Image Feature Detection and Matching Classification

Zuha Tassawar Hussain

MS-AI, SEECS.

Abstract

This paper explores the Bag of Visual Words (BoVW) technique for image classification, which was a classical method prior to the advent of deep learning models. While deep learning has achieved impressive results in image classification, BoVW remains a relevant and effective approach. The paper discusses the underlying principles of BoVW, how it works, and its advantages and limitations. The BoVW approach involves breaking down an image into small patches, identifying the most frequently occurring visual words in these patches, and creating a histogram of the visual words for each image. These histograms are then used as features for classification using traditional machine learning algorithms.
The Bag of Visual Words (BoVW) model is a crucial concept in computer vision for image classification. BoVW is highly scalable and accurate, making it ideal for developing Content-Based Image Retrieval (CBIR) systems. BoVW is also useful for texture classification using textons. This paper aims to explore the underlying principles and techniques of BoVW.

Keywords: Bag of Visual Words, Feature Extraction, SIFT, SVM

1 Introduction

The concept of "Bag of Visual Words" is derived from the concept of "Bag of Words" from the Natural Language Processing Domain. In Bag of Words, the text is represented with it's occurrence frequency without taking into account the order of the words (Hence "Bag" and not list or array). Documents that share a large number of same keywords are considered to be similar or *relevant* to each other. Treating any textual file or document as a "bag of words" is a

more efficient method to compare and analyze because the locality of words does not have to be stored and taken to account.

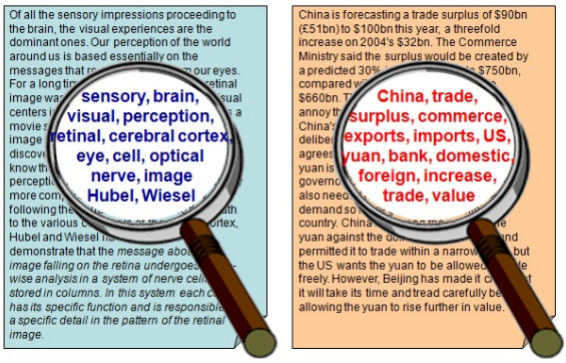

Fig. 1 A bag of Words model which disregards the order of words entirely and just counts the frequency of occurrence of each word

In Computer Vision, the same concept can be applied for images as well. Our "words" are now patched of images along with their feature descriptors.

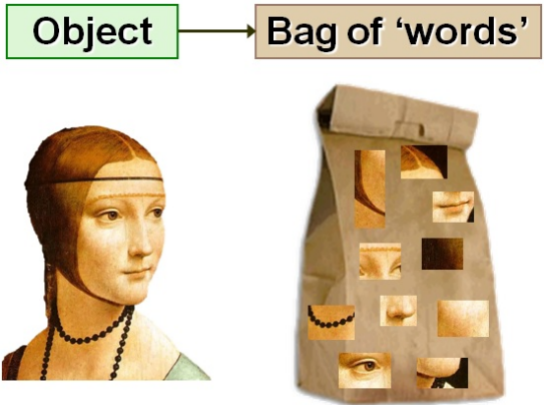

Fig. 2 For Images we count the number of times each image patch occurs

In Natural Language Processing, recording the frequency of a word in a document is an easy task done by a histogram but in Computer Vision, how do you count the number of times an image patch has occurred in an image? And how do you ensure that vocabulary is same for all images throughout the entire dataset?
The bag of words model can be utilized in object categorization by creating

an extensive vocabulary of visual words and creating a histogram that represents each image's frequency of occurrence of the words in the image. The accompanying figure 13 provides a visual representation of this concept.

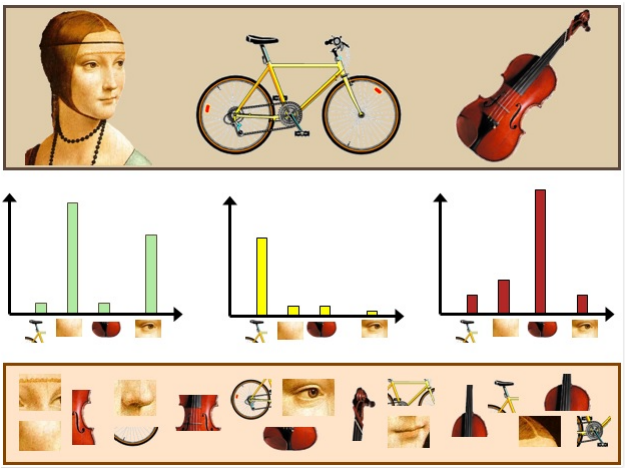

Fig. 3 Three images (top), extracting image patches (middle) and number of times each patch occurred (bottom)

In this report, we will be doing Image classification. This will be done in three stages.

- Represent each training image by a vector
- Train a classifier to discriminate vectors corresponding to positive and negative training images
- Apply the trained classifier to the test image

2 Building a Bag of Visual Words

Visual Words are "iconic" image patches or fragments. There are three basic steps that are involved in building a bag of visual words.

- Feature Extraction
- Code-book Generation
- Vector Quantization

2.1 Feature Extraction

The first step is to extract feature descriptors from each image in our dataset. This feature extraction can be done in a variety of ways. One possible approach for feature extraction is to detect salient regions in the images and extract SIFT features from them. Another method is to apply a grid at regular intervals using the Dense key point detector and extract another type of local invariant

descriptor. Alternatively, random locations in the images can be used to extract the mean RGB values. For each image given as input, an output of multiple features is received.

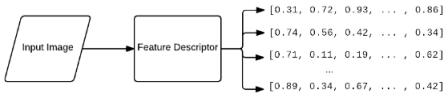

Fig. 4 Multiple feature vectors per image

Local descriptors are widely used in computer vision for recognition and matching, as they are robust to partial visibility and clutter. These descriptors are computed over limited spatial support and need to be repeatable, meaning that if there is a transformation between two instances of an object, corresponding points should be detected, and ideally, identical descriptor values should be obtained around each point. This has led to the development of various scale and affine invariant point detectors and descriptors that are resistant to geometric and illumination variations.[4]-[8].

In [8], scale invariant point detector is not sufficient to have stability of point's location if we have some kind of affine transformations between images. That's why Harris offline detector [8] was a preferable choice but the benefits are not clear cut. This maybe due to multiple reasons i.e the real world is three dimensional and affine transformations do not capture its variations accurately and any attempt to increase invariance of a feature results in loss of information that might have been important.

Harris affine points are detected by an iterative process. Firstly, positions and scales of interest points are determined as local maxima (in position) of a scaleadapted Harris function, and as local extrema in scale of the Laplacian operator. Then an elliptical (i.e. affine) neighborhood is determined. This has a size given by the selected scale and a shape given by the eigenvalues of the image's second moment matrix. The selection of position/scale and the elliptical neighborhood estimation are then iterated and the point is kept only if the process converges within a fixed number of iterations.

The affine region is then mapped to a circular region, so normalizing it for affine transformations. Scale Invariant Feature Transform (SIFT) descriptors [5] are computed on that region. SIFT descriptors are multi-image representations of an image neighborhood. They are Gaussian derivatives computed at 8 orientation planes over a 4x4 grid of spatial locations, giving a 128-dimension vector. Figure 5 shows an example of the maps of gradient magnitude corresponding to the 8 orientations.

Fig. 5 (from left to right) Harris affine region; the normalized region; 8 maps of gradient magnitudes constituting SIFT descriptor

We prefer SIFT descriptors to alternatives such as steered Gaussian derivatives or differential invariants of the local jet for the following reasons:

- They are simple linear Gaussian derivatives. Hence we expect them to be more stable to typical image perturbations such as noise than higher Gaussian derivatives or differential invariants.
- The use of a simple Euclidean metric in the feature space seems justified. In the case of differential invariants obtained by a combination of the components of the local jet, the use of a Mahalanobis distance is more appropriate. For instance, one would expect a second derivative feature to have a higher variance than a first derivative. Selecting an appropriate Mahalanobis distance a priori seems challenging. It would not be appropriate to use the covariance matrix of SIFT descriptors over the entire dataset, since this is predominantly influenced by inter-class variations (or more precisely, by variations between keypoints that we do not wish to ignore). Measuring a Mahalanobis distance would probably requiring manual specification of multiple homologous matching points between different images of objects of the same category, seriously working against our objective of producing a simple and automated categorization system. [10]
- There are far more components to these feature vectors (128 rather than 12 to 16). Hence we have a richer and potentially more discriminative representation.

Mikolajczyk et al [9] have compared several descriptors for matching and found that SIFT descriptors perform best.

2.2 Code-book Generation or Vocabulary Construction

Vocabulary is a way of constructing a feature classifier that gives new descriptors for query images for descriptors that were previously seen in training. One extreme approach is to compare each query descriptor to every descriptor seen in training. But this approach is quite time-taking and impractical due to a large number of training descriptors. Another extreme method is to identify a small number of large clusters that are good a separating a particular class. Most clustering algorithms are based on iterative square error or hierarchical techniques. We here use the simplest square-error partitioning method called *K-Means*.[1]

2.2.1 K-Means Clustering

This algorithm proceeds by iterated assignments of points to their closest cluster centers and recomputation of the cluster centers. Two difficulties are that the k-means algorithm converges only to local optima of the squared distortion, and that it does not determine the parameter k. There exist methods allowing to automatically estimating the number of clusters. For example, Pelleg et al [2] use cluster splitting to do it, where the splitting decision is done by computing the Bayesian Information Criterion. However, in the present case we do not really know anything about the density or the compactness of our clusters. Moreover, we are not even interested in a "correct clustering" in the sense of feature distributions, but rather in accurate categorization. We therefore run k-means several times with different number of desired representative vectors (k) and different sets of initial cluster centers. We select the final clustering giving the lowest empirical risk in categorization.[3]

2.3 Vector Quantization

For any image, we can represent the image using our bag of visual words through this process:

- Extract features from an image using the same feature extractor.
- For every extracted feature vector, use a distance metric such as Cosine Similarity to computer its nearest neighbour in the code-book.
- From the set of nearest neighbours, make a histogram of length k (where k= number of clusters), the *i-th* value in the histogram is the frequency of the *i-th* visual word.

The above process in which you get an abstract model of an object through the distribution of it's prototype vectors is called **vector quantization**.

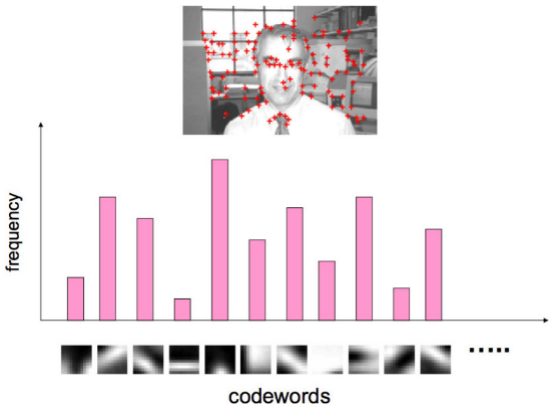

Fig. 6 Vector Quantization

3 Classification

The above histograms are then used has an input to a classification model. Here we will be using Support Vector Machines(SVM) and Random Forest Classifier.

3.1 Classification by SVM

The SVM classifier finds a hyperplane which separates two-class data with maximal margin [3]. The margin is defined as the distance of the closest training point to the separating hyperplane. For given observations **X**, and corresponding labels **Y** which takes values 1, one finds a classification function:

$$f(x) = \text{sign}(w^T x + b) \tag{1}$$

where **w** and **b** represent the hyperplane parameters. Data sets are not always linearly separable. The SVM takes two approaches to this problem. Firstly it introduces an error weighting constant **C** which penalizes mis-classification of samples in proportion to their distance from the classification boundary. Secondly a mapping is made from the original data space of **X** to another feature space. This second feature space may have a high or even infinite dimension. One of the advantages of the SVM is that it can be formulated entirely in terms of scalar products in the second feature space, by introducing the *kernel*

$$K(u, v) = \Phi(u) \cdot \Phi(v) \tag{2}$$

Both the kernel **K** and penalty **C** are problem dependent and need to be determined by the user.

In the kernel formulation, the decision function can be expressed as

$$f(x) = \text{sign}\left(\sum_i y_i \alpha_i K(x_i, x) + b\right) \tag{3}$$

where x_i are the training features from data space **X** and y_i is the label of x_i. Here the parameters α_i are typically zero for most *i*. Equivalently, the sum can be taken only over a select few of the x_i . These feature vectors are known as *support vectors*. It can be shown that the support vectors are those feature vectors lying nearest to the separating hyperplane. In our case, the input features x_i are the binned histograms formed by the number of occurrences of each keypoint V_i from the vocabulary **V** in the image I_i.

In order to apply the SVM to multi-class problems we take the one-against-all approach. Given an *m*-class problem, we train *m* SVM's, each distinguishes images from some category *i* from images from all the other *m-1* categories *j* not equal to *i*. Given a query image, we assign it to the class with the largest SVM output.

3.2 Classification Done by Random Forest

In a random forest classification, multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree, then taking the most popular result. (For regression, predictions use an averaging technique instead.)

Random Forest classification begins by constructing a large number of decision trees on randomly selected subsets of the training data. Each decision tree is constructed using a random subset of features, and splits on the feature that maximizes the separation between the classes. The resulting tree is then pruned to reduce overfitting.

Once all the decision trees have been built, their outputs are combined to make a final prediction. This can be done using a majority vote, where each tree's prediction is counted as a vote for the corresponding class. The class with the most votes is then selected as the final prediction. The equation for combining the outputs of the decision trees is:

$$\hat{y} = \text{mode} f_1(x), f_2(x), ..., f_n(x) \tag{4}$$

where is the final prediction, $f_i(x)$ is the prediction of the *i-th* decision tree, and mode is the function that returns the most common value.

To evaluate the performance of the Random Forest classifier, the prediction error is calculated. This can be done using various metrics such as accuracy, precision, recall, or F1 score. One commonly used metric is the mis-classification error rate, which is defined as the proportion of mis-classified instances. The equation for mis-classification error rate is:

$$\text{Mis-classification Error Rate} = \frac{\text{Number of Mis-classified Instances}}{\text{Total Number of Instances}} \tag{5}$$

where the total number of instances is the sum of correctly classified and mis-classified instances.

4 Results

We present the results from experiments done on two datasets. Some images from Flower (Figure 7) and Object Dataset (Figure 8) are shown below.

Both ORB and SIFT were applied on both datasets for feature extraction. With ORB and K-Nearest Neighbour as a classifier gave a very poor accuracy of 27.03% for Flower dataset but 62.5% for Object dataset. SVM gave an accuracy of 33% on Flower dataset while on object dataset it gave 84.375% accuracy.

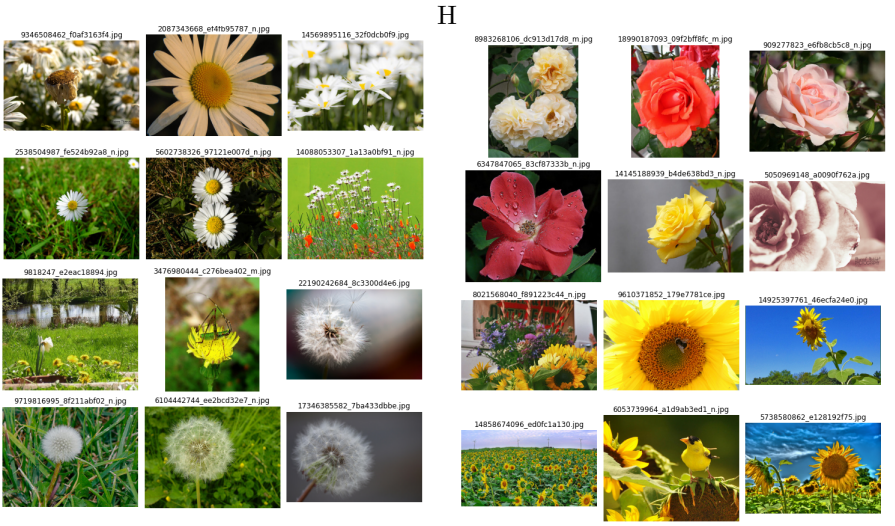

Fig. 7 Example images from the Flower Dataset

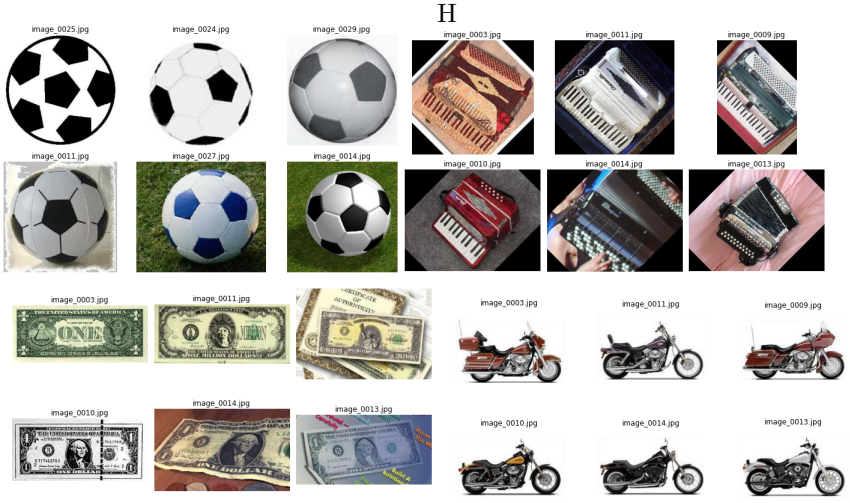

Fig. 8 Example images from the Objects Dataset

	precision	recall	f1-score	support
daisy	0.26	0.17	0.20	127
sunflowers	0.41	0.34	0.37	180
dandelion	0.19	0.11	0.14	129
tulips	0.36	0.31	0.33	140
roses	0.33	0.64	0.43	160
accuracy			0.33	736
macro avg	0.31	0.31	0.29	736
weighted avg	0.32	0.33	0.31	736

Fig. 9 SVM on ORB for Flower Dataset; Polynomial kernel

	precision	recall	f1-score	support
daisy	0.27	0.16	0.20	127
sunflowers	0.42	0.52	0.47	180
dandelion	0.29	0.09	0.14	129
tulips	0.36	0.36	0.36	140
roses	0.34	0.56	0.43	160
accuracy			0.36	736
macro avg	0.34	0.34	0.32	736
weighted avg	0.34	0.36	0.33	736

Fig. 10 SVM on ORB for Flower Dataset; linear kernel

	precision	recall	f1-score	support
daisy	0.55	0.28	0.37	127
sunflowers	0.40	0.79	0.53	180
dandelion	0.38	0.29	0.33	129
tulips	0.51	0.44	0.47	140
roses	0.41	0.24	0.31	160
accuracy			0.43	736
macro avg	0.45	0.41	0.40	736
weighted avg	0.45	0.43	0.41	736

Fig. 11 SVM on SIFT for Flower Dataset; Polynomial kernel

```
[ ] a_test_image = test["accordian"][0]
    p = make_prediction(a_test_image, rfc)
    list(test.keys())[p]

    'accordian'

[ ] a_test_image = test["motorbike"][0]
    p = make_prediction(a_test_image, rfc)
    list(test.keys())[p]

    'accordian'

[ ] a_test_image = test["dollar_bill"][0]
    p = make_prediction(a_test_image, rfc)
    list(test.keys())[p]

    'dollar_bill'
```

h!

Fig. 12 Object Dataset; SIFT and Random Forest

From these classification reports, we can calculate the true positive and false positive.

$$TruePositive = \text{precision} \times \text{positive instances} \tag{6}$$

$$FalsePositive = (1 - \text{precision}) \times \text{negative instances} \tag{7}$$

$$TrueNegative = \text{recall} \times \text{negative instances} \tag{8}$$

$$FalseNegative = (1 - \text{recall}) \times \text{positive instances} \tag{9}$$

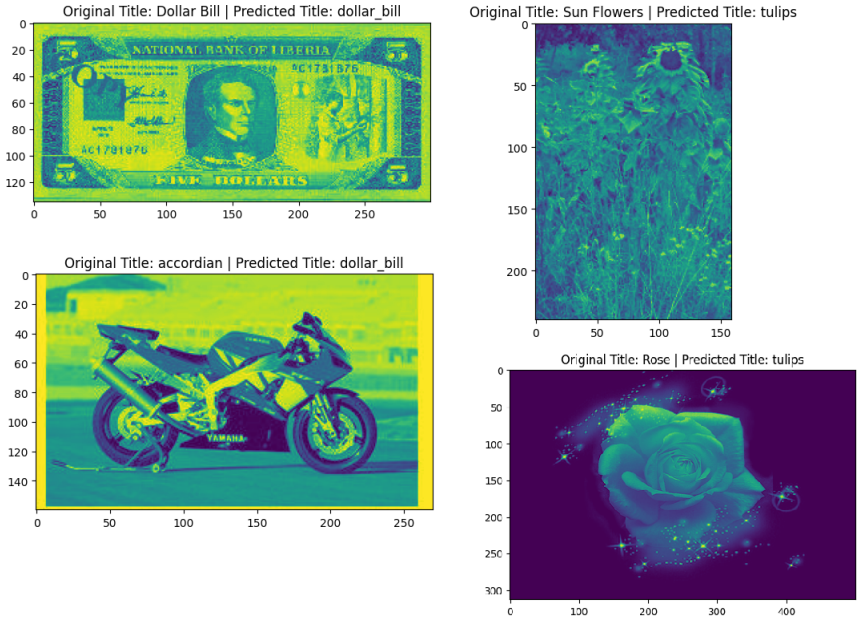

Fig. 13 Some Qualitative Results

Observations

SVM with SIFT on Objects Dataset gives an accuracy of 75% while on Flower Dataset it gives an accuracy of 30%. These results show that SVC classifier did great on Objects Dataset with an accuracy of 87% and Random Forest got 62.5% accuracy. But on Flowers dataset, it was giving an accuracy of less than 50% with either of these classifiers and even K-Nearest Neighbours.

There could be several reasons why the SVC and Random Forest classifiers did not perform well on the flower dataset:

- **Dataset size and diversity**: The flower dataset may be smaller or less diverse than the object dataset, which could make it harder for the classifiers to generalize well to new data. If the dataset is too small, the classifiers may not have enough data to learn from and could be prone to overfitting.
- **Complexity of the data**: The flower dataset may be more complex than the object dataset, which could make it harder for the classifiers to accurately distinguish between different classes. For example, different flower species may have similar visual features, which could lead to mis-classifications.

References

[1] O. Duda, P.E. Hart, D.G. Stork, Pattern classification, John Wiley Sons, 2000.

[2] D. Pelleg and A. Moore. X-Means: Extending K-means with Efficient Estimation of the Number of Clusters, International Conference on Machine Learning, 2000.

[3] V. Vapnik. Statistical Learning Theory. Wiley, 1998

[4] T. Lindenberg, Scale-space theory in computer vision, Kluwer Academic Publishers, 1994.

[5] D. G. Lowe, Object Recognition from local scale–invariant features, ICCV (International Conference on Computer Vision), 1999.

[6] J. Matas, J. Burianek, and J. Kittler. Object recognition using the invariant pixel-set signature, BMVC (British Machine Vision Conference), 2000.

[7] F. Schaffalitzky and A. Zisserman. Viewpoint invariant texture matching and wide baseline stereo, ICCV, 2001.

[8] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector, ECCV, 2002.

[9] K. Mikolajczyk and C. Schmid, A performance evaluation of local descriptors, CVPR, 2003.

[10] Csurka, Gabriela Dance, Christopher Fan, Lixin Willamowski, Jutta Bray, Cédric. (2004). Visual categorization with bags of keypoints. Work Stat Learn Comput Vision, ECCV. Vol. 1.

Links

Overleaf Project Link
Github Repository Link