CS-867   Spring   2021

Computer Vision

Assignment # 03

Scene Image Recognition

Instructor: Dr. Shahzor Ahmed

Submitted by: Monazza Qadeer Khan

Id: 206154

Dated: 27th June, 2021

Scene Image Recognition

I have done this assignment using Pycharm IDE. There's only one .py file for this assignment which can be found in the zip folder. The complete code can be found at the end of this document. Each part with code snippets is briefly explained. The following functions and libraries have been used in this assignment.

- **Functions used for K-Means Clustering, Bag of Words, Plotting, Computations:**

```
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from PIL import Image
import numpy as np
import cv2
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

- **Command used for computing SIFT features:**

```
sift = cv2.xfeatures2d.SIFT_create()
```

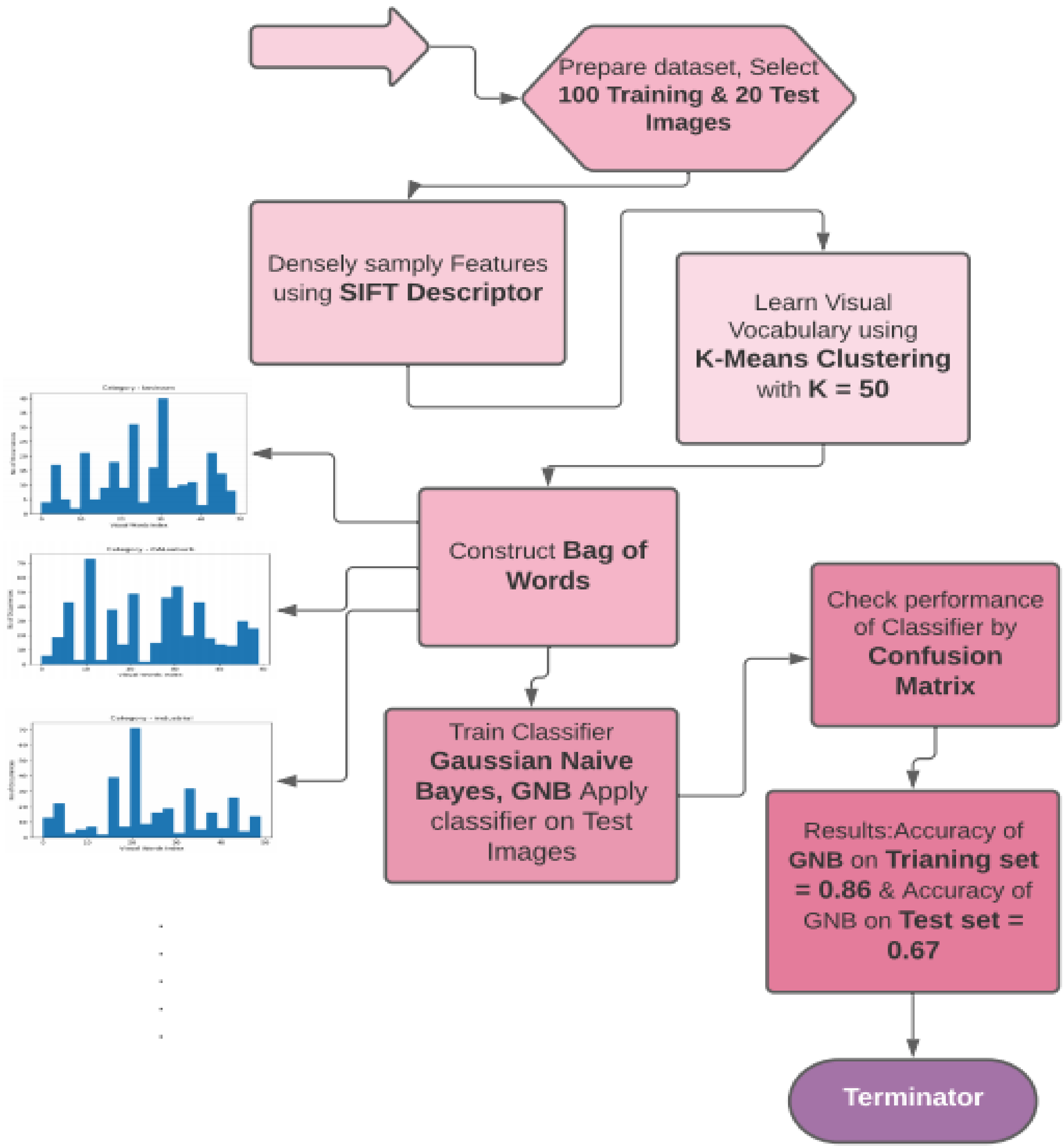
- **Functions used for Classification:**

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import GaussianNB
```

- **Functions used for Confusion Matrix:**

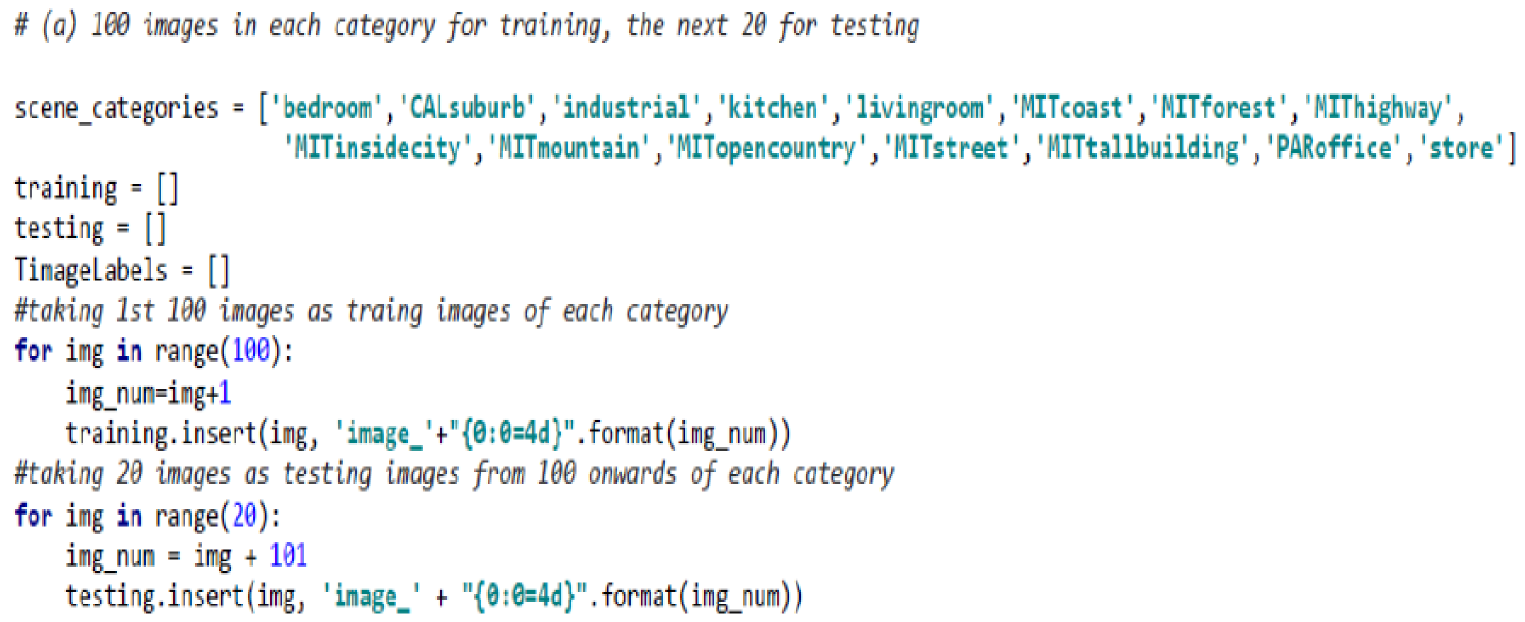
```
from sklearn.metrics import confusion_matrix
```

System Flow Diagram


Prepare dataset, Select **100 Training & 20 Test Images**
Densely samply Features using **SIFT Descriptor**
Learn Visual Vocabulary using **K-Means Clustering** with **K = 50**
Construct **Bag of Words**
Train Classifier **Gaussian Naive Bayes, GNB** Apply classifier on Test Images
Check performance of Classifier by **Confusion Matrix**
Results:Accuracy of **GNB** on **Trianing set = 0.86** & Accuracy of GNB on **Test set = 0.67**
Terminator

Part – A

In this part we have to select first 100 images in each category for training and the next 20 images for testing and discard the remaining ones. So, I downloaded all images as default folder and picked up the required images by making directory path for each image to select.



```
# (a) 100 images in each category for training, the next 20 for testing

scene_categories = ['bedroom','CALsuburb','industrial','kitchen','livingroom','MITcoast','MITforest','MIThighway',
                    'MITinsidecity','MITmountain','MITopencountry','MITstreet','MITtallbuilding','PARoffice','store']
training = []
testing = []
TimageLabels = []
#taking 1st 100 images as traing images of each category
for img in range(100):
    img_num=img+1
    training.insert(img, 'image_'+"{0:0=4d}".format(img_num))
#taking 20 images as testing images from 100 onwards of each category
for img in range(20):
    img_num = img + 101
    testing.insert(img, 'image_' + "{0:0=4d}".format(img_num))
```

Categories names are saved in the list, so, it will be easy to choose the category and corresponding images using “for” loop while processing. This way the images would be selected automatically and no need for manual selection. Separate training and test images lists are made and “for” loop is used to save the images in both the lists.

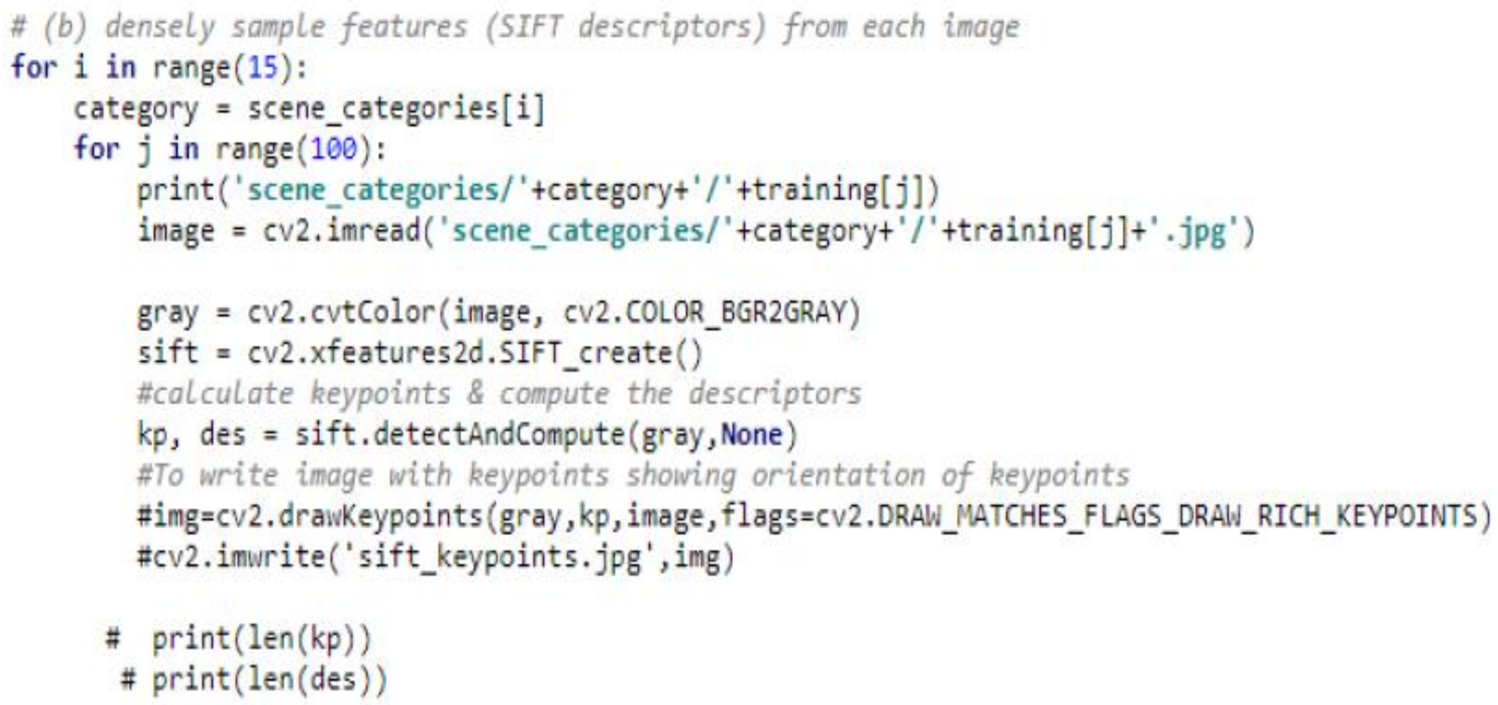
```
"{0:0=4d}".format(img_num)
```

The above command is used to make format of the image name. It gives index of the image name as 1, 2, 3, ... while running “for” loop but images are stored in the format as image_0001.

So, with this part we have successfully selected and stored the first 100 images for training and the next 20 for testing and discarded the remaining ones. Although they are present there but we ignore them.

Part – B

In this part we’ve to densely sample features i.e. compute SIFT descriptors for each image.



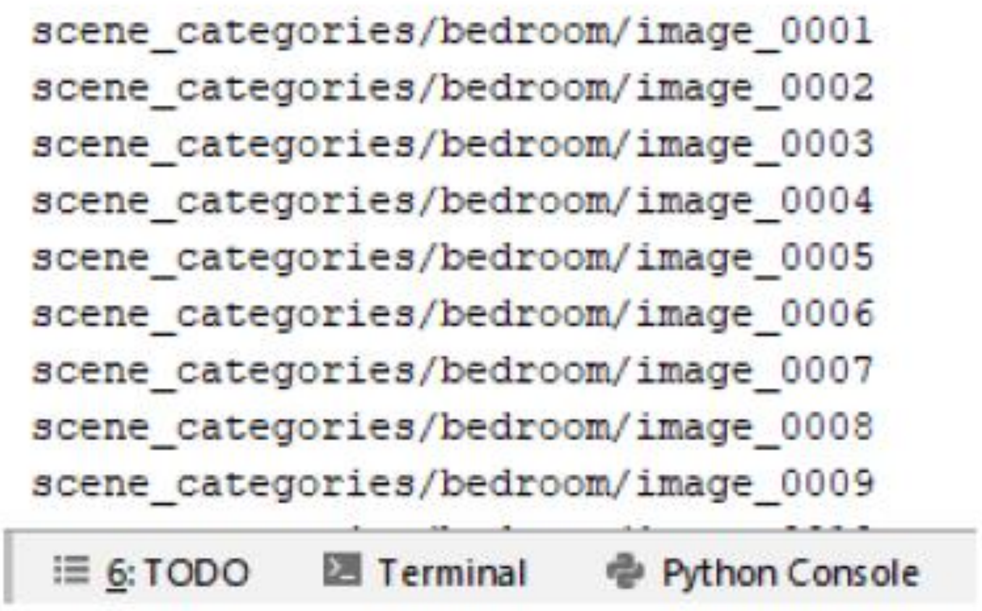
```
# (b) densely sample features (SIFT descriptors) from each image
for i in range(15):
    category = scene_categories[i]
    for j in range(100):
        print('scene_categories/'+category+'/'+training[j])
        image = cv2.imread('scene_categories/'+category+'/'+training[j]+'.jpg')

        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        sift = cv2.xfeatures2d.SIFT_create()
        #calculate keypoints & compute the descriptors
        kp, des = sift.detectAndCompute(gray,None)
        #To write image with keypoints showing orientation of keypoints
        #img=cv2.drawKeypoints(gray,kp,image,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
        #cv2.imwrite('sift_keypoints.jpg',img)

      #  print(len(kp))
       # print(len(des))
```

The path of an image that has to be selected is made by the previously assigned names. There are 15 categories and 100 training images in each category, so, this code will select category 1 in first “for” loop and in the second “for” loop it will select the 100 training images from that category and then the code finds descriptors for each image.

The descriptors have been commented in the code, so, to show it just remove the hash tag. Processing of each image with category path can also be seen as below:

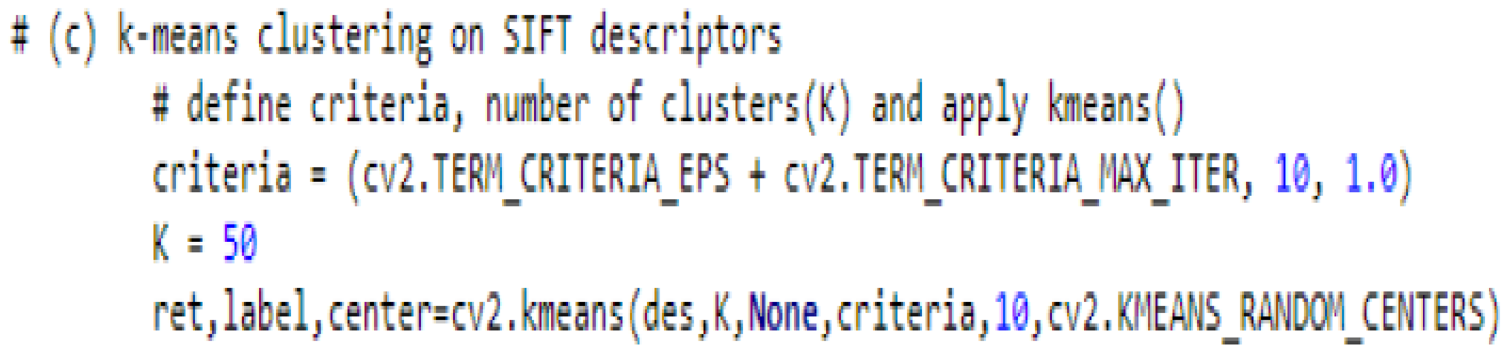


```
scene_categories/bedroom/image_0001
scene_categories/bedroom/image_0002
scene_categories/bedroom/image_0003
scene_categories/bedroom/image_0004
scene_categories/bedroom/image_0005
scene_categories/bedroom/image_0006
scene_categories/bedroom/image_0007
scene_categories/bedroom/image_0008
scene_categories/bedroom/image_0009
```

To save the computations, code for part “c” and “d” is written inside these loops.

Part – C

In this part we have to learn visual vocabulary for the scenes category. Actually, we have to perform K-means clustering on the SIFT features extracted in previous part.

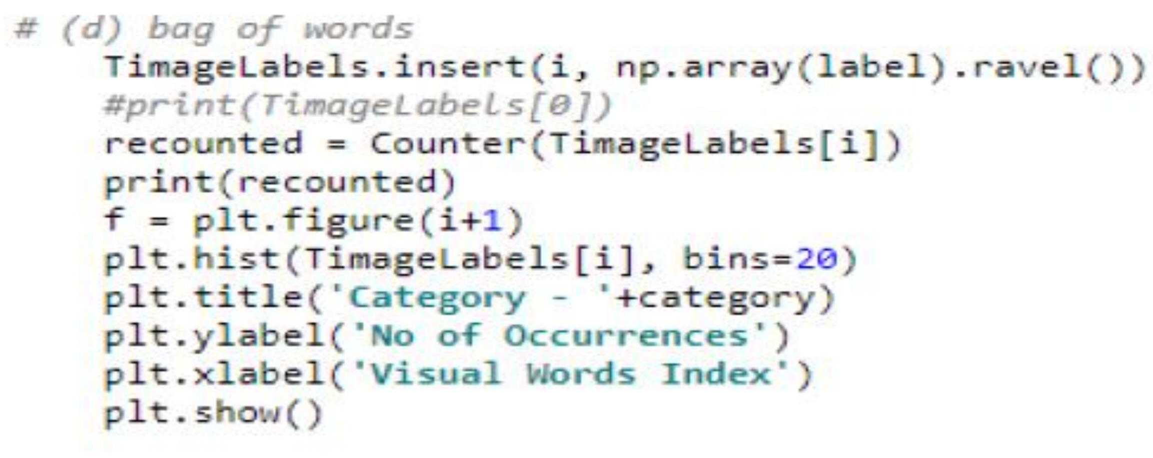


```
# (c) k-means clustering on SIFT descriptors
        # define criteria, number of clusters(K) and apply kmeans()
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
        K = 50
        ret,label,center=cv2.kmeans(des,K,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)
```

In this piece of code, K- means clustering is performed on a few thousand SIFT descriptors randomly chosen in equal number from each training image in the dataset. This has been performed in the same "for" loops as previously described as we are processing the images one by one.

Part – D

In this part, we have to construct a bag of words representation by using visual vocabulary learnt in previous part. We have to do this for each training and test image.

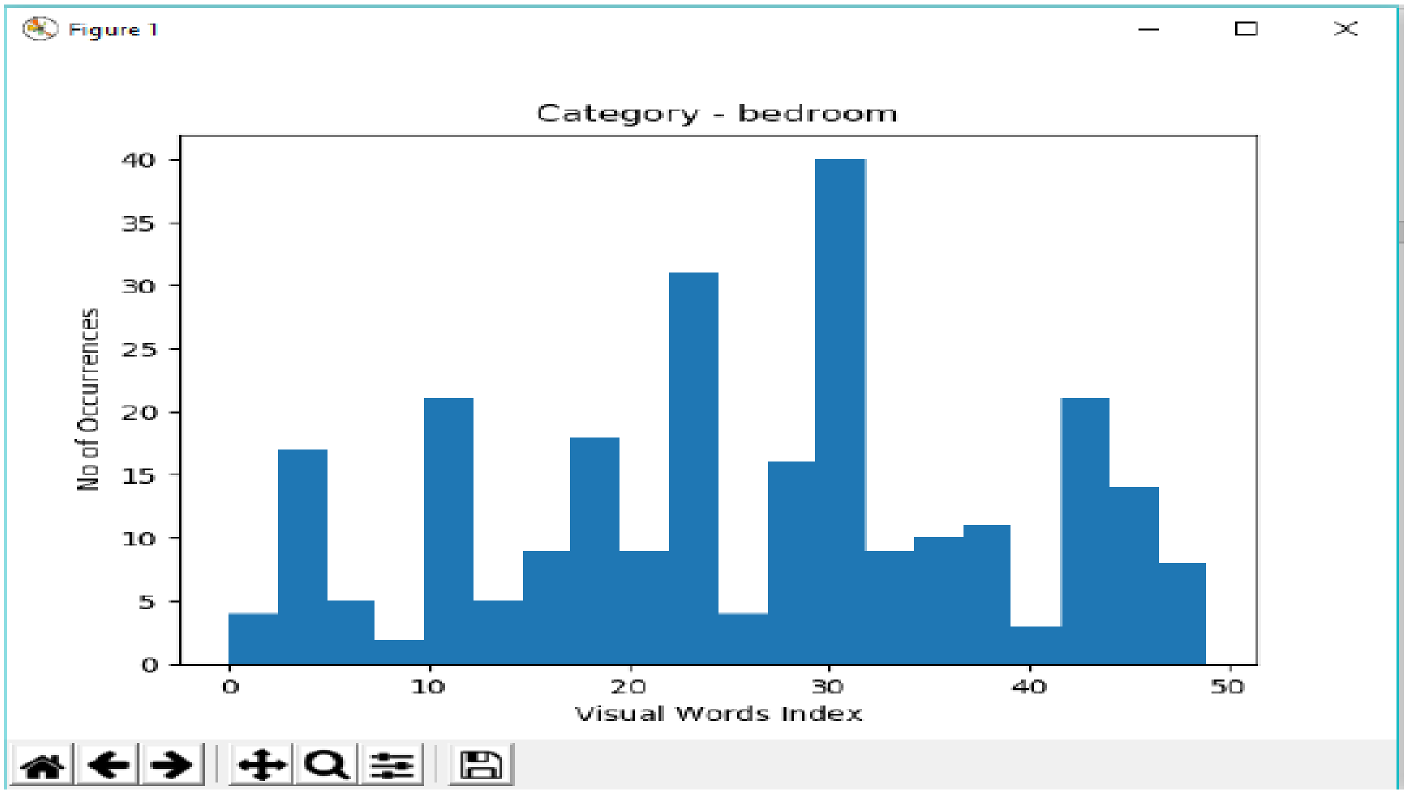


```
# (d) bag of words
    TimageLabels.insert(i, np.array(label).ravel())
    #print(TimageLabels[0])
    recounted = Counter(TimageLabels[i])
    print(recounted)
    f = plt.figure(i+1)
    plt.hist(TimageLabels[i], bins=20)
    plt.title('Category - '+category)
    plt.ylabel('No of Occurrences')
    plt.xlabel('Visual Words Index')
    plt.show()
```

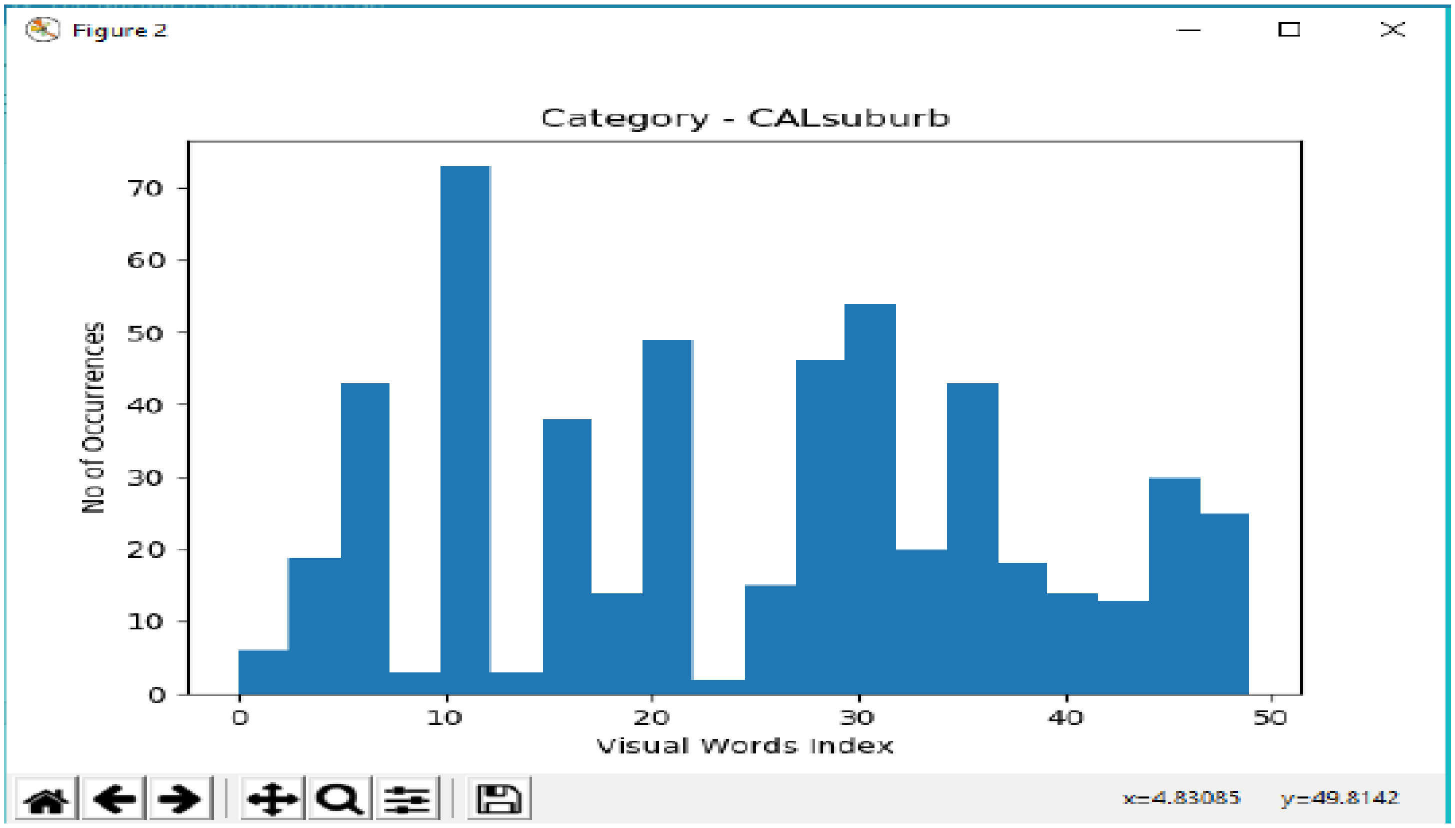
As we are running the above code in loops mentioned in the first part of this assignment, so, this code will also run on each image and construct a bag of words representation.

Although, the code runs on each image from each category and will show the results but here I am showing results for each category only once. It is as follows:

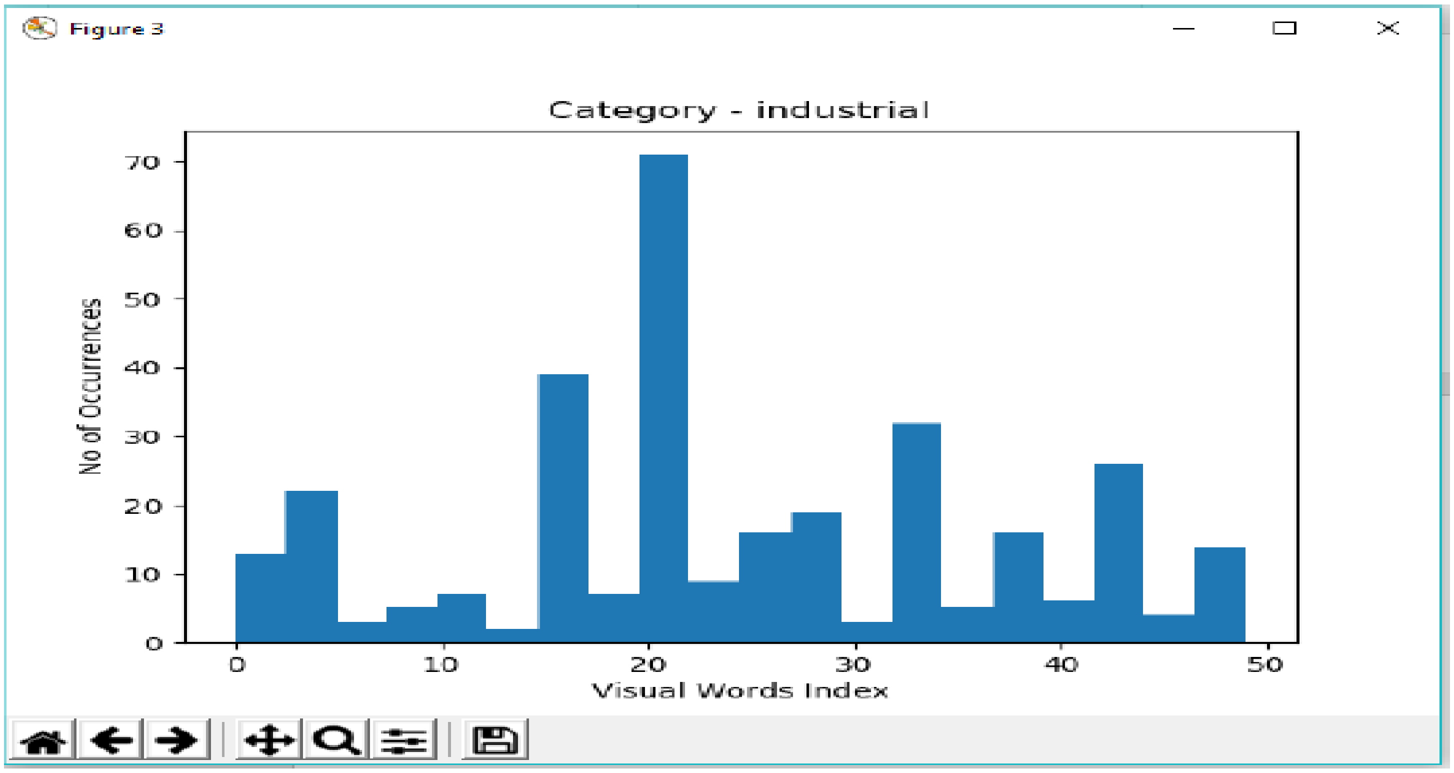
```
scene_categories/bedroom/image_0001
Counter({30: 32, 23: 21, 43: 19, 4: 15, 18: 14, 27: 14, 10: 13, 24: 10, 45: 10, 31: 8, 17
```



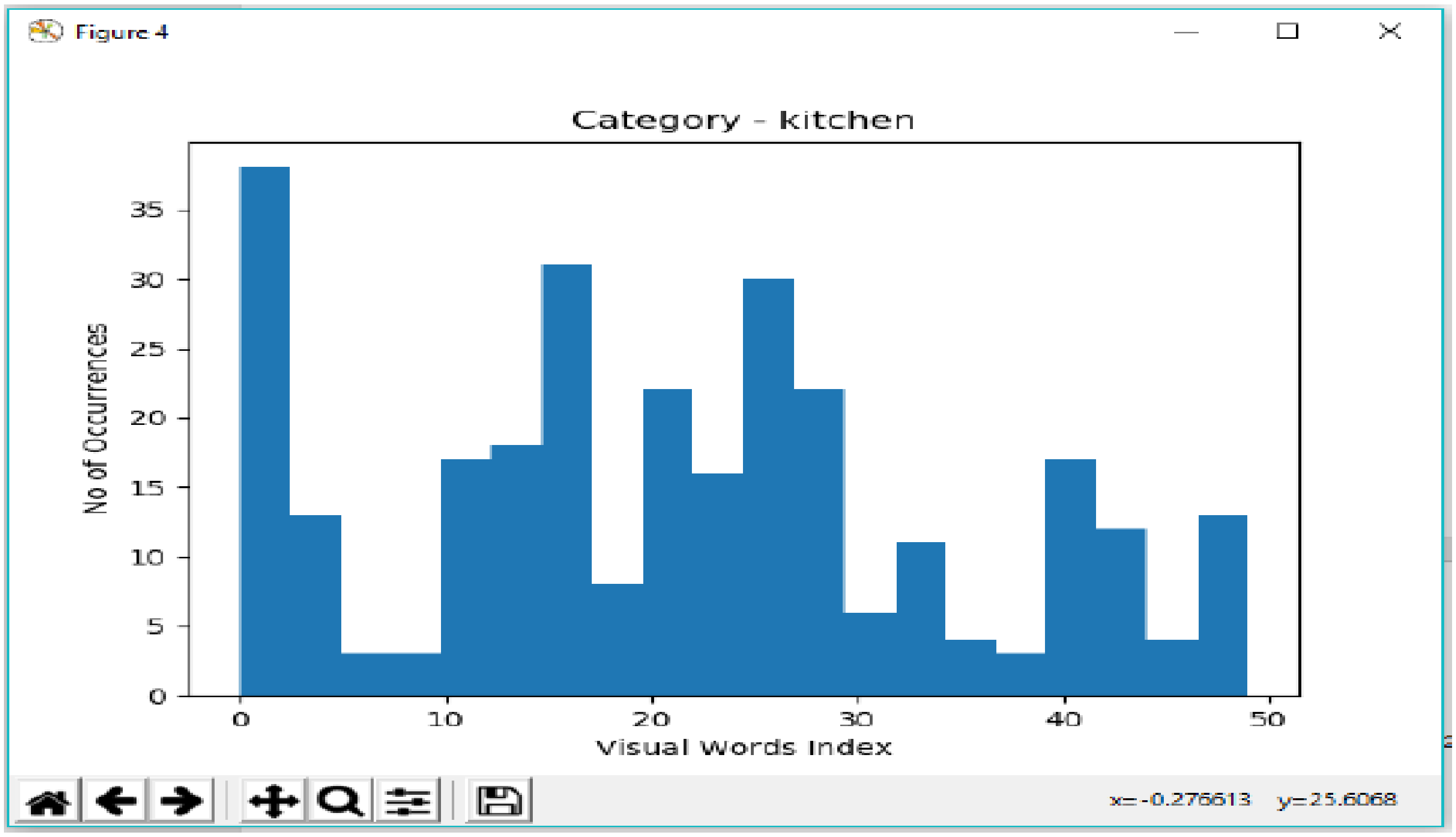
```
scene_categories/CALsuburb/image_0001
Counter({10: 67, 35: 42, 31: 29, 22: 27, 30: 25, 46: 23, 28: 23, 27: 22, 21: 2C
```

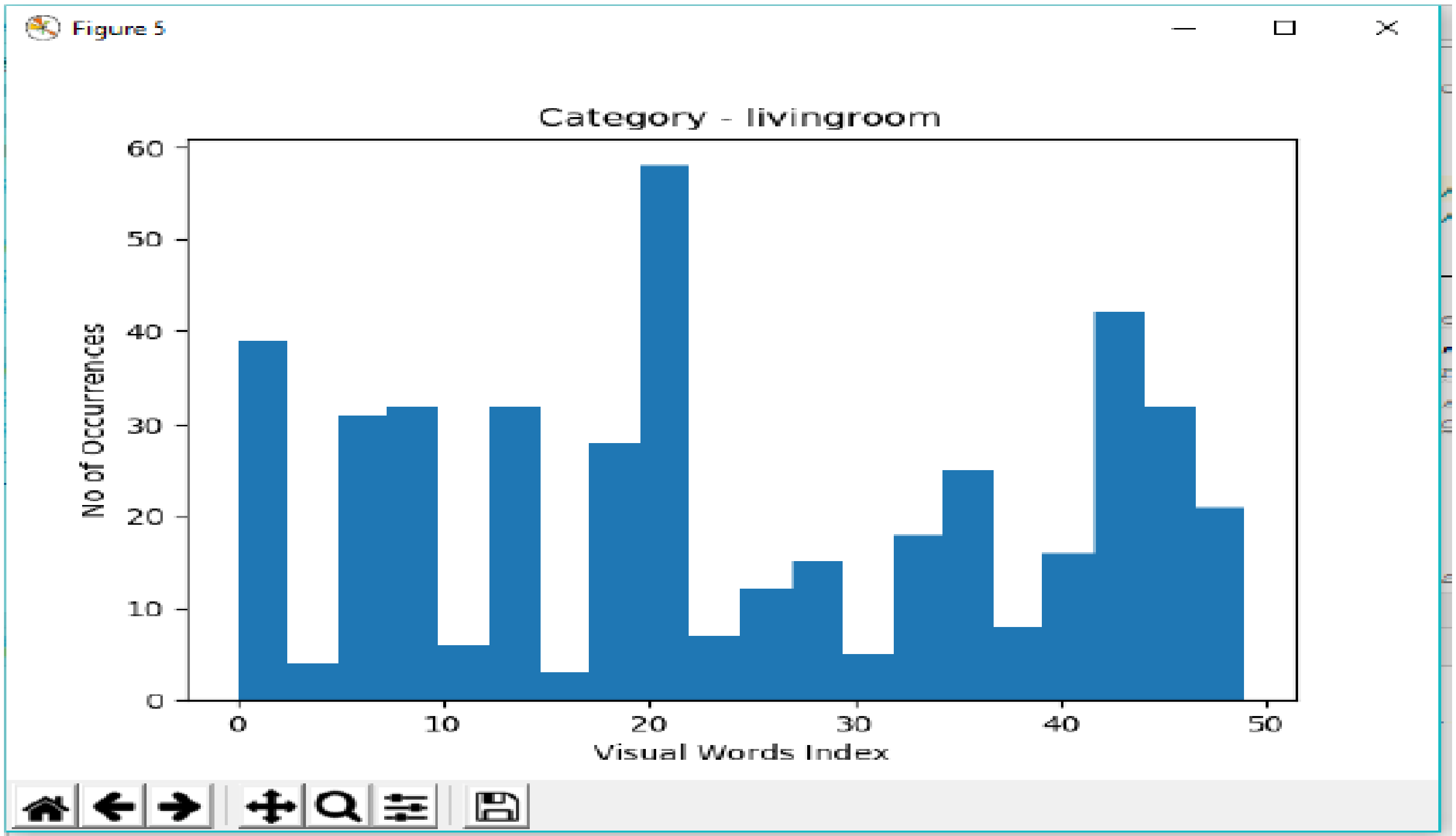


```
scene_categories/industrial/image_0001
Counter({20: 66, 17: 20, 44: 20, 3: 19, 15: 15, 32: 15, 37: 14, 34: 13, 29: 13, 1:
```

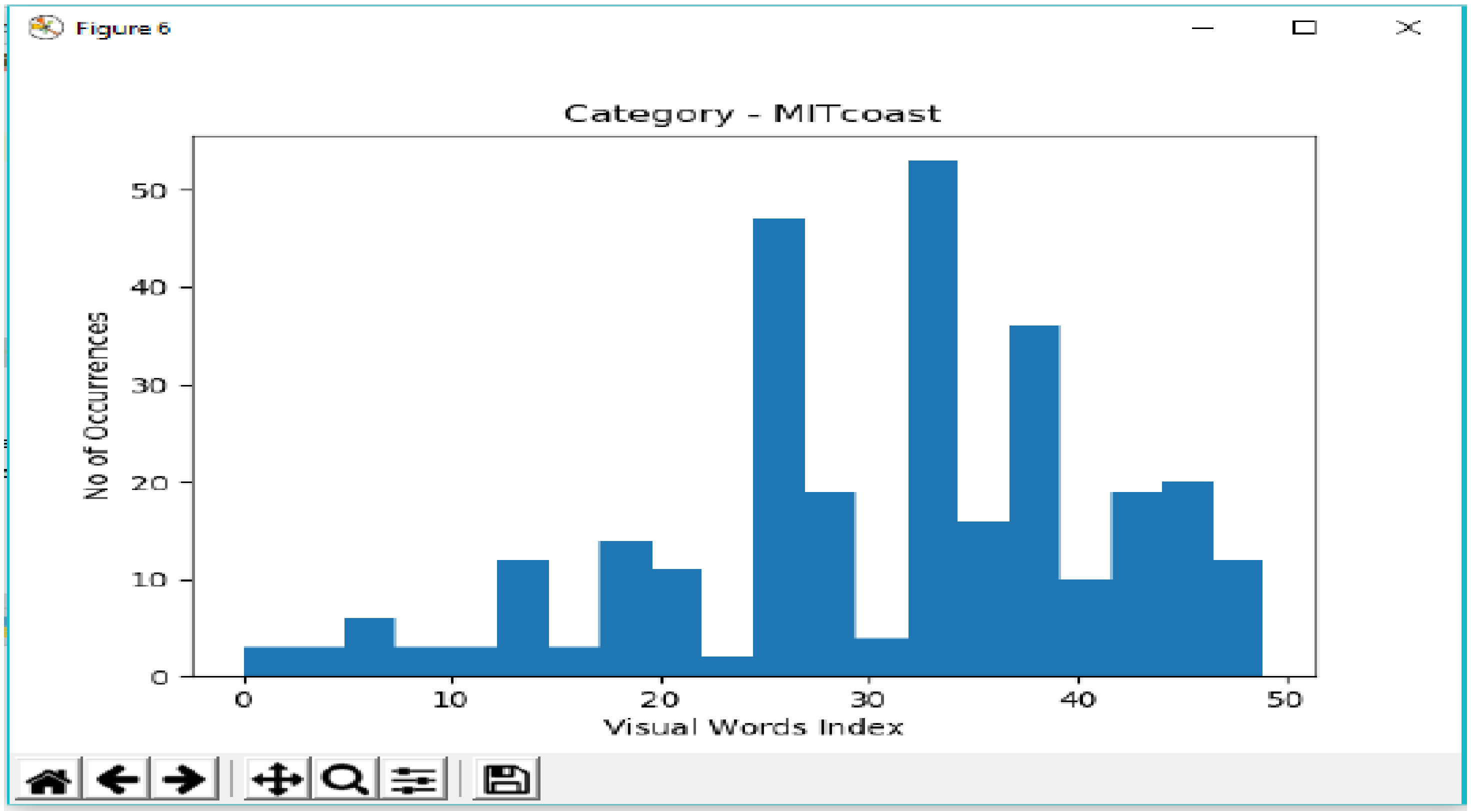


```
scene_categories/kitchen/image_0001
Counter({2: 22, 25: 22, 16: 18, 41: 16, 24: 15, 1: 14, 3: 12, 15: 12, 13: 12, 20: 12,
```

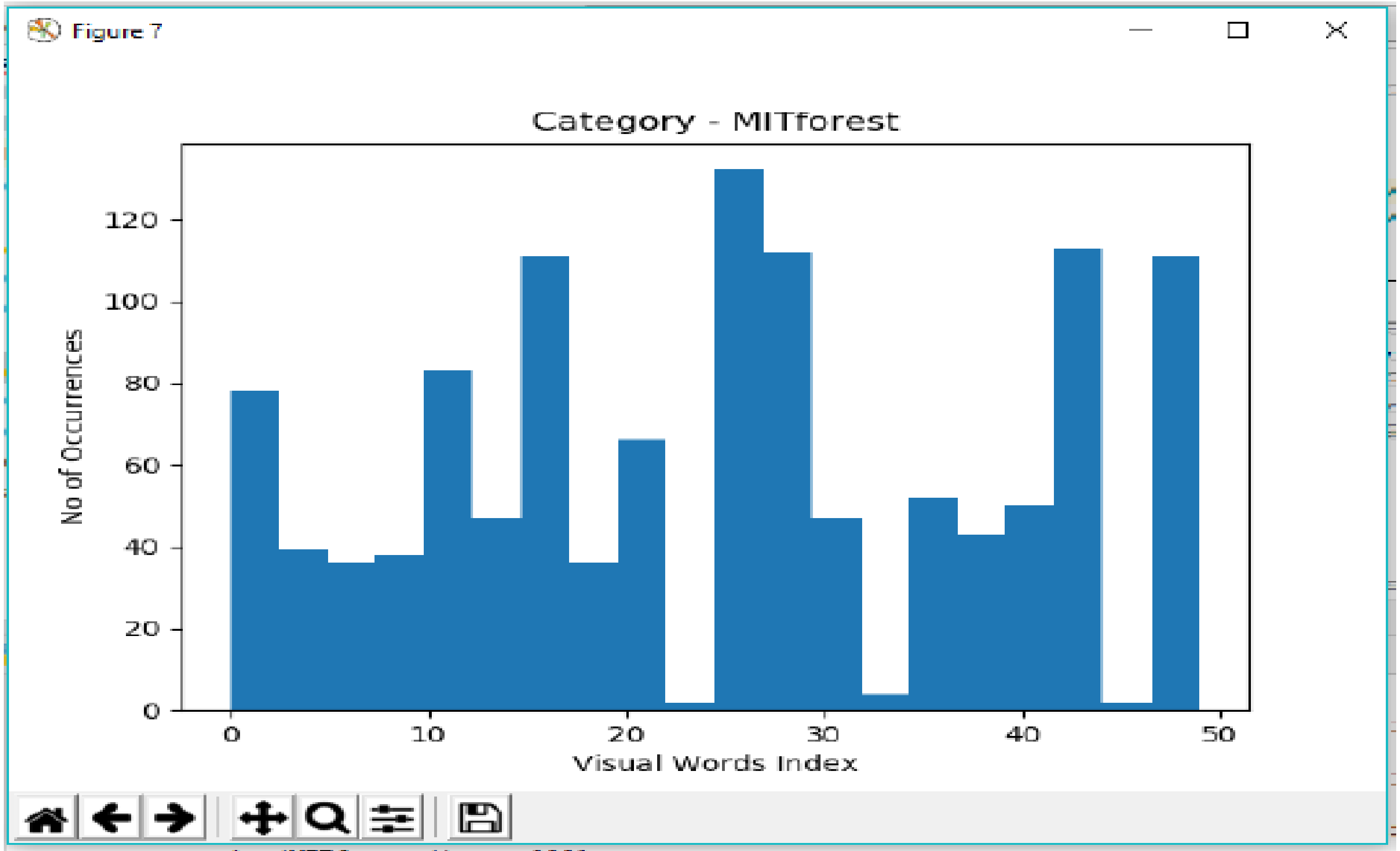




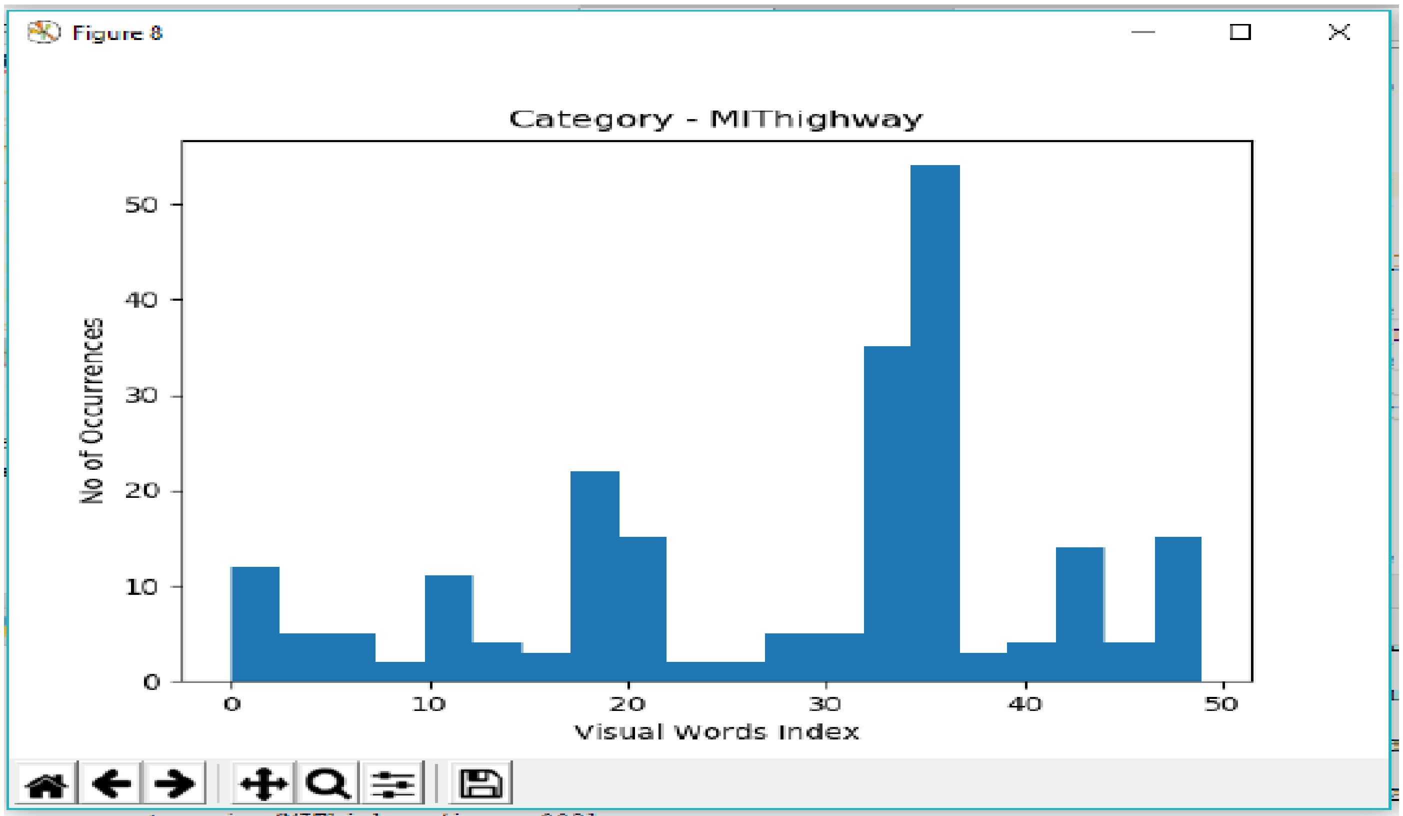
```
scene_categories/livingroom/image_0001
Counter({20: 54, 2: 36, 9: 31, 45: 30, 5: 28, 18: 27, 13: 24, 44: 24, 35: 16, 33: 16, 42:
```



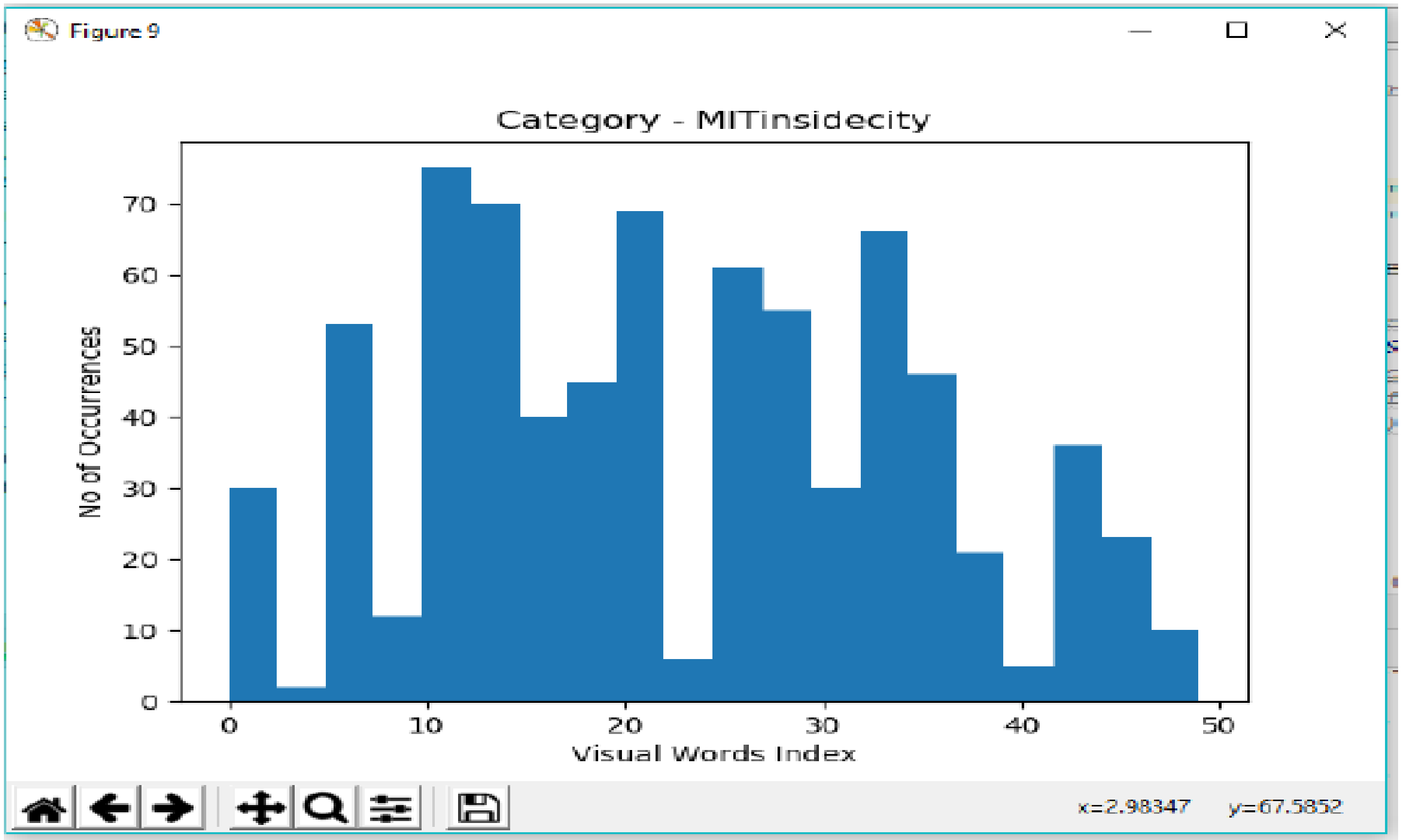
```
scene_categories/MITcoast/image_0001
Counter({32: 38, 38: 32, 26: 29, 25: 18, 43: 17, 36: 15, 45: 15, 33: 14, 28: 13, 18: 12, 1
```



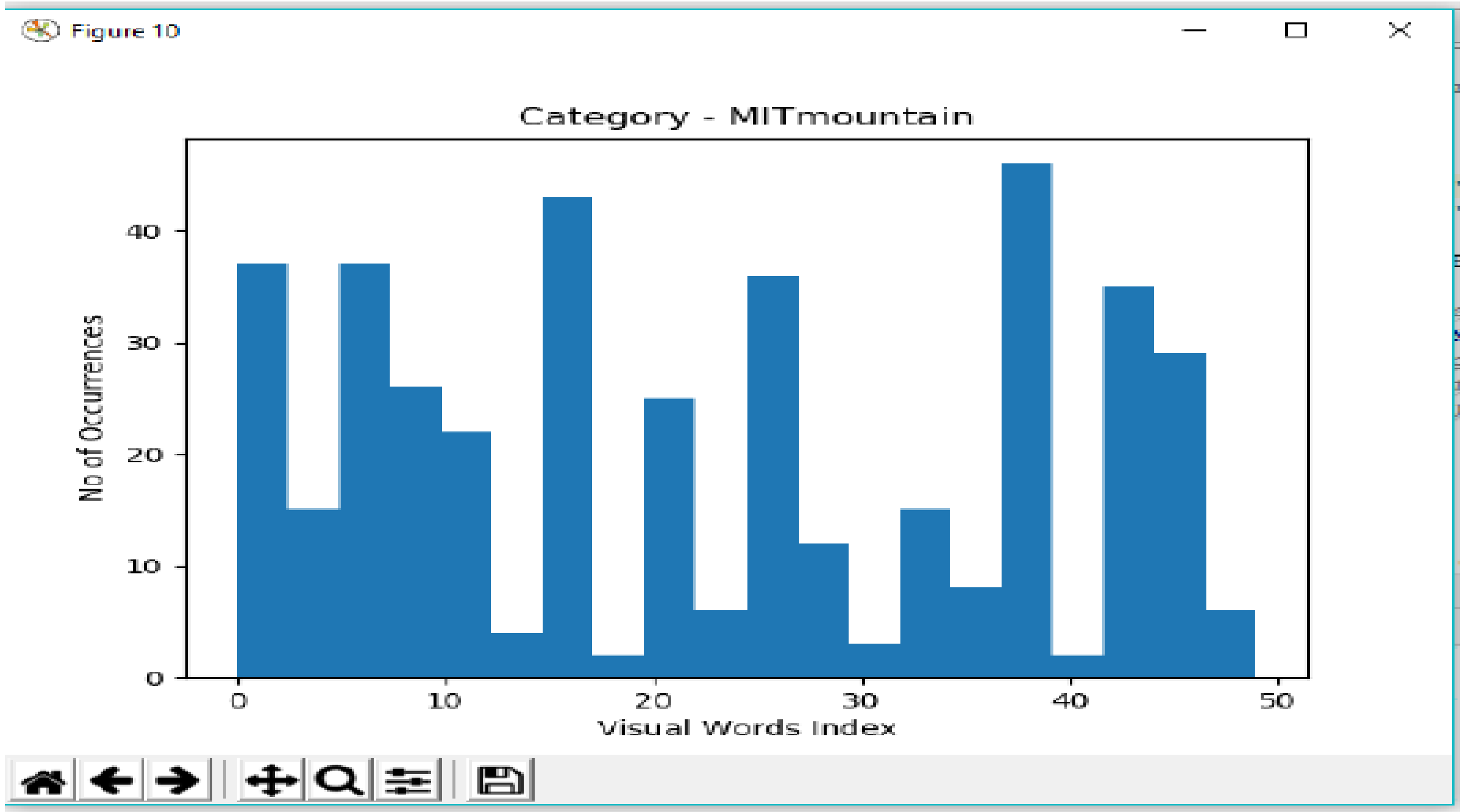
```
scene_categories/MITforest/image_0001
Counter({25: 72, 43: 67, 26: 60, 17: 58, 49: 53, 48: 52, 21: 52, 16: 51, 36: 51, 29: 50, 1
```



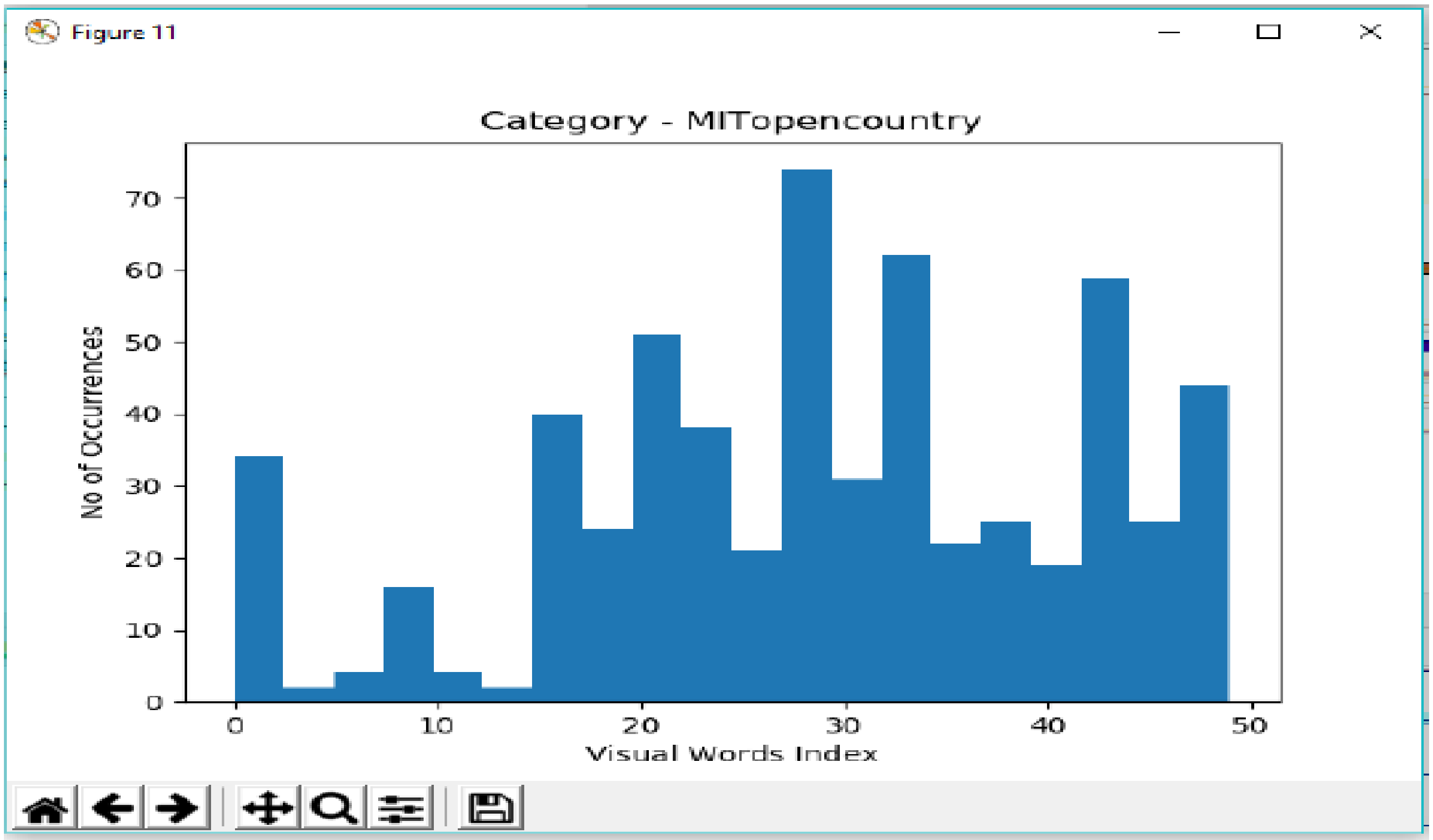
```
scene_categories/MIThighway/image_0001
Counter({35: 36, 32: 31, 19: 20, 36: 18, 48: 13, 11: 9, 43: 9, 20: 8, 2: 7, 22: 6, 3: 4, 44
```



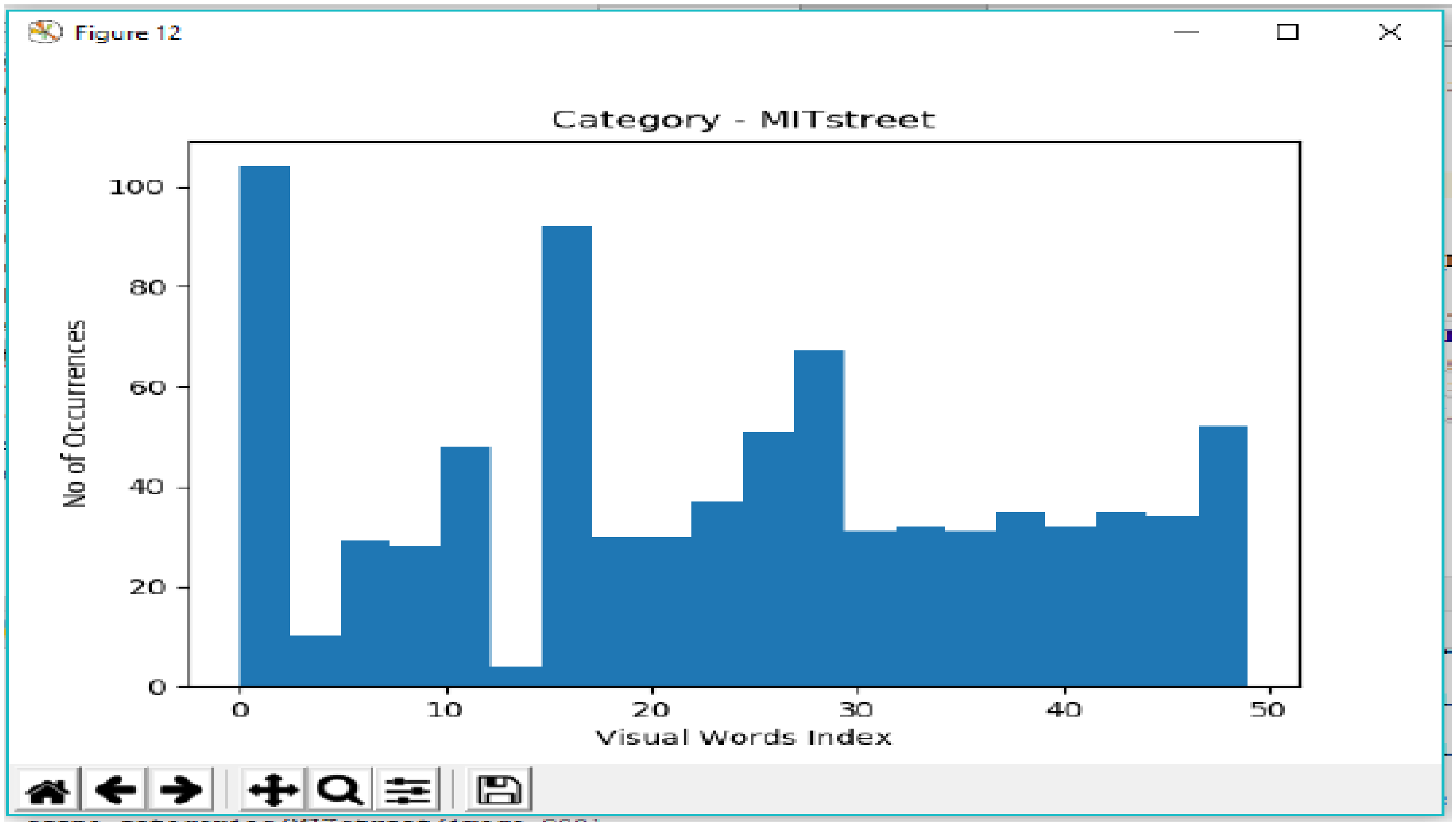
```
scene_categories/MITinsidecity/image_0001
Counter({22: 62, 10: 48, 25: 46, 19: 41, 28: 38, 14: 38, 33: 32, 13: 32, 36: 32, 31: 25, 11
```



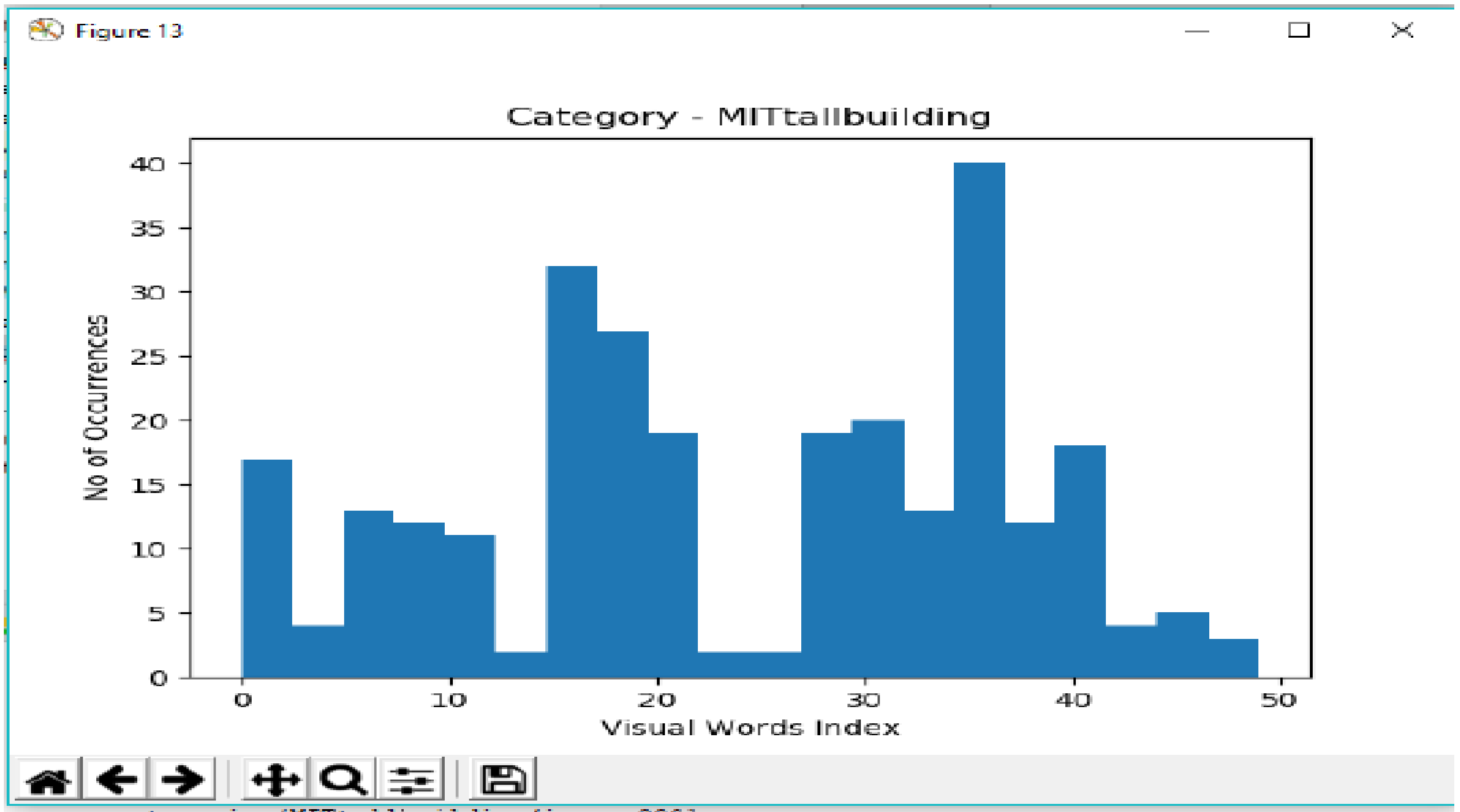
```
scene_categories/MITmountain/image_0001
Counter({2: 35, 17: 34, 6: 34, 38: 30, 46: 23, 44: 22, 25: 21, 12: 20, 8: 18, 26: 15, 37: 1
```



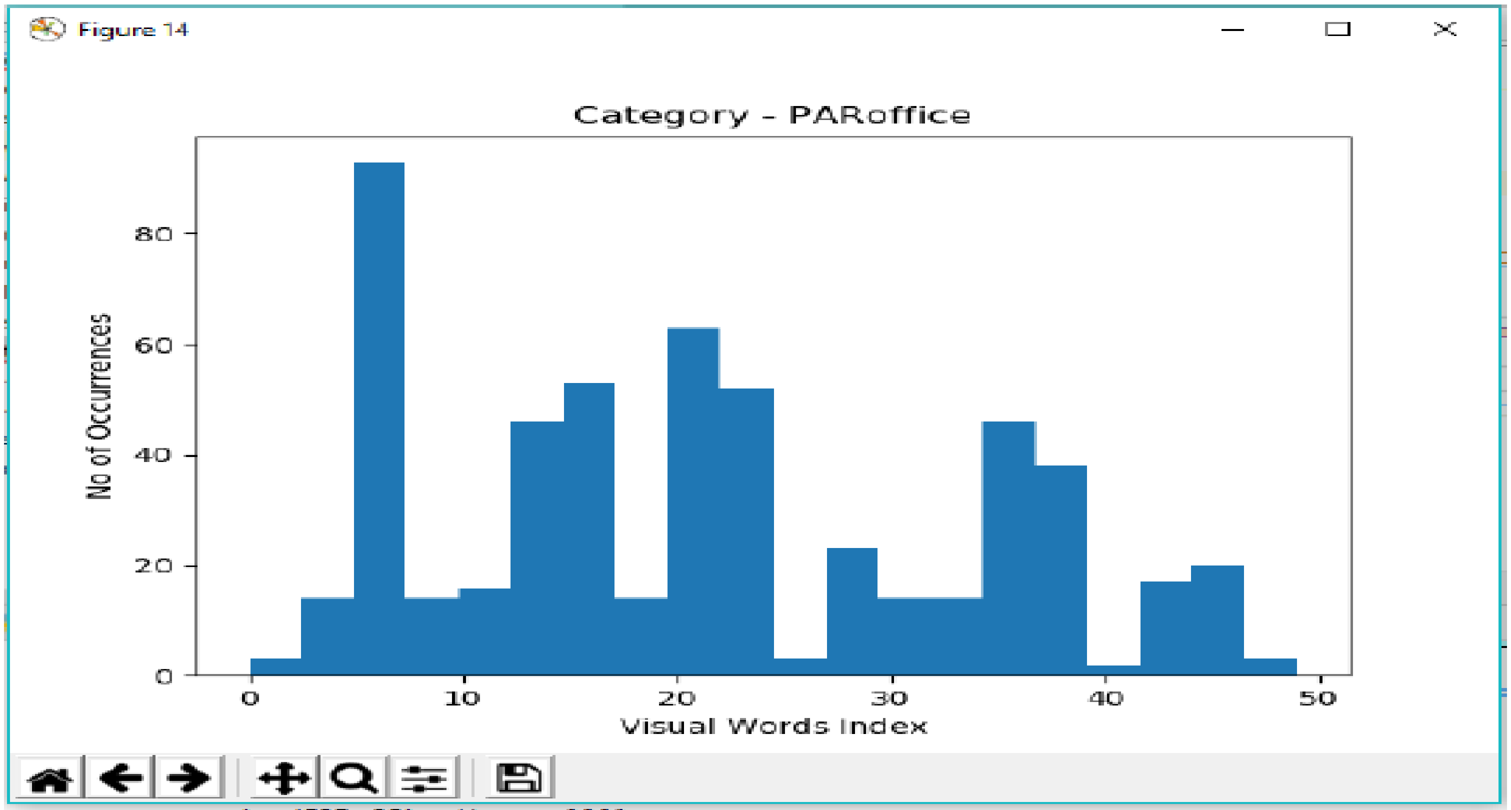
```
scene_categories/MITopencountry/image_0001
Counter({20: 48, 47: 42, 27: 38, 42: 37, 15: 37, 24: 34, 0: 32, 34: 32, 31: 30, 32: 29, 18:
```



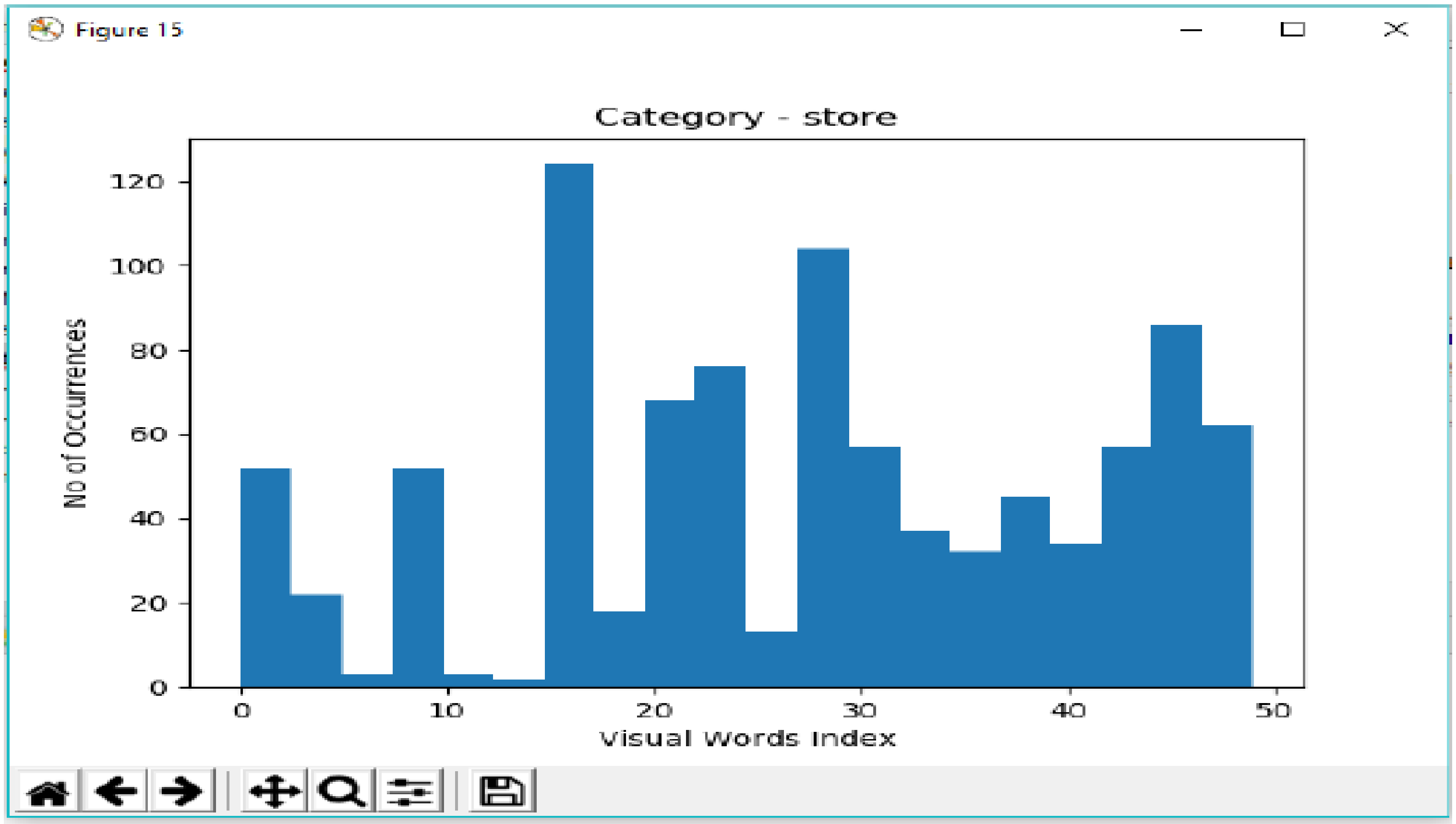
```
scene_categories/MITstreet/image_0001
Counter({17: 66, 1: 51, 2: 48, 12: 46, 49: 38, 27: 34, 24: 34, 37: 33, 25: 28, 40: 27, 35:
```



```
scene_categories/MITtallbuilding/image_0001
Counter({16: 29, 18: 25, 36: 24, 35: 16, 40: 15, 0: 15, 20: 12, 30: 12, 34: 11, 8: 10, 28:
```



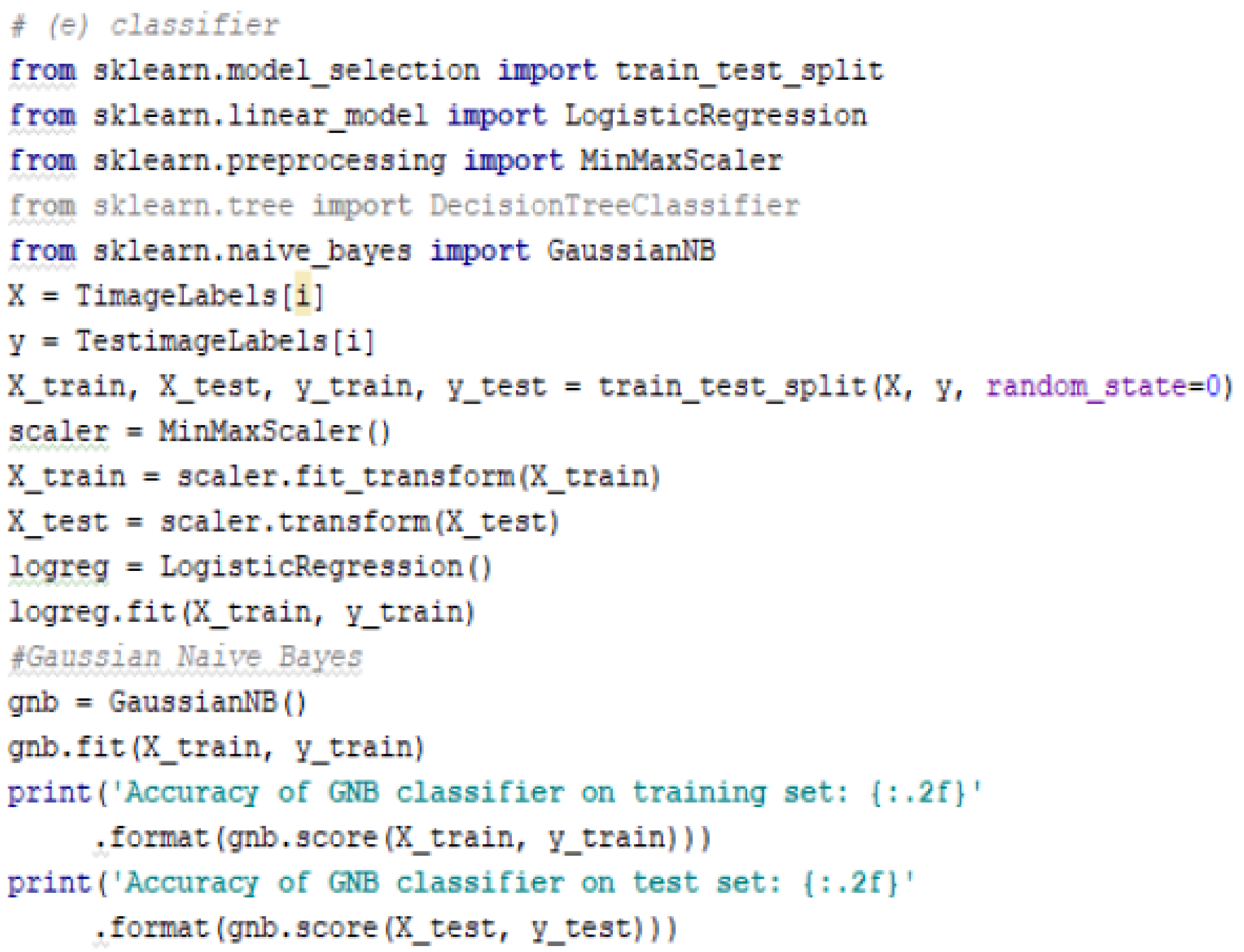
```
scene_categories/PARoffice/image_0001
Counter({6: 52, 21: 40, 24: 34, 14: 31, 36: 28, 15: 25, 7: 25, 20: 22, 29: 21, 46: 19, 23:
```



```
scene_categories/store/image_0001
Counter({28: 58, 20: 57, 15: 56, 45: 54, 17: 54, 24: 47, 27: 45, 9: 44, 31: 35, 34: 35, 48:
```

Part – E

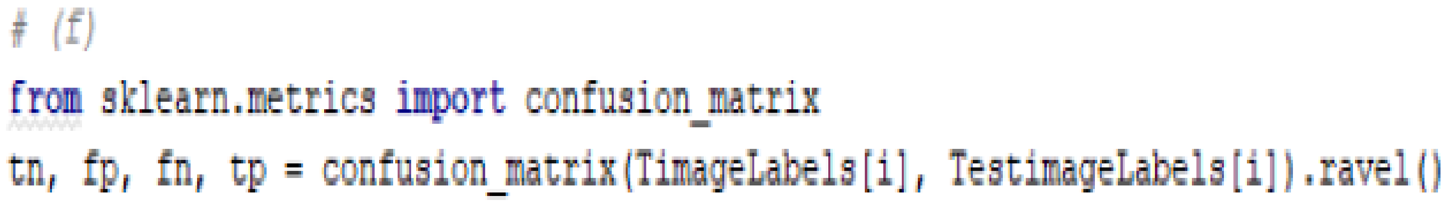
In this part we have to train a classifier based on the training images and then classify the test images. Gaussian naive Bayes is used for classification.



```
# (e) classifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
X = TimageLabels[i]
y = TestimageLabels[i]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
#Gaussian Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train, y_train)
print('Accuracy of GNB classifier on training set: {:.2f}'
     .format(gnb.score(X_train, y_train)))
print('Accuracy of GNB classifier on test set: {:.2f}'
     .format(gnb.score(X_test, y_test)))
```

Part – F

In this part, we have to compute confusion matrix for classification and the overall accuracy.



```
# (f)
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(TimageLabels[i], TestimageLabels[i]).ravel()
```

The table or confusion matrix can be seen by running the code. Here are the final results:

Accuracy of GNB classifier on training set: 0.86

Accuracy of GNB classifier on test set: 0.67

By running the same code only for 50 training images, it was observed that frequency of words is less as compared to that for 100 images. So, 100 images store more words and its performance is better.

********************Complete Code**********************

```
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from PIL import Image
import numpy as np
import cv2
from sklearn.cluster import KMeans
```

```
import matplotlib.pyplot as plt

# Part A
# 100 images in each category for training, the next 20 for testing

scene_categories=['bedroom','CALsuburb','industrial','kitchen','livingroom
','MITcoast','MITforest','MIThighway','MITinsidecity','MITmountain','MITop
encountry','MITstreet','MITtallbuilding','PARoffice','store']
training = []
testing = []
TimageLabels = []
TestimageLabels = []
#taking 1st 100 images as traing images of each category
for img in range(100):
    img_num=img+1
    training.insert(img, 'image_'+"{0:0=4d}".format(img_num))
#taking 20 images as testing images from 100 onwards of each category
for img in range(20):
    img_num = img + 101
    testing.insert(img, 'image_' + "{0:0=4d}".format(img_num))

# Part B
# densely sample features (SIFT descriptors) from each image
for i in range(15):
    category = scene_categories[i]
    for j in range(100):
        print('scene_categories/'+category+'/'+training[j])
        image =
cv2.imread('scene_categories/'+category+'/'+training[j]+'.jpg')

        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        sift = cv2.xfeatures2d.SIFT_create()
        #calculate keypoints & compute the descriptors
        kp, des = sift.detectAndCompute(gray,None)
        #To write image with keypoints showing orientation of keypoints

#img=cv2.drawKeypoints(gray,kp,image,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RIC
H_KEYPOINTS)
        #cv2.imwrite('sift_keypoints.jpg',img)

      #  print(len(kp))
       # print(len(des))

# Part C
        # k-means clustering on SIFT descriptors
        # define criteria, number of clusters(K) and apply kmeans()
        criteria = (cv2.TERM_CRITERIA_EPS
+cv2.TERM_CRITERIA_MAX_ITER,10,1.0)
        K = 50

ret,label,center=cv2.kmeans(des,K,None,criteria,10,cv2.KMEANS_RANDOM_CENTE
RS)
```

```
# Part D
    # bag of words
    TimageLabels.insert(i, np.array(label).ravel())
    #print(TimageLabels[0])
    recounted = Counter(TimageLabels[i])
    print(recounted)
    f = plt.figure(i+1)
    plt.hist(TimageLabels[i], bins=20)
    plt.title('Category - '+category)
    plt.ylabel('No of Occurrences')
    plt.xlabel('Visual Words Index')
    plt.show()

# Part E
#  classifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
X = TimageLabels[i]
y = TestimageLabels[i]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
#Gaussian Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train, y_train)
print('Accuracy of GNB classifier on training set: {:.2f}'
     .format(gnb.score(X_train, y_train)))
print('Accuracy of GNB classifier on test set: {:.2f}'
     .format(gnb.score(X_test, y_test)))

# Part F
# Confusion matrix
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(TimageLabels[i],
TestimageLabels[i]).ravel()
```

-------------------------------- The End --------------------------------