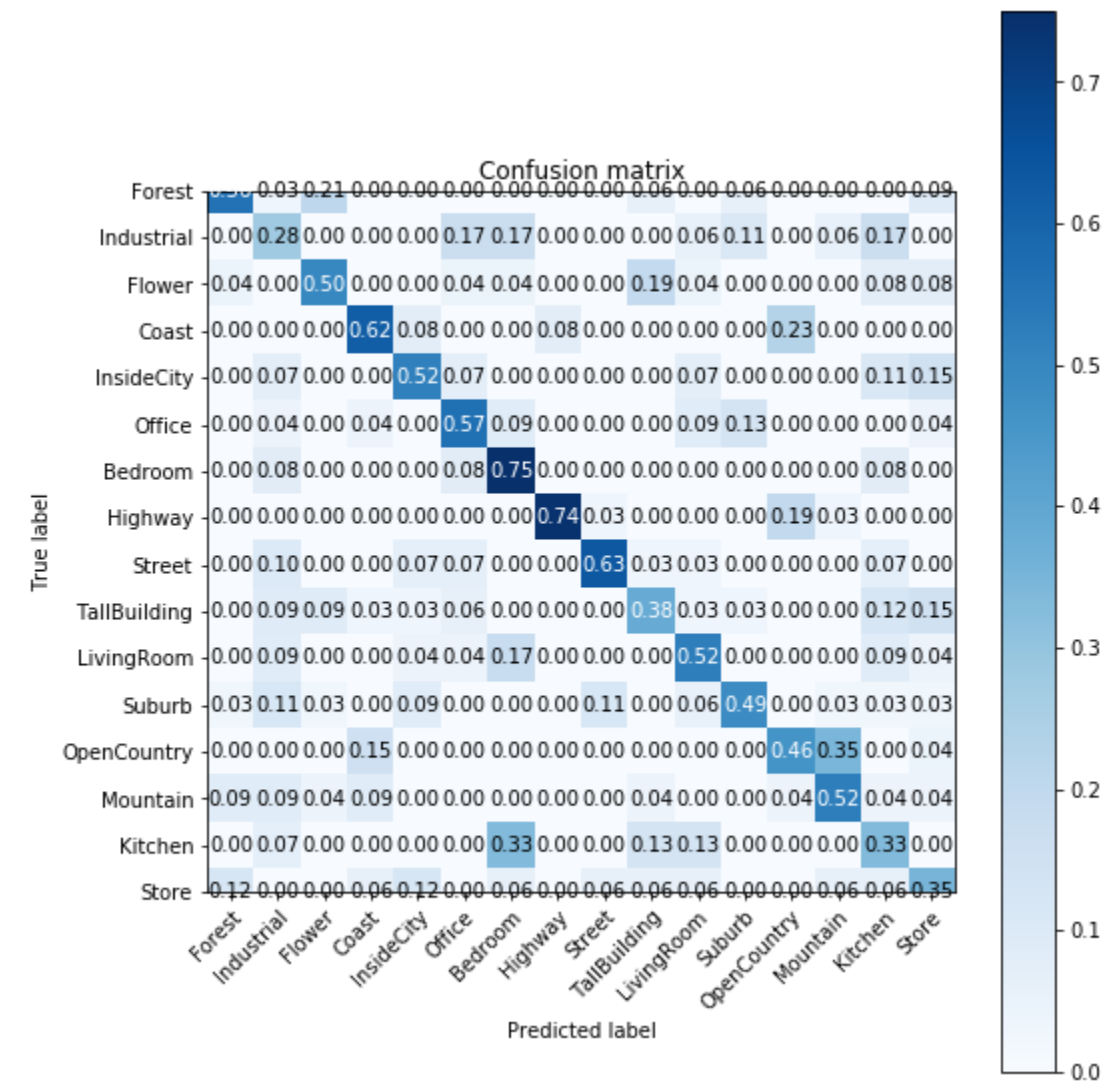


Normalized confusion matrix
<Figure size 864x864 with 0 Axes>





Normalized confusion matrix
<Figure size 864x864 with 0 Axes>



References:
https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

Q1] Tiny Image Representation + Nearest Neighbor Classifier

In the above algorithm, the below steps were taken:

- 1.) Resizing of training and testing images to 16 x 16 resolution
- 2.) Once the images were resized, normalization was carried out to have zero mean and unit variance on both the sets of images
- 3.) numpy.stack() function is used to join the sequence of images in the training set along a new axis
- 4.) A KNeighborsClassifier object is initialized for 'n_neighbors = 10' where n_neighbors is the number of neighbors that will vote for the class of the target point

5.) In the case of knn, the classifier is identified by the training data itself, the training data is scaled and normalized earlier is fit to get a classifier

6.) This classifier is used to predict the labels of test data

7.) After comparing the accuracies obtained using different number of neighbors, it was found out that the model works best for when it uses 10 neighbors for voting for a class. It gives an accuracy of 21.5 % in 603.409 ms.

8.) As we move the value of n_neighblrs around 10, we observe a dip in accuracy of the model in both the directions mainly because as the number of nearest neighbors increases, the model starts underfitting and as it decreases, it overfits.

Q2]

For the bag of visual words, (Bag of SIFT Representation + Nearest Neighbor Classifier)

1.) We compute descriptors for all the images in the training and testing data by selecting every 25th pixel from each image to compute the keypoints to get the descriptors.

2.) Then we cluster the descriptors obtained fitting Kmeans on training data and a cluster size of 60.

3.) Using the clusters obtained for all the images' descriptors, we compute the histogram for each image.

4.) This list of histogram is our bag of visual words

5.) Using the bag of visual words obtained for the training and testing data, we compute the nearest neighbors for each image using the histograms.

6.) The bag of visual words obtained from the testing data was used to predict the labels of testing images.

7.) The number of nearest neighbors chosen to fit the training bag of visual was set to 25 since, the accuracy of the predictions decreased around the mark of 25.

8.) Time taken for classification and prediction = 136.726 ms with an accuracy: 47.25 %

Q3a]

Bag of SIFT Representation + one-vs-all SVMs

1.) The bag of visual words obtained in the previous step were used again to be passed to a Linear Support Vector classifier.

2.) The accuracy of predictions obtained from the classifier peaks for a penalty parameter C value 0.2.

3.) Time taken for classification and prediction = 182.307 ms with an Accuracy: 52.0 % in

Q3b]

Bag of SIFT Representation + one-vs-all SVMs

1.) The accuracy of the Linear Support Vector Classifier increases linearly as the size of the training sizes increases till training size of 2000

2.) After the mark of 2000, the linearity observed in accuracy starts decreasing.

Q4]

In the confusion matrix each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class

1: For Tiny images representation and nearest neighbor classifier, it is observed that there is a lot of confusion between predictions and the corresponding labels for images, since the dark elements. So for example, an office image is labelled as industrial.

2: For the bag of visual words, (Bag of SIFT Representation + Nearest Neighbor Classifier), we can see many darker diagonal values which represents that the classifier as predicted labels with a better accuracy than the first algorithm. For eg. A highway image has been rightly predicted as a highway

3: For Bag of SIFT Representation + one-vs-all SVMs, there is higher prediction accuracy for a few more sets of images. For eg. A bedroom image is rightly labelled as a bedroom image with an accuracy of 0.75 while getting erroneously predicted as industrial(0.08), kitchen(0.08) and office(0.08) at other instances.

Acknowledgements:

1.) Shrikant Jagtap

Submission guidelines

Extract the downloaded .zip file to a folder of your preference. The input and output paths are predefined and **DO NOT** change them, (we assume that 'Surname_Givenname_SBUID_hw3' is your working directory, and all the paths are relative to this directory). The image read and write functions are already written for you. All you need to do is to fill in the blanks as indicated to generate proper outputs. **DO NOT** zip and upload the dataset on blackboard due to size limit.

When submitting your .zip file through blackboard, please -- name your .zip file as **Surname_Givenname_SBUID_hw*.zip**.

This zip file should include:

```
Surname_Givenname_SBUID_hw*
|---Surname_Givenname_SBUID_hw*.ipynb
|---Surname_Givenname_SBUID_hw*.pdf
|---Surname_Givenname_SBUID_Pred*.txt
```

For instance, student Michael Jordan should submit a zip file named "Jordan_Michael_111134567_hw3.zip" for homework3 in this structure:

```
Jordan_Michael_111134567_hw3
|---Jordan_Michael_111134567_hw3.ipynb
|---Jordan_Michael_111134567_hw3.pdf
|---Jordan_Michael_111134567_Pred.txt
```

The **Surname_Givenname_SBUID_hw*.pdf** should include a **google shared link** and **Surname_Givenname_SBUID_Pred*.pdf** should be your test set prediction file in the specified format. To generate the **google shared link**, first create a folder named **Surname_Givenname_SBUID_hw*** in your Google Drive with your Stony Brook account. The structure of the files in the folder should be exactly the same as the one you downloaded. If you alter the folder structures, the grading of your homework will be significantly delayed and possibly penalized.

Then right click this folder, click **Get shareable link**, in the People textfield, enter two TA's emails: bo.cao.1@stonybrook.edu and sayontan.ghosh@stonybrook.edu. Make sure that TAs who have the link **can edit, not just can view**, and also **uncheck** the **Notify people** box.

Colab has a good feature of version control, you should take advantage of this to save your work properly. However, the timestamp of the submission made in blackboard is the only one that we consider for grading. To be more specific, we will only grade the version of your code right before the timestamp of the submission made in blackboard.

You are encouraged to post and answer questions on Piazza. Based on the amount of email that we have received in past years, there might be dealys in replying to personal emails. Please ask questions on Piazza and send emails only for personal issues.

Be aware that your code will undergo plagiarism check both vertically and horizontally. Please do your own work.

Late submission penalty:

There will be a 10% penalty per day for late submission. However, you will have 4 days throughout the whole semester to submit late without penalty. Note that the grace period is calculated by days instead of hours. If you submit the homework one minute after the deadline, one late day will be counted. Likewise, if you submit one minute after the deadline, the 10% penalty will be imposed if not using the grace period.

https://drive.google.com/open?id=1XDIDyba8BSbKx1TU4Df6xDagN_Q7Pe4B

