Concept Of Bag Of Visual Words Model Implemented For Image Classification And Prediction

By Venkata Vijay Krishna Gabbula
v_gabbula@uncg.edu

***Abstract* – In Computer Vision, the concept of Bag Of Visual Words (BOVW) Model can be applied for the classification of images or retrieval of images. BOVW is used in document classification by taking a sparse vector of the occurrences of the word count, that is a sparse histogram over the vocabulary of words in a document. In this document instead of words/documents we take images and generate vectors of the features generated from the images.**

In the BOVW model, Images are treated as documents and the next step is to get the feature detection (keypoints and descriptors). For the extracted features we implement a histogram for standardized values of the total count of each feature. I use the k-means clustering algorithm for formation of clusters for the image features which will be helpful in prediction of classes at the validation phase.

In this paper I implemented BOVW model and used the support vector machine classifier (SVC) and the Random Forest Classifier (RF) in the testing phase and in the validation phase I use the predicted values of the testing phase for prediction. All the images taken in the validation phase are new images and are not used in the testing phase.

Keywords- Bag Of Visual Words, Random Forest Classifier, Support Vector Machine, K-Means Clustering Algorithm.

1. Introduction on Classification and Clustering

Classification between anything is easy task for the humans. But for the machines it is quite a tough task to predict the images as a part of machine learning. So, in the field of computer vision, it is important task for classification of images. Classification of images refers to the labeling of images into classes (any number) of predefined classes. So, there is a potential of having 'n' number of classes for a given image. For example, say if we have 100k images it is insanely difficult to manually classify the images checking all the images. So, we implement image classification techniques for image classification which is one of the important techniques in computer vision.

Some real-world examples for image classification are the use of image classification in the autonomous vehicles. For these autonomous vehicles we build a model for image classification and prediction for the objects like vehicles, pedestrians, traffic lights…. on the roads.

For image classification to be done we need to follow some of the steps like Preprocessing of the image (Initial data collection and to reduce the distortions and noisy data). Next step after preprocessing is to implement image segmentation for detection of object (detecting object of interest in a image). Next step is one of the crucial steps where we implement machine learning methods in finding the important features/patterns of the image.

We need to find particular features which are unique to certain class which will be used in the later stages of machine learning process to create a model and differentiate between classes and the process of the model learning about the features comes under the testing phase or simply called model testing.

As of now we have plenty of image classification techniques and in this paper, I am going to take a look at one such approach called as image classification with bag of visual words. Generally, in this technique of bag of visual words is used in the natural language processing and in this model the text is represented with the frequency of words occurred and not taking into account the order of words.

Images contains features that is keypoints and descriptors. Keypoints are the unique points of the image. (Unique features are those which cannot be differed by changing the image resolution, resizing, shrinking, expanding, rotating of image and always remains the same). Descriptors are nothing but the descriptions of the above said keypoints. So, the main functionality of keypoint descriptors is to describe interesting/unique/important patches/features in an image.

In this project we use the same concept of Bag of visual words but here instead of taking the words from text we use images (patches of images and there feature vectors) for image classification.

In this project I implemented a clustering algorithm for analysis of the keypoints. Clustering is basically dividing the data points into homogeneous classes(similar) or clusters. Clusters are formed by the data points and the points are in the same groups are similar as possible. Similarly, points in the other groups are as different as possible with compared to other groups. In the clustering technique we collect objects into groups based on the similarities.

The clustering algorithm will analyse similar groups of data by checking the similarities and locates the centroid for the clustered group of data points. For carrying out an effective clustering the algorithm implemented must evaluate distance between each point from centroid of the cluster in each cluster. The main objective of implementing clustering is to determine the

intrinsic grouping for a set of unlabeled data. In this project I implement a clustering technique (k-means clustering algorithm) as we have a non-labelled data of images.

2. K-Means Clustering Algorithm

K-means clustering implements vector quantization methods which is one of the most popular techniques implemented in the cluster analysis of data mining. K-means clustering algorithm computes centroids and repeats the process until optimal centroids are found, As in this project the accuracy in the prediction improves with increase in the number of clusters of k-means for each image taken.

For the k-means clustering the datapoints are assigned in such a way that the sum of the squared distances between the data points and the centroid are as small as possible.

I briefly give explanation on how k-means clustering algorithm technique works. First of all, we provide the number of clusters 'k' that is needed to be generated(that is the total number of clusters this algorithm needed). Next choose 'k' datapoints in random and assign it to each cluster. Here we briefly categorise the data based on the number of data points. After the random clusters generated we compute the centroids for these clusters.

Later we choose the ideal/optimal centroid which is the cluster that do not vary with the assigning data points. For this we calculate the sum of squared distances between the centroids and also the data points and then allocate the data points to the cluster which is the closest to other centroids and then compute the clusters by the average of clusters of all the data points.

The k-means algorithm step implements the Expectation-Maximisation strategy to solve the problem by assigning the data points to the nearest cluster and this step is used to compute centroid for each cluster.

Final output of the k-means clustering technique will be the formation of clusters with similar data points in the same cluster and different data points are in different clusters.

3. Implementation and code

In the pre-processing of images in this paper I have resized all the images into 128 x 128 pixels using cv2.resize function in python.

For all the images we perform a task of extracting the keypoints and descriptors by using cv2.BRISK function. In this project we use detectAndCompute function in the cv2 package to extract the descriptors and keypoints for every image. The code implementation is given in the above Figure1.

In the above code we use the 'os' package to handle the folders and file structures. Image classes are set in appropriate folders and so read a folder and make it a class id. Class ID is the name of the folder and then we give the class ID numbering (0,1,2).

```
import cv2
import numpy as np
import os

# extracting the training classes and storing them in a list
# Here we use folder names for class names
#(so we get the names of the labels as that of the folder names)

# Names are acerola, guava, cantaloupe (selected 3 fruits from the FIDS30 dataset)
train_image_path = '/Users/vijaykrishna/ML mini/train'
training_names = os.listdir(train_image_path)
# os.listdir() gives the list of all files in the path
# Get path to all images and save them in a list (Image_paths)
# for each image storing the class names of the image
#(Image class names are given as the folder names using 'os' package)
image_paths = []
image_classes = []
class_id = 0

# we define a function to list all names
def imglist(path):
    return [os.path.join(path, f) for f in os.listdir(path)]

# Fill the image_paths and image_classes and also add class ID number

for training_name in training_names:
    dir = os.path.join(train_image_path, training_name)
    class_path = imglist(dir)
    image_paths+=class_path
    image_classes+=[class_id]*len(class_path)
    class_id+=1
# for each image we extract the key points using cv2 package and then store them in a numpy array
# Create feature extraction and keypoint detector objects
# Create List where all the descriptors will be stored
des_list = []

# BRISK is used to perform feature based image segmentation using open cv2 function
# ORB also might work
# key point features are retrieved by the function brisk.detectAndCompute and creating a list to store
brisk = cv2.BRISK_create(30)

for image_path in image_paths:
    im = cv2.imread(image_path)
    kpts, des = brisk.detectAndCompute(im, None)
    des_list.append((image_path, des))
# data generated by the detectAndCompute function is converted into a numpy array
# Stack all the descriptors vertically in a numpy array
descriptors = des_list[0][1]
for image_path, descriptor in des_list[1:]:
    descriptors = np.vstack((descriptors, descriptor))

#kmeans works only on float, so converting integers to float
descriptors_float = descriptors.astype(float)
# k-means clustering tries to group similar kind of items in the form of clusters
#and also it finds the similarities between items and groupsthem into clusters
# collection of data points aggregated together because of similarities
# reference https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.vq.kmeans.html
# Performing k-means clustering and vector quantization
from scipy.cluster.vq import kmeans, vq

k = 200  #k means with 100 clusters gives lower accuracy for the considered fruits datasets
voc, variance = kmeans(descriptors_float, k, 1)

# Calculate the histogram of features and represent them as vector
#vq Assigns codes from a code book to observations.
im_features = np.zeros((len(image_paths), k), "float32")
for i in range(len(image_paths)):
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        im_features[i][w] += 1

# Perform Tf-Idf vectorization
nbr_occurences = np.sum( (im_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) / (1.0*nbr_occurences + 1)), 'float32')

# Scaling the words
#Standardize features by removing the mean and scaling to unit variance
#In a way normalization
from sklearn.preprocessing import StandardScaler
stdSlr = StandardScaler().fit(im_features)
im_features = stdSlr.transform(im_features)

#Train an algorithm to discriminate vectors corresponding to positive and negative training images
# Train the Linear SVM
from sklearn.svm import LinearSVC
clf = LinearSVC(max_iter=50000)  #Default of 100 is not effective as the output is not converging
clf.fit(im_features, np.array(image_classes))

#Train Random forest to compare how it does against SVM
#from sklearn.ensemble import RandomForestClassifier
#clf = RandomForestClassifier(n_estimators = 10000, random_state=30)
#clf.fit(im_features, np.array(image_classes))


# Save the SVM and RFC to be used in the validation stage
#We use the joblib package and dump the objects into a pickle file
# pickle allows the objects to be serialized to files on disk
# the pickle files generated can be deserialized back into the program while runtime
#Joblib dumps Python object into one file
import joblib
joblib.dump((clf, training_names, stdSlr, k, voc), "bovw.pkl", compress=3)

# increasing the k-means clustering values increases the accuracy.
```

Figure1: code for implementing the k-means algorithm and extracting the features for a image.

In the project I used the FIDS 30 dataset and picked 3 fruits (acerolas, guava and cantaloupe). I have taken '18' images each for the test data in prediction phase. Keypoints are selected by the package BRISK(in the OpenCV package) a pre-defined detector. After computing the descriptors and keypoints we push them into a list.

We use np.vstack function to stack the arrays(arrays here are the descriptors generated) generated(lists of all images after computing the detectors and keypoints) (of all the images) in vertical sequence i.e., row-wise.

We convert the descriptors into vectors as we need to feed them on to a support vector machine/ Random Forest Classifier. Later, we convert all the integers to

floating point numbers as the k-means only works on float (In the SciPy package we can do k-means only on the float numbers). In python we use 'SciPy' package to perform k-means clustering and the default 'k' value is 100.

Later, we implement Td-Idf (Text vectorization technique Text frequency – Inverse document frequency). This technique helps for converting the data into finite length vectors. So the Image taken has a bag of data with important features. And for the number of occurrences of the same feature in the image is counted by this vectorization technique. This technique of vectorization is greatly useful for information retrieval.

Number of occurrences of a feature in the image is extracted and stored the total number of occurrences of a particular feature in an array. So, in general we have a bunch of features of an image with the total number of times an image feature is repeated. For the values obtained we use the StandardScalar function in the sklearn python package which standardizes the features and eliminating the mean and scaling it to the unit variance. To determine the unit variance, we need to take the standard deviation of a particular feature and then divide it by the total number of entries/feature occurrences. We generally use the process of taking the standardization of each feature to reduce the distortions in plotting a histogram for the number of feature occurrences for particular image. We use the StandardScalar function on the image features generated to adjust the distorted values after vectorization of number of occurrences.

For the image features thus generated (after standardization) we plot a support vector machine (SVM) or the random forest classifier by using the sklearn package in python. Max number of iterations is 1000 for the SVM in sklearn package. We use the 'joblib' package in python to store all the objects to the disk/file and we store the objects in the pkl file (pickle file). Pickle file in python is used in serializing and deserializing a python object structure. This process helps in storing the data/objects in the 'byte stream' which can be stored in databases or transfer the data over a network.

The pickle file which I stored after serializing the objects into byte stream is the "Training file" The training file contains of objects of all the images (for all the classes considered) and then I perform a validation phase by deserializing this '.pkl' file and again implementing the same code of extracting the descriptor and keypoints from the file, get the histogram done for each feature with normalized values, Do the vectorization. (In the validation phase as we take totally new images we again consider to extract the keypoints, descriptors and do the vectorization and standardize the values to be predicted by the model.

Here in the validation phase we don't need the k-means clustering as it is already done in the training phase for the descriptors (In the validation phase we only deserialize the data from the pickle file and apply the validation set of images and test the accuracy of the model).

In the validation phase I take completely new images of the same fruits from the FIDS 30 dataset. Again, all the images are resized to 128 x 128 and I take a total of 4 images for each class (acerolas, guava and cantaloupe). I test the model constructed with 18 images for each class with 4 images in the validation phase.

We already have the predicted classes and the original image classes from the pickle file(which was generated at the test phase) and then we print the original image classes and the outputs of the classifier after prediction. Later I implement a confusion matrix to explain the true positive, true negative, false positive and false negative values and later we plot the accuracy for the predictions made by the classifier.

```
import cv2
import numpy as np
import os
import pylab as pl
from sklearn.metrics import confusion_matrix, accuracy_score
import joblib


# Load the classifier, class names, scaler, number of clusters and vocabulary
#from stored pickle file (generated during training phase)
# we unpack the contents of the pkl file
clf, classes_names, stdSlr, k, voc = joblib.load("bovw.pkl")

# Get the path of the testing image(s) and store them in a list
#test_path = 'dataset/test' # Names are Aeroplane, Bicycle, Car
test_path = '/Users/vijaykrishna/ML mini/test'  # Folder Names are of fruits (acerolas, guava, cantaloupe)
#instead of test if you use train then we get great accuracy

testing_names = os.listdir(test_path)

# Get path to all images and save them in a list
# image_paths and the corresponding label in image_paths
image_paths = []
image_classes = []
class_id = 0

#To make it easy to list all file names in a directory let us define a function

def imglist(path):
    return [os.path.join(path, f) for f in os.listdir(path)]

#Fill the empty lists with image path, classes, and add class ID number
# class ID of each class will be names of the folders
# the code is the same but we only take different images for predictions
for testing_name in testing_names:
    dir = os.path.join(test_path, testing_name)
    class_path = imglist(dir)
    image_paths+=class_path
    image_classes+=[class_id]*len(class_path)
    class_id+=1

# Create feature extraction and keypoint detector objects
    #SIFT is not available anymore in openCV
# Create List where all the descriptors will be stored
des_list = []

#BRISK is a good replacement to SIFT. ORB also works but didn;t work well for this example
brisk = cv2.BRISK_create(30)

for image_path in image_paths:
    im = cv2.imread(image_path)
    kpts, des = brisk.detectAndCompute(im, None)
    des_list.append((image_path, des))

# Stack all the descriptors vertically in a numpy array
descriptors = des_list[0][1]
for image_path, descriptor in des_list[0:]:
    descriptors = np.vstack((descriptors, descriptor))

# Calculate the histogram of features
#vq Assigns codes from a code book to observations.
from scipy.cluster.vq import vq
test_features = np.zeros((len(image_paths), k), "float32")
for i in range(len(image_paths)):
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        test_features[i][w] += 1

# Perform Tf-Idf vectorization
nbr_occurences = np.sum( (test_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) / (1.0*nbr_occurences + 1)), 'float32')

# Scale the features
#Standardize features by removing the mean and scaling to unit variance
#Scaler (stdSlr comes from the pickle file we imported at the start of the code)
test_features = stdSlr.transform(test_features)

# Until here most of the above code is similar to train phase except for kmeans clustering

#true class names are considered so they can be compared with predicted classes
true_class =  [classes_names[i] for i in image_classes]
# Perform the predictions and report predicted class names.
predictions =  [classes_names[i] for i in clf.predict(test_features)]


#Print the true class and Predictions
print ("true_class =" + str(true_class))
print ("prediction =" + str(predictions))

#To make it easy to understand the accuracy let us print the confusion matrix

def showconfusionmatrix(cm):
    pl.matshow(cm)
    pl.title('Confusion matrix')
    pl.colorbar()
    pl.show()


accuracy = accuracy_score(true_class, predictions)
print ("accuracy = ", accuracy)
cm = confusion_matrix(true_class, predictions)
print (cm)

showconfusionmatrix(cm)
print('Confusion Matrix for SVC 1000 iterations')
#print('Confusion Matrix for RF classifier for n-estimators=100')
```

Figure2: Code for implementing the validation phase with prediction for new set of images.

4. Results, Comparisons and Conclusions

I use the same algorithm/model for both the support vector machine (SVM) and the Random Forest Classifier (RF). As I have taken a smaller number of images and for only three classes, we get almost similar accuracy levels in predictions.

Accuracy values mainly depend on the number of clusters generated for each image. When I implemented 100 clusters for the images, we get a low accuracy level and we get relatively higher accuracies with increase in the number of clusters for every image.

```
In [37]: runfile('/Users/vijaykrishna/ML mini/code/train.py', wdir='/Users/vijaykrishna/ML mini/code')

In [38]: runfile('/Users/vijaykrishna/ML mini/code/validate.py', wdir='/Users/vijaykrishna/ML mini/code')
true_class =['acerolas', 'acerolas', 'acerolas', 'acerolas', 'guava', 'guava', 'guava', 'guava', 'cantaloupe',
'cantaloupe', 'cantaloupe', 'cantaloupe']
prediction =['acerolas', 'acerolas', 'guava', 'acerolas', 'cantaloupe', 'guava', 'guava', 'cantaloupe',
'cantaloupe', 'cantaloupe', 'cantaloupe', 'cantaloupe']
accuracy =  0.75
[[3 0 1]
 [0 4 0]
 [0 2 2]]
Confusion Matrix for SVC 1000 iterations

In [52]: runfile('/Users/vijaykrishna/ML mini/code/validate.py', wdir='/Users/vijaykrishna/ML mini/code')
true_class =['acerolas', 'acerolas', 'acerolas', 'acerolas', 'guava', 'guava', 'guava', 'guava', 'cantaloupe',
'cantaloupe', 'cantaloupe', 'cantaloupe']
prediction =['acerolas', 'acerolas', 'guava', 'acerolas', 'guava', 'guava', 'guava', 'acerolas', 'cantaloupe',
'guava', 'cantaloupe', 'cantaloupe']
accuracy =  0.75
[[3 0 1]
 [0 3 1]
 [1 0 3]]
Confusion Matrix for RF classifier for n-estimators=100
```

Figure3: Output and accuracies after prediction for both SVM and RF classifier for '100' clusters on every image (SVC→1000 iterations and RF classifier →100 estimates).

```
In [44]: runfile('/Users/vijaykrishna/ML mini/code/validate.py', wdir='/Users/vijaykrishna/ML mini/code')
true_class =['acerolas', 'acerolas', 'acerolas', 'acerolas', 'guava', 'guava', 'guava', 'guava', 'cantaloupe',
'cantaloupe', 'cantaloupe', 'cantaloupe']
prediction =['acerolas', 'acerolas', 'acerolas', 'acerolas', 'guava', 'guava', 'guava', 'guava', 'cantaloupe',
'guava', 'cantaloupe', 'cantaloupe']
accuracy =  0.9166666666666666
[[4 0 0]
 [0 3 1]
 [0 0 4]]
Confusion Matrix for RF classifier for n-estimators=100

In [2]: runfile('/Users/vijaykrishna/ML mini/code/validate.py', wdir='/Users/vijaykrishna/ML mini/code')
true_class =['acerolas', 'acerolas', 'acerolas', 'acerolas', 'guava', 'guava', 'guava', 'guava', 'cantaloupe',
'cantaloupe', 'cantaloupe', 'cantaloupe']
prediction =['acerolas', 'acerolas', 'guava', 'acerolas', 'acerolas', 'guava', 'guava', 'guava', 'cantaloupe',
'cantaloupe', 'cantaloupe', 'cantaloupe']
accuracy =  0.8333333333333334
[[3 0 1]
 [0 4 0]
 [1 0 3]]
Confusion Matrix for SVC 1000 iterations
```

Figure4: Output and accuracies after prediction for both SVM and RF classifier for '200' clusters on every image (SVC→1000 iterations and RF classifier →100 estimates).

I used the matplotlib.pyplot.matshow() in python to display the confusion matrix outputs. Matshow() function uses the data in the array and display it in the form of matrix as a new figure window. I have only included two of the confusion matrices (included RF classifier for k=100 and SVM for k=200).

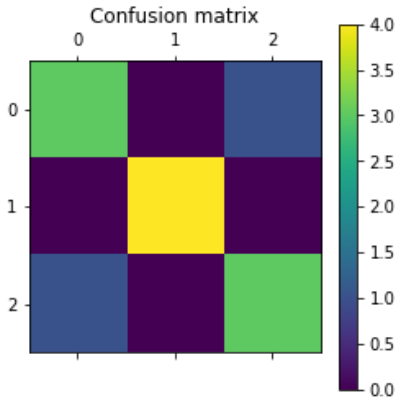

Figure5: Confusion matrix for SVC(iterations→1000) of figure4 here accuracy=83% for 200 clusters.

The figures 3 and 4 are the output predictions for the three class images and in validation phase I used 4 images for each class. Figures 5 and 6 are the confusion matrices outputs for RF and SVM

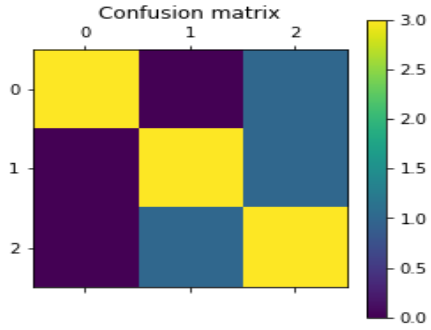

Figure6: Confusion matrix for RF classifier for k=100 and estimators=100 (accuracy=75%)

From the outputs I say that the random forest classifier fairs well in comparison to that of the Support Vector Machine. Initial class and the prediction classes are shown in figures 3 and 4. For same parameters RF classifier and SVM has the same accuracy of 75% when the number of clusters 'k' set to 100 clusters per image. But when I increase the 'k' value from 100 to 200 we can get a difference in the prediction accuracies.

I even tried for more values of clustering and also for more iterations and increasing n- observations for SVM and RF respectively, but there are no significant changes in the accuracy values at the prediction.

For the above project I Implemented, we can Increase the range (no of images) and also we can implement for more number of labels (more types of images). This process is quick simple and fast in prediction if the images are of less resolution.

References:

[1]. For implementation of cv2 BRISK
https://stackoverflow.com/questions/62571115/how-to-implement-brisk-using-python-and-opencv-to-detect-features

[2]. Bag of visual words model
https://machinelearningknowledge.ai/image-classification-using-bag-of-visual-words-model/

[3]. K-means clustering
https://www.analyticsvidhya.com/blog/2021/11/understanding-k-means-clustering-in-machine-learningwith-examples/

[4]. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
[5]. Image classification techniques
https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/
[6]. Usage of pickle package in python
https://docs.python.org/3/library/pickle.html
[7]. joblib package
https://joblib.readthedocs.io/en/latest/generated/joblib.dump.html
[8]. StandardScalar sklearn package
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[9]. vectorization technique
https://towardsdatascience.com/text-vectorization-term-frequency-inverse-document-frequency-tfidf-5a3f9604da6d
[10]. Code and implementation of concept of visual bag of words model
https://github.com/bnsreenu/python_for_microscopists/blob/master/069a-Train_BOVW_V1.0.py