Bag of Words

APPROACH

Bag of Words algorithm is based on the idea of document classification. It can be used for image classification in small systems wherein user defines the number of clusters. Bag of words is the vector of occurrence of count of words and is expressed as a histogram of each word. Training of Bag of Words is done to cluster the training images into several clusters. The calculated features of test images can be compared to words in a dictionary to get histogram. The steps followed in this process are as follows:

1. Calculate the SIFT descriptors of all the training and testing images.
2. Club the SIFT descriptors of the training images together in a bag and choose K from this bag of descriptors of the training images.
3. Now the K means algorithm is applied after choosing the feature descriptors from the set or bag and each of the image descriptor is then classified on the basis of these descriptors.
4. The algorithm runs until convergence of the k means is reached in this case.
5. It is important to then consider the nearest neighbor algorithm for the descriptors of the test image descriptor.
6. Once this is carried out, each of the image descriptors are classified into K bins on the basis of the codeword generated and the histogram of each of the image descriptors is calculated.
7. Once all the histograms are obtained, the histogram which has the largest similarity with that of the test image descriptor histogram or the lowest error margin in between the histograms isconsidered as the best match to the test image. Hence through this histogram matching principle we can state which image is closely associated with the test image. We use KNN to check in this case.

OpenCV Python implementation

- Command line arguments dealt with for the source folder containing all the training and testing images.
- Reading the training images and loading in the list using glob in python.
- Computing the SIFT features for each of them.
- Find the key points and descriptors with SIFT for all the training images.
- Creating manual training label for all the training images.
- Setting up PCA, reduction of the dimensionality of the SIFT to utilise only the SVD based essential singular values through SVD.
- Stack up the descriptors so that that we have a descriptor array that can be used for histogram calculation.
- Perform k means for this clustering the descriptors.
- Develop vocabulary of the histograms for all the descriptors to be put inside the bag.
- Display histogram for the normalized histograms that need to be fed to the KNN for prediction and training.
- Train the KNN for obtaining the decision boundaries and generating the codewords to be verified for the training phase
- Testing phase for all the images given by load test images again using glob.

- For each image again create a vocabulary for histogram matching.
- After calculation of the SIFT descriptor of the train image and calculate the PCA similar to the training phase.
- Use the K means algorithm to check with the centroids already assigned during training phase for labels and create the histogram.
- Use this histogram to predict the class of the object using KNN classifier for prediction which was setup during the training phase.
- Append the predictions list with all the obtained final results for the test images and finally compute the confusion matrix.

SOURCE CODE

The code was built on MAC-OS Sierra with python 3.5 and OpenCV 3.1
It can be run with the Opencv version mentioned above with the required packages for python to support them. It can be simply compiled with any Python compiler.

Arguments to be passed for Question 1:
./BOW.py folder_images

argv[0]: The compiled executable that you save it with.
argv[1]: Source folder containg classes in directory with test and train image folders .


```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  29 12:24:19 2017

@author: aakashshanbhag
"""

import cv2
import numpy as np
from sklearn.decomposition import PCA
import glob
from sys import argv
import os
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from matplotlib import pyplot as plt


# Command line argumets dealt with
script, train = argv

print("The script called:", script)
print("Training folder utlised: ", train)

# Reading the training images and loading in the list
imlist={}
```

```
train_count=0
classtype_count=0
for each in glob.glob(os.path.abspath(train) + "/*"):
    word=each.split("/")[-1]
    print("Reading image category "+word)
    imlist[word]=[]
    classtype_count+=1
    for imagefile in glob.glob(os.path.abspath(train)+"/"+word+
"/train/*"):
        print ("Reading file "+imagefile)
        im=cv2.imread(imagefile,0)
        imlist[word].append(im)
        train_count+=1

# Computing the SIFT features for each of them
# Initiate SIFT detector
sift = cv2.xfeatures2d.SIFT_create()

# Find the keypoints and descriptors with SIFT for all the
training images
desc_list=[]
train_labels=np.array([])
train_dict={}

label_count=0
for word in imlist:
    train_dict[str(label_count)]=word
    print("computing the features for "+word )
    for im in imlist[word]:
        train_labels=np.append(train_labels,label_count)
        kp,des=sift.detectAndCompute(im,None)
        desc_list.append(des)
    print("Train label "+str(label_count)+" signifies "+word)
    label_count+=1

# Setting up PCA an reduction of the dimensionality of the SIFT
# to utilise only the SVD based essential singular values

PCA_ncomponents=20
print("128 D SIFT feature reduced to "+str(PCA_ncomponents)+" D
feature space")
desc_transformed_list=[]
pca=PCA(n_components=PCA_ncomponents)
for desc in desc_list:
    pca.fit(des)
    desc_transformed=pca.transform(desc)
    desc_transformed_list.append(desc_transformed)


# Perform clustering on the lower dimesnsional

# Stack up the normals so that that we have a descriptor array
stack=np.array(desc_transformed_list[0])
for i in desc_transformed_list:
    stack=np.vstack((stack,i))

# Dealing with the vertical stack up issue eficiently by
reshaping
```

```
stack1=stack[np.array(desc_transformed_list[0]).shape[0]:stack
.shape[0],:]

# Perform k means for this clustering the descriptors
k=800
print("K means clustering initialised")
l = KMeans(k)
Kmeans_result=l.fit_predict(stack1)
print("K means clustering completed")

# Develop vocabulary
# Generating histogram per image
hist_count=0
histogram=np.array([np.zeros(k) for i in range(train_count)])
for i in range(train_count):
    for j in range(len( desc_transformed_list[i])):
        index=Kmeans_result[j+hist_count]
        histogram[i][index]+=1
    hist_count+=len( desc_transformed_list[i])
print( np.sum(histogram[:][:]))
print("Histogram generated")

#Scale the standard histogram
scale=StandardScaler().fit(histogram)
histogram=scale.transform(histogram)

# Display histogram
x_scalar=np.arange(k)
y_scalar=np.array([abs(np.sum(histogram[:,h],dtype=np.int32))
for h in range(k)])
print (y_scalar)
print(np.sum(y_scalar))
plt.bar(x_scalar,y_scalar)
plt.xlabel("Visual Word Index")
plt.ylabel("Frequency")
plt.title("Compute Vocabulary Generated")

# Train the KNN for obtaining the decision boundaries
clf=KNeighborsClassifier(n_neighbors=55)
print(clf)
clf.fit(histogram,train_labels)
print("Training phase completed")


# Testing phase for all the images given

# Load test images
imlist_test={}
test_count=0
for each in glob.glob(os.path.abspath(train) + "/*"):
    word=each.split("/")[-1]
    print("Reading image category "+word)
    imlist_test[word]=[]
    for imagefile in glob.glob(os.path.abspath(train)+"/"+word+
"/test/*"):
        print ("Reading test file "+imagefile)
        im=cv2.imread(imagefile,0)
        imlist_test[word].append(im)
```

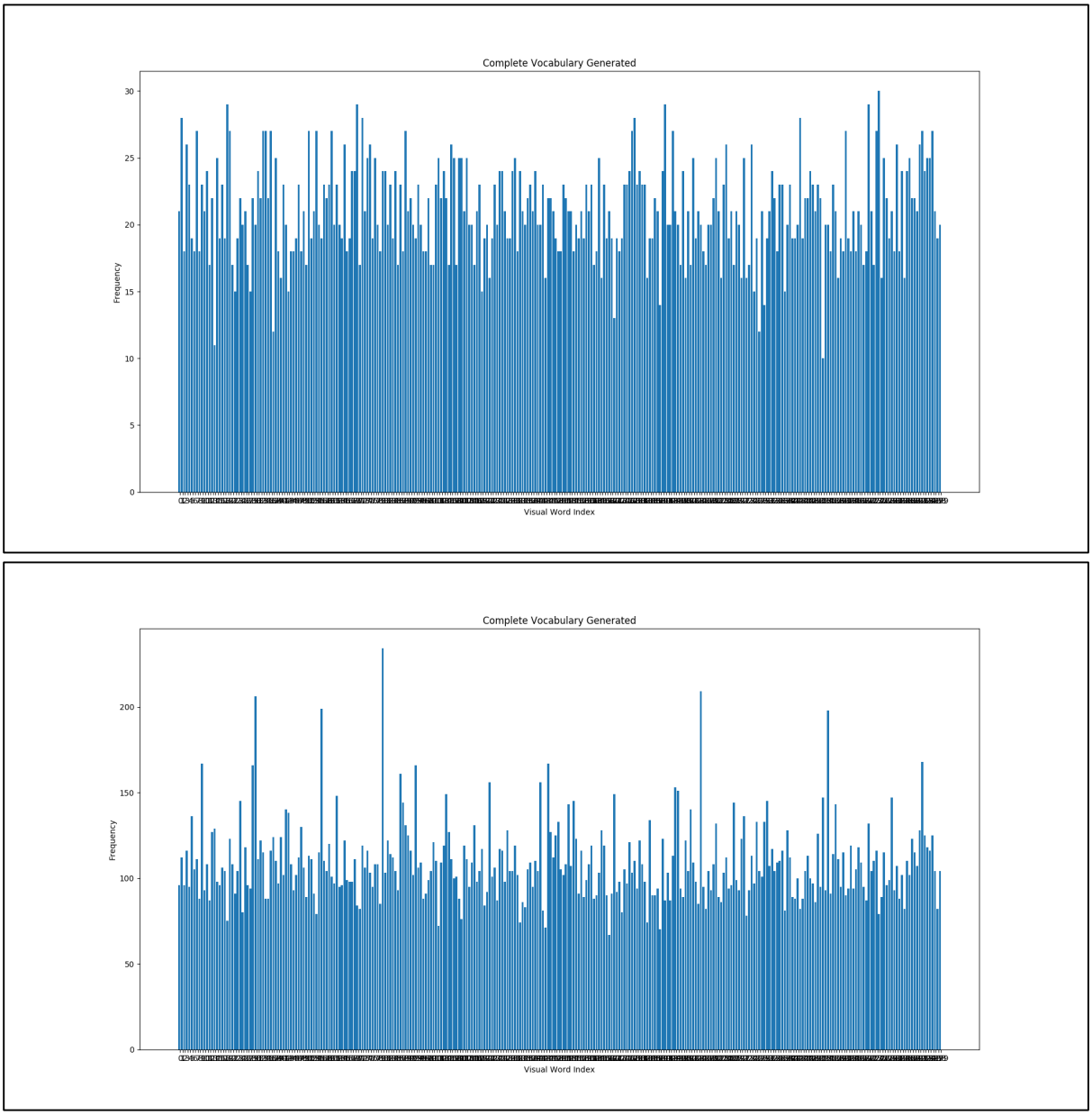
```
        test_count+=1

# Create a prediction list
predictions = []
for word in imlist_test:
            print ("processing " +word)
            for im in imlist_test[word]:
                # Calculate the descriptor for the test images

kp_test,des_test=sift.detectAndCompute(im,None)
                # Apply PCA
                pca.fit(des_test)
                des_transformed_test=pca.transform(des_test)
                # Generate vocab for test image
                test_vocab=np.zeros((1,k))
                test_ret = l.predict(des_transformed_test)

                # Generate test vocab
                for each in test_ret:
                    test_vocab[0,each]+=1

                # Scale for each histogram
                test_vocab=scale.transform(test_vocab)
                cl=clf.predict(test_vocab)
                # Append finally into the prediction list
                predictions.append({
                    'image':im,
                    'class':cl,
                    'object_name':train_dict[str(int(cl[0]))]
                    })
                print(train_dict[str(int(cl[0]))])

# Display final predictions
for each in predictions:
    plt.imshow(cv2.cvtColor(each['image'],
cv2.COLOR_GRAY2RGB))
    plt.title(each['object_name'])
    plt.show()
```

OUTPUT FOR ONE TEST CASE

We have 100 train images and 50 images (5 classes of 10 test images each)
Computing the **confusion matrix** using K=300, PCA_components=20, NN=10 with a clear bias towards guitar set. Diagonal elements show the accuracy of the estimated model with the above parameters. Confusion matrix created using the following results obtained from testing which present the robustness of the obtained features using Spyder for python.

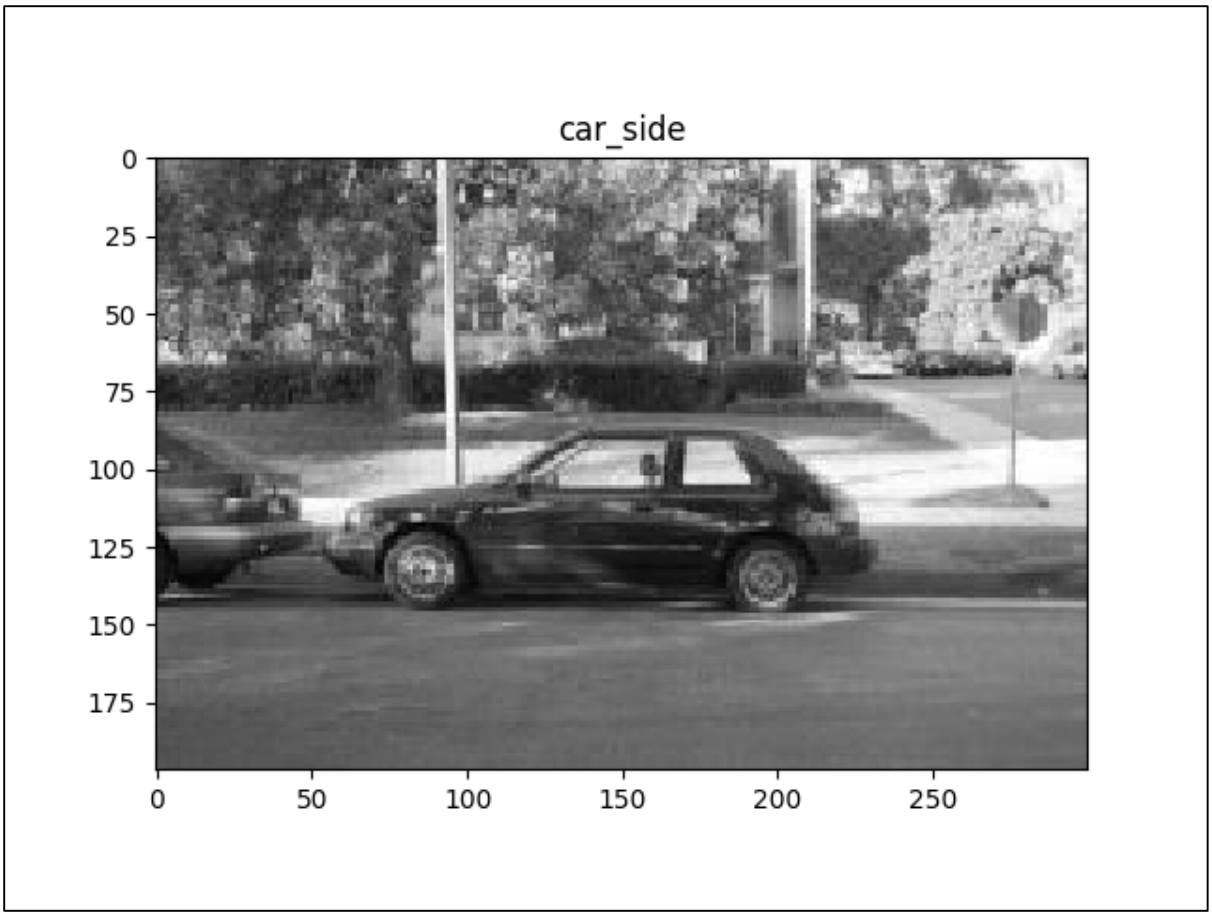
```
processing  car_side
Motorbikes
Motorbikes
car_side
car_side
car_side
car_side
car_side
car_side
car_side
electric_guitar

processing  faces
Motorbikes
car_side
faces
car_side
faces
faces
faces
electric_guitar
Motorbikes
electric_guitar

processing airplanes
faces
airplanes
airplanes
electric_guitar
Motorbikes
Motorbikes
car_side
electric_guitar
Motorbikes
Motorbikes

processing  Motorbikes
Motorbikes
Motorbikes
car_side
Motorbikes
Motorbikes
Motorbikes
car_side
Motorbikes
electric_guitar
Motorbikes

processing  electric_guitar
car_side
electric_guitar
Motorbikes
electric_guitar
electric_guitar
Motorbikes
electric_guitar
electric_guitar
Motorbikes
electric_guitar
```

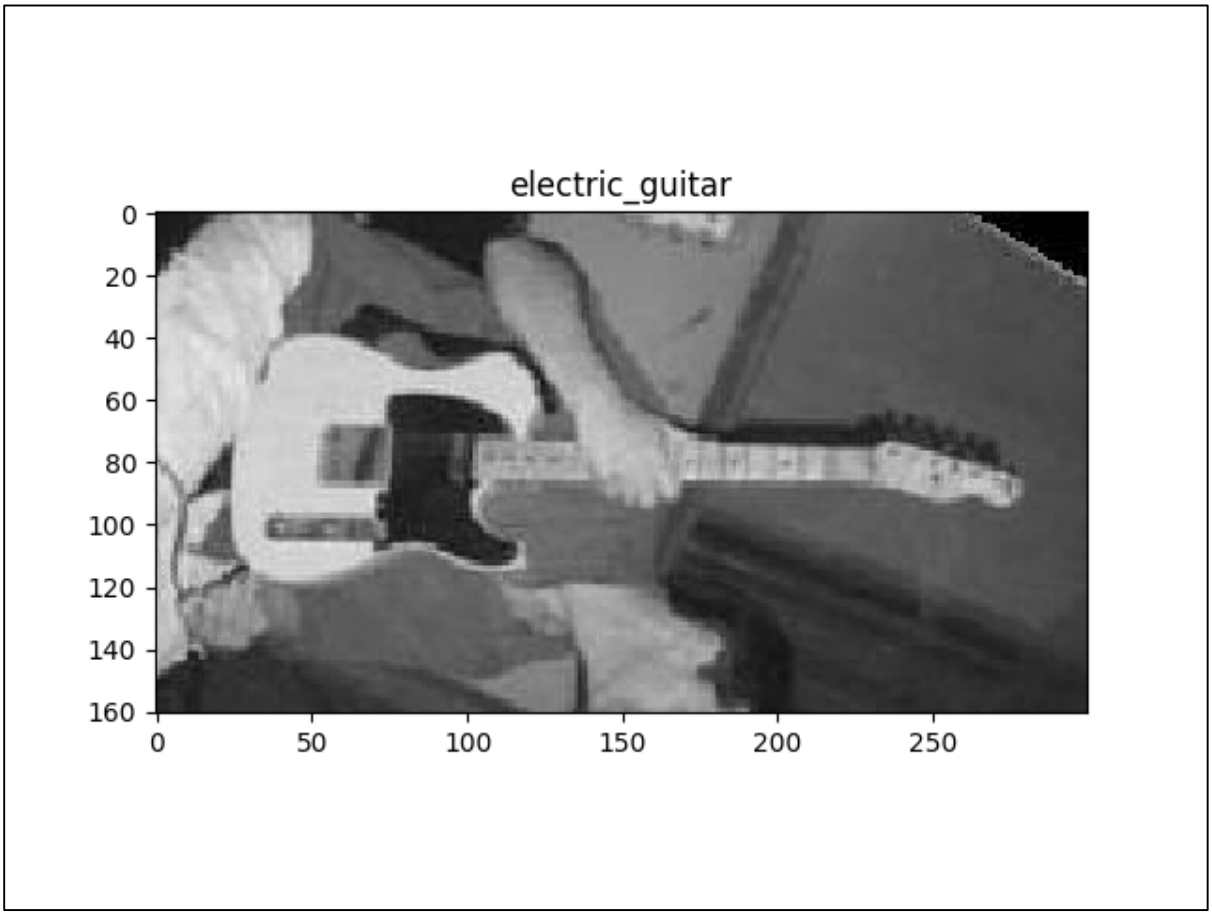
Predicted\Actual	**CAR_SIDE**	**FACES**	**AIRPLANE**	**MOTORBIKE**	**GUITARS**
CAR_SIDE	7	2	1	2	1
FACES	0	4	1	0	0
AIRPLANE	0	0	2	0	0
MOTORBIKE	2	2	4	7	3
GUITARS	1	2	2	1	6

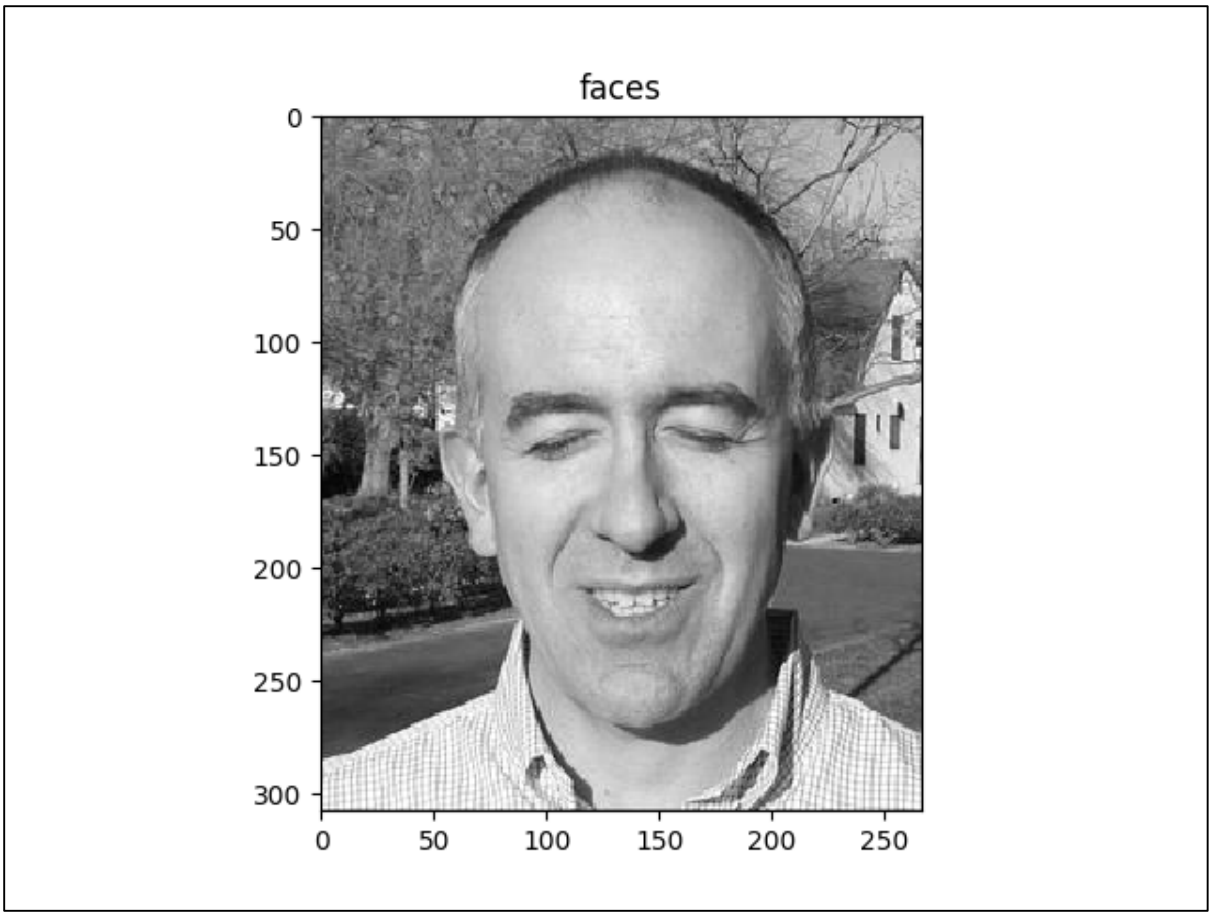


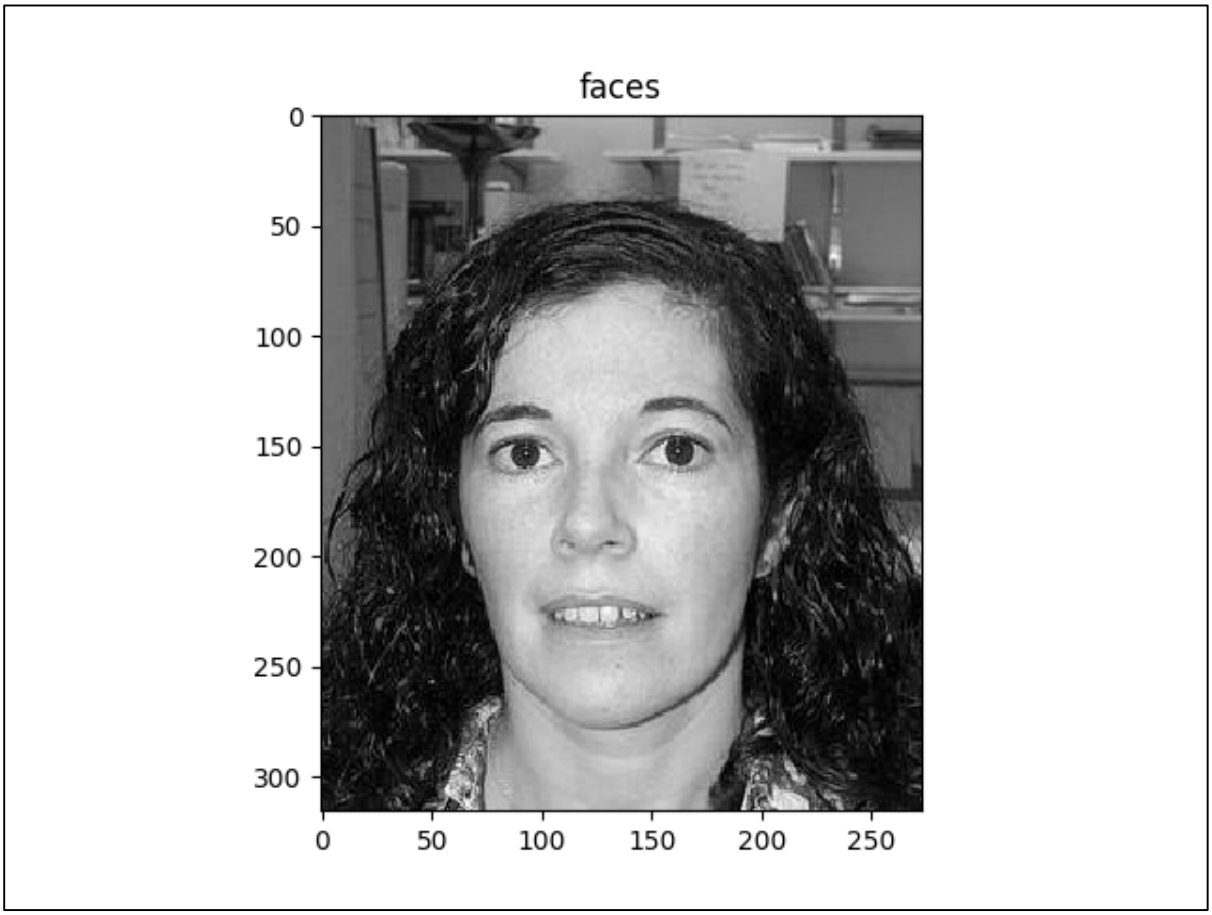
Histograms obtained for creation of the bag of features after training with upper signifying no normalisation and lower with normalisation.

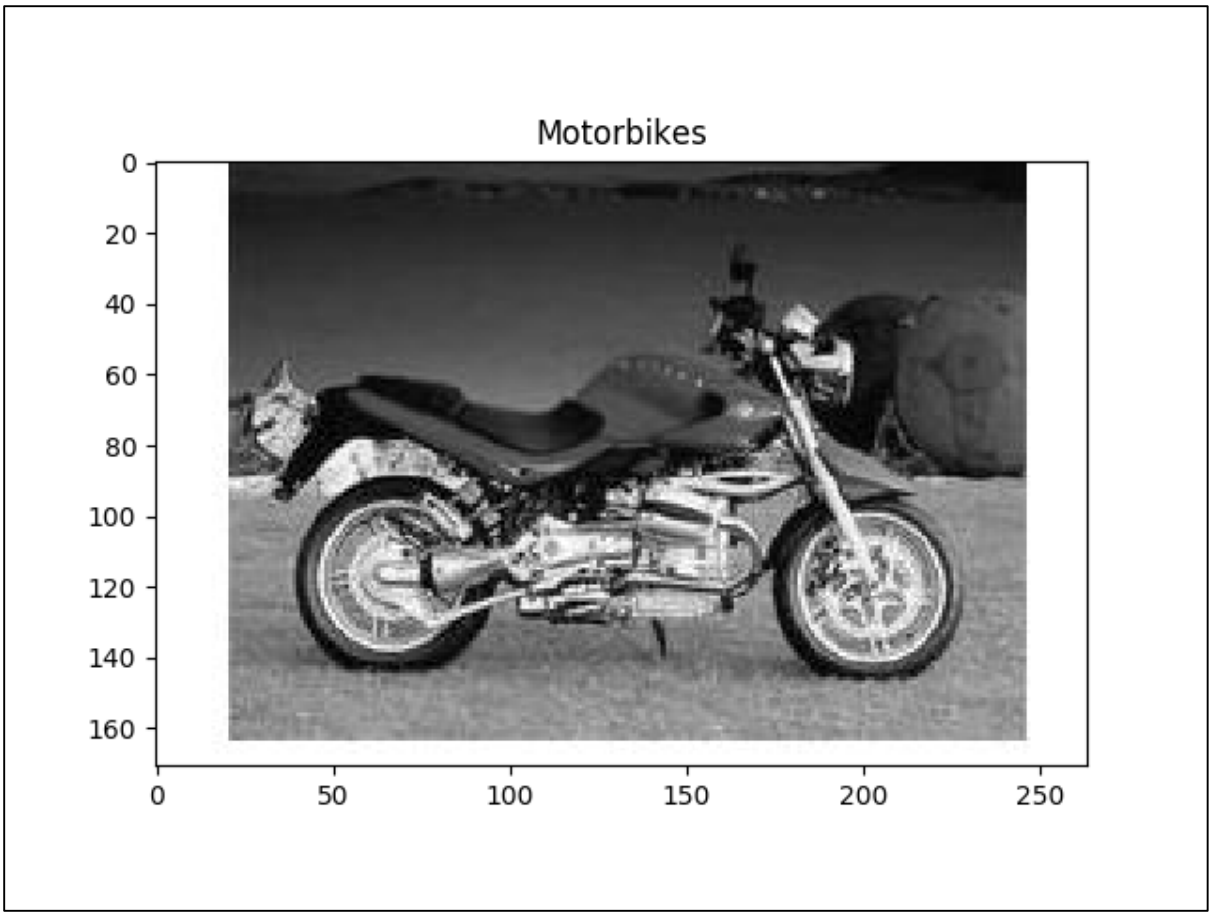
Correct predicted images

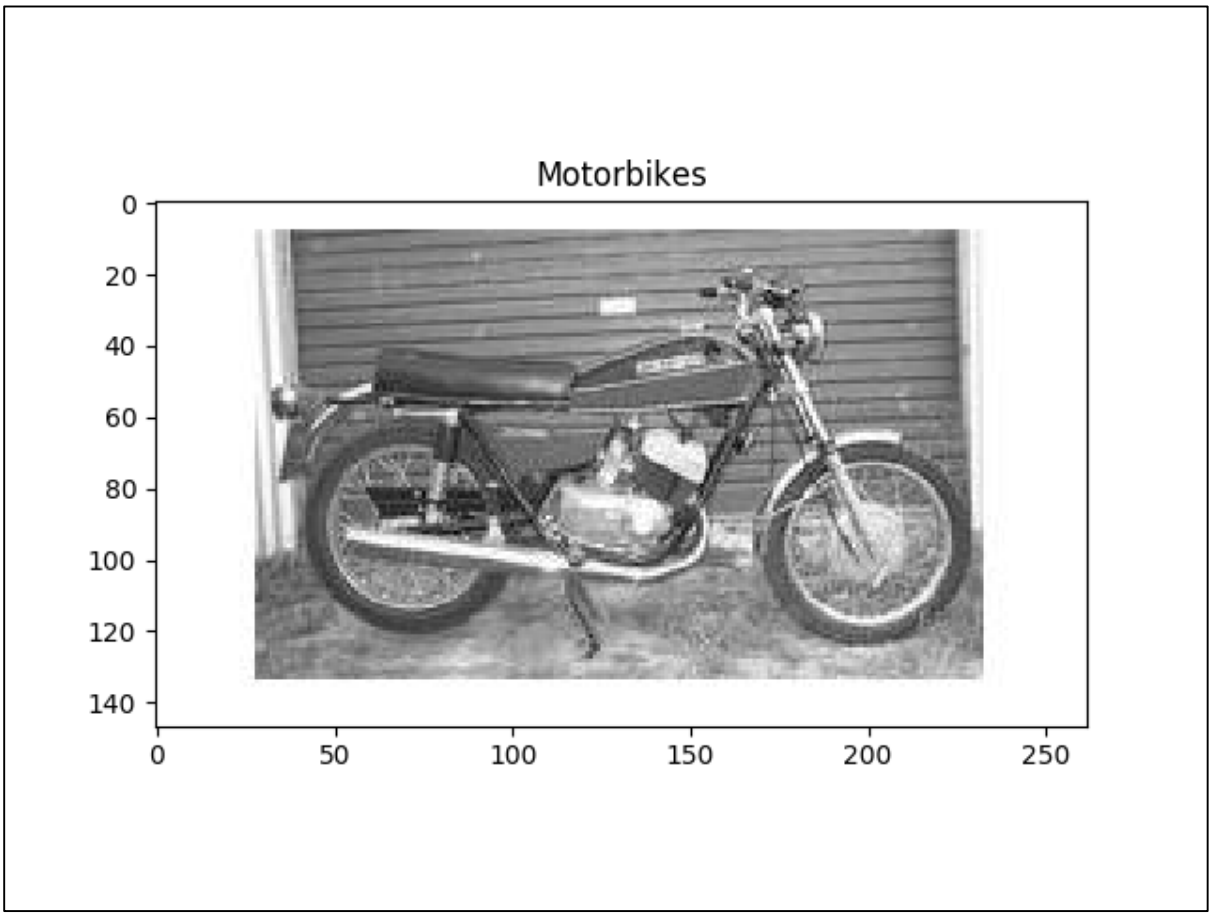

car_side
0
25
50
75
100
125
150
175
0
50
100
150
200
250

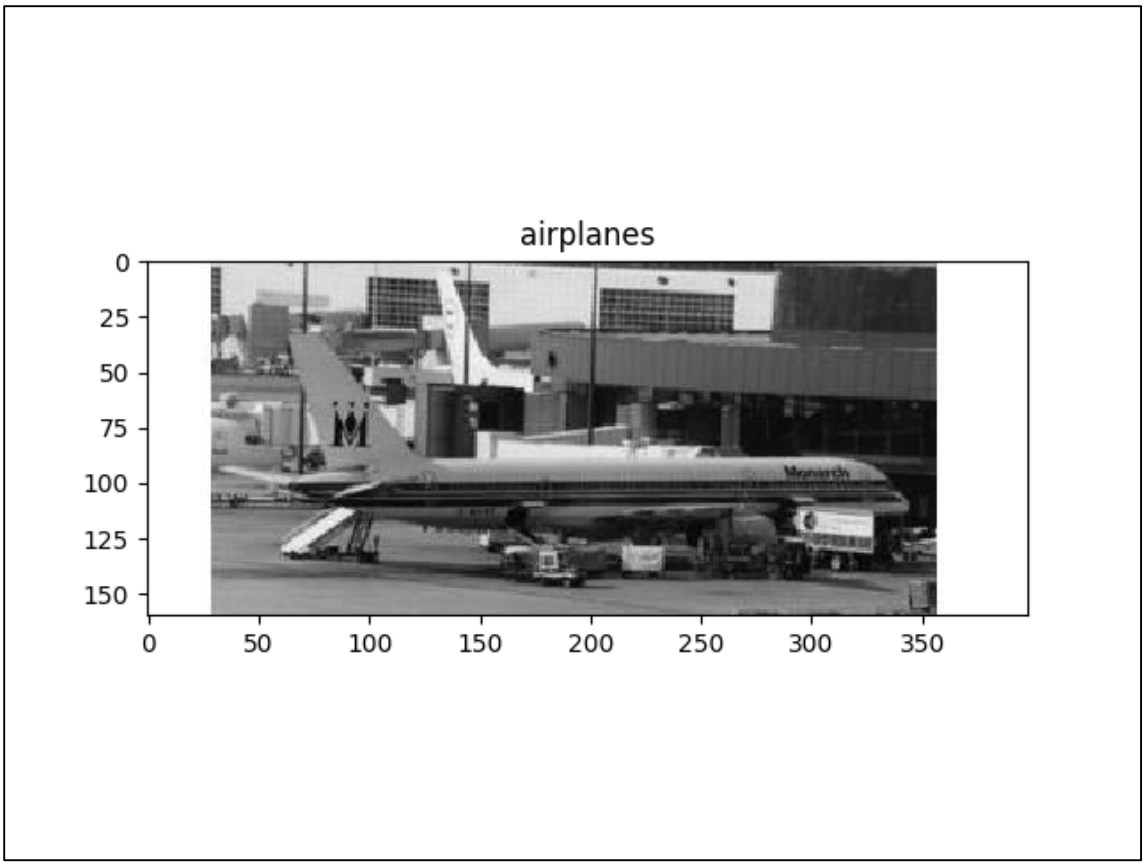

electric_guitar
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


faces
0
50
100
150
200
250
300
0
50
100
150
200
250

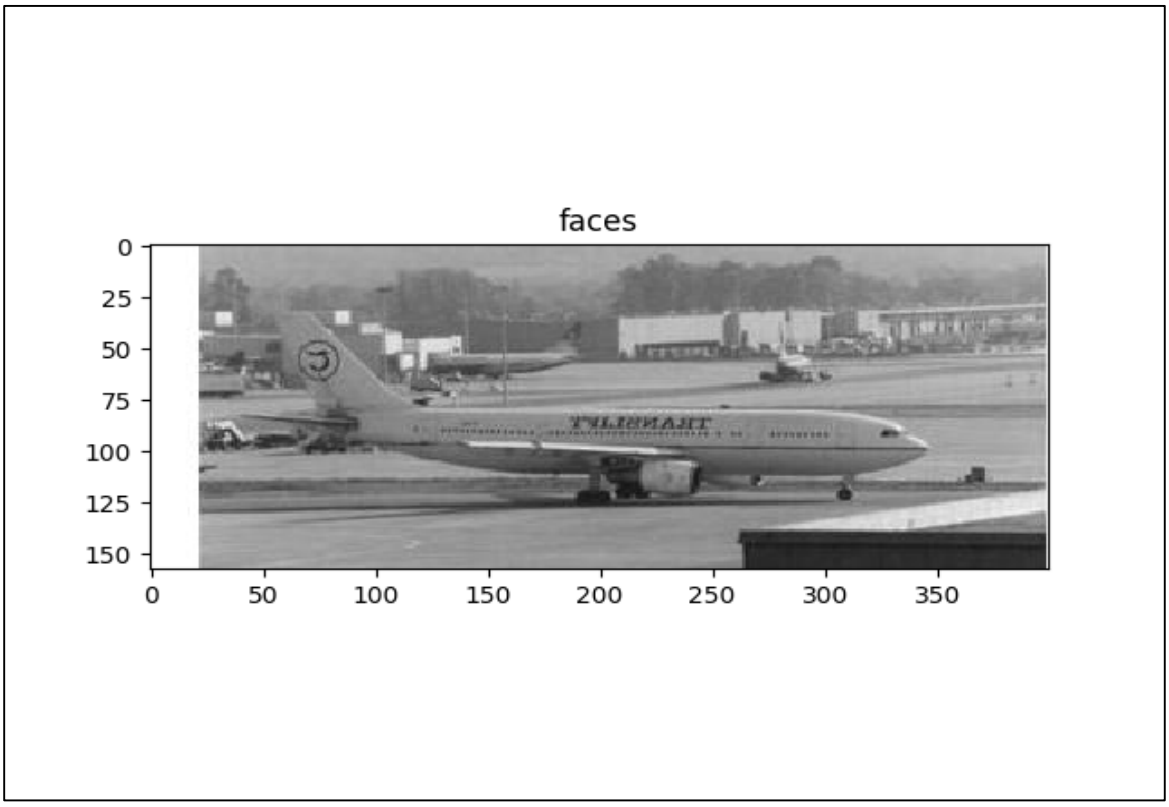

faces
0
50
100
150
200
250
300
0
50
100
150
200
250

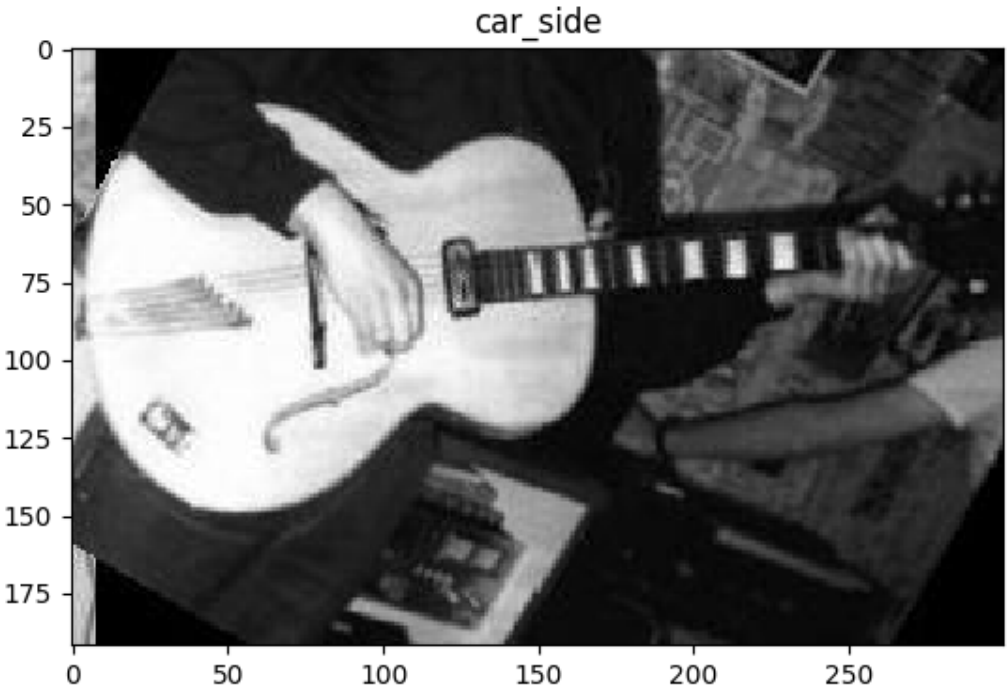

Motorbikes
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


Motorbikes
0
20
40
60
80
100
120
140
0
50
100
150
200
250

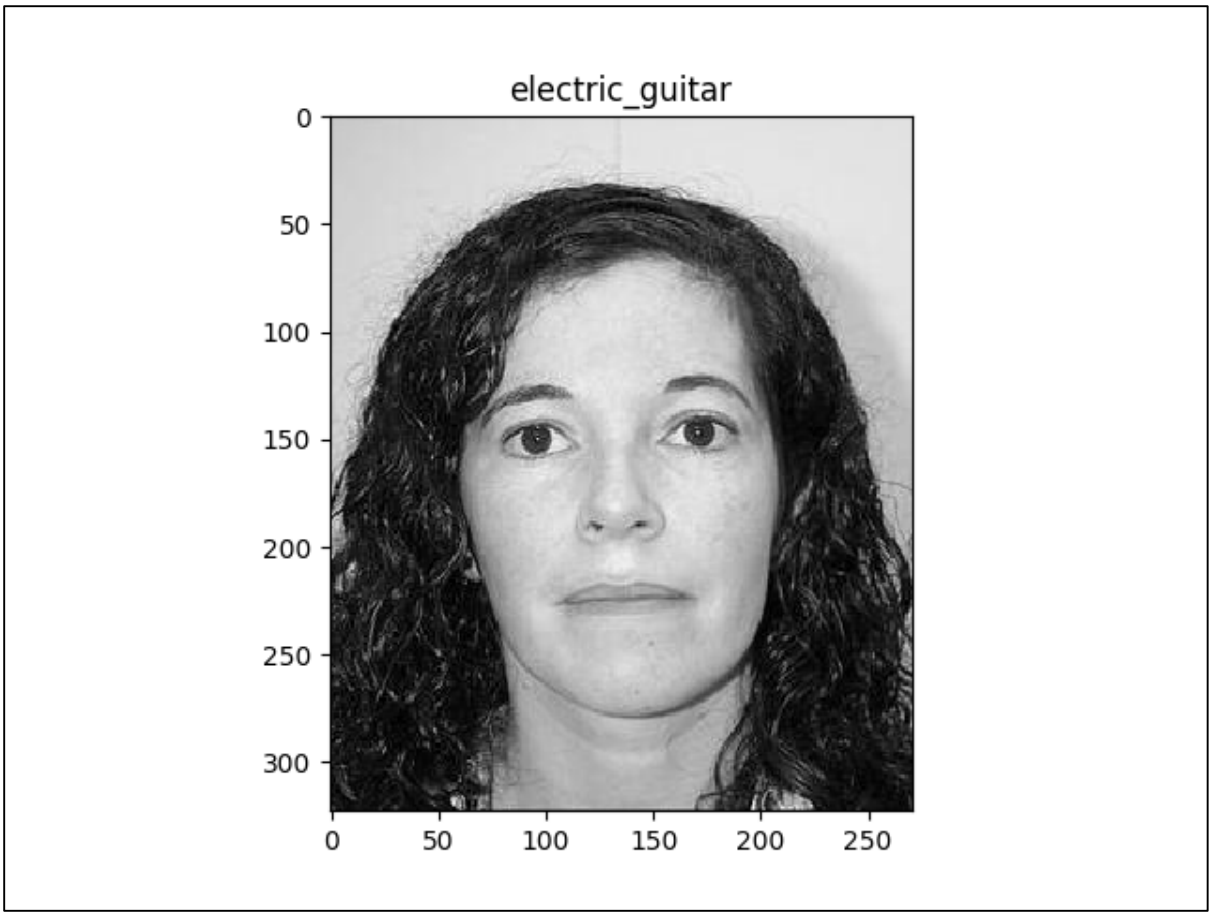


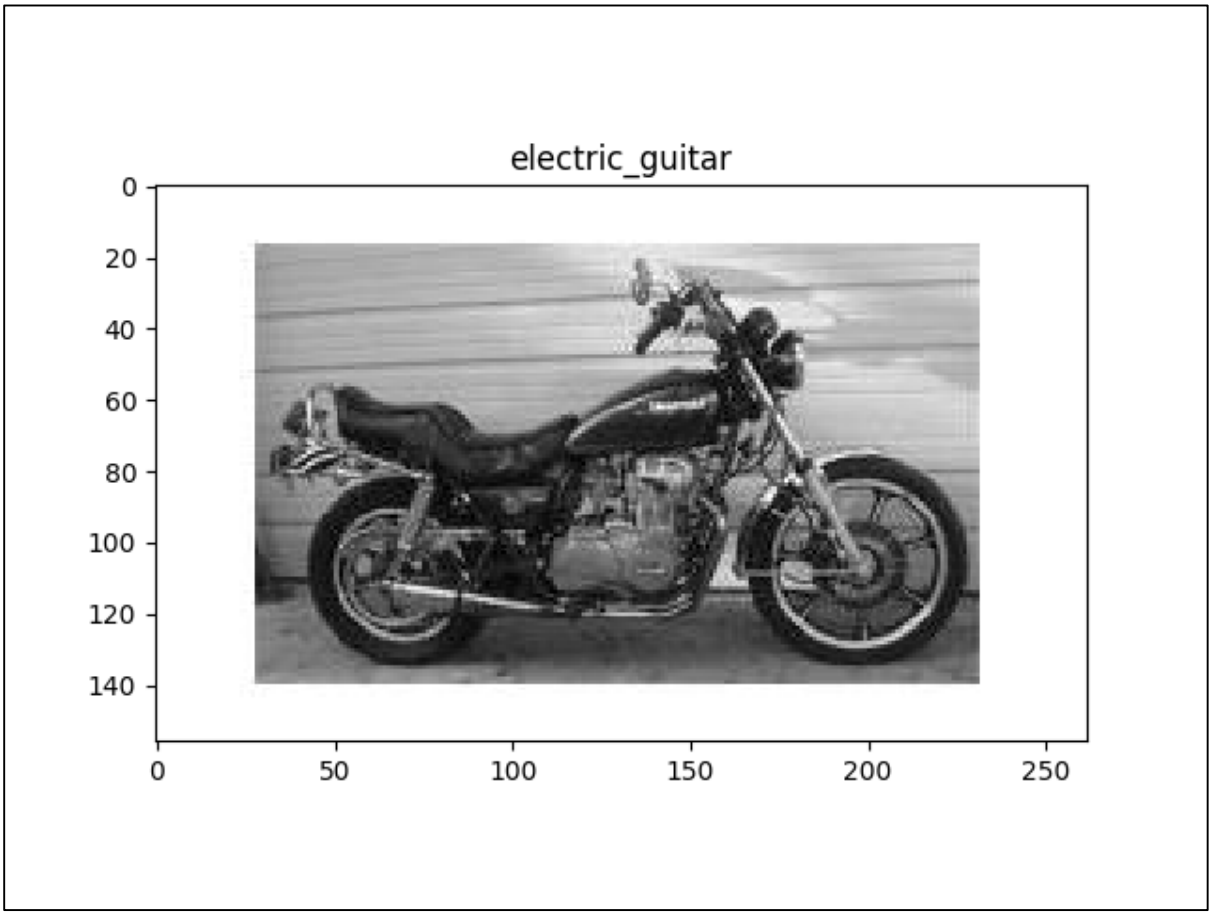
Incorrect predicted images




car_side
0
25
50
75
100
125
150
175
0
50
100
150
200
250


Motorbikes
0
50
100
150
200
250
300
0
50
100
150
200
250


electric_guitar
0
50
100
150
200
250
300
0
50
100
150
200
250


electric_guitar
0
20
40
60
80
100
120
140
0
50
100
150
200
250

ANALYSIS OF THE OBTAINED RESULTS

According to the obtained results of the confusion matrix we can clearly state the accuracy of this BoW models is not very high. It depends on a lot of parameters for its performance, which can be adjudged by the diagonal of the confusion matrix.

The results predominately depend on the parameters such as:

- Type of descriptor utilised.
- Matching criteria used for descriptors.
- Type of initial estimate in k means.
- Number of k clusters used during training.
- Amount of training images supplied.
- PCA decomposition component choice based on the essential eigen or singular values obtained after SVD.
- Type of classifier used for training the bag of features.
- Number of nearest neighbours tested for KNN algorithm.

Effects of the parameters observed in the results obtained from the above source code.

- SIFT features were utilised which provided 128 feature vectors or descriptors. These values also depend on a Hessian parameter which changes the number of key points detected and hence if fewer key points and descriptors are found, we obtain lower set of vectors which might not the correct representation of each image.
- BF matcher or Flann based techniques can be used. BF matcher was used as a default parameter, with Flann based matching just being lesser computationally expensive and increasing speed, hardly affect the outcome of the matches.
- K means algorithm is strongly dependent on the initial guess and the number of iterations that are needed for convergence. In our case we had 32815 feature vectors that we needed to be clustered in K spaces from the 100 images after application of SIFT. Initialisation was also random in this case. The results tend to change based on this initial guess. Number of iterations depend on the computation power of the machine utilised.
- Since we had 32815 features, choice of K parameter was very essential in defining the number of clusters. If K parameter was kept to 1000 or higher, larger clusters were produced with lower number of vectors in each cluster, whereas if k was kept to as low as 100 lesser clusters with more points were created. It is important to note that when K=100, the output of the confusion matrix was biased towards a particular class in certain cases. Even in the output displayed, Electric guitar class has a certain degree of bias. When k=1000, since smaller clusters were created, Overfitting issues very clearly prevalent. This makes it difficult in obtaining the decision boundary and leads to inaccuracy.
- The amount of training data presented was 100 which was again a small number and hence lead to a lower accuracy. Higher amount of training data would create a better dictionary of the code words needed for prediction.
- PCA depends completely on the singular value decomposition(SVD). The Eigen values that contribute largely to the final output are only needed to taken in to consideration and the lower values can be omitted. With SIFT producing 128 dimensional feature vectors, lots or redundancy can be eliminated by using PCA with

the correct number of Eigen values. With lower Singular values, we can easily visualise data but may eliminate a large amount of essential values and hence PCA with minimum 20 components was considered. With PCA for 40 components, hardly drastic changes were seen since only a SVD has lower Singular values of importance.

- KNN or SVM can be used with different metrics for computing the difference between the features. KNN classifier was used in this case for training. The choice of N is an important factor for the robustness of the algorithm. KNN uses metrics such as Euclidean distance using the k neighbours vote. KD tree or brute force can be used for finding the points. Brute force seems to perform decently in with the amount of training data.
- The N parameter defines the polling to the neighbours around it to decide a particular label to a specific point. Experimenting with N=10,50,100 made it possible to understand that if a smaller neighbourhood is chosen for the above example, better classification is possible. If large number of neighbours are considered like in the case of N=100, most of the test images tend to have the same label matrix and this is erroneous.

Hence with the above experiments, it can be assumed that BoW does a decent job in classification provided the above parameters discussed above are dealt with correctly.