Lecture #13: Visual Bag of Words

Jared Hector, Taylor Song, Jeffrey Propp, Ryota Eki, Jared Smith, Ivan Villa-Renteria, Eunice Yang.
Department of Computer Science
Stanford University
Stanford, CA 94305
{jhector, jhtsong, jpropp, reki, smithje, ivillar, eunicey}@stanford.edu

1 Introduction

The visual bag of words algorithm is used to represent images as collections of unordered patches, in order to eventually perform object classification. This method disregards the location of features, simply amassing them in order to match the bag of visual patches to an object.

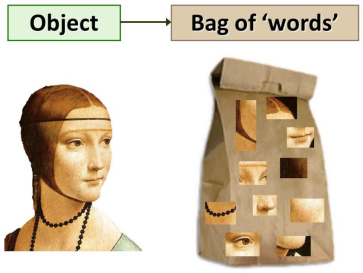


Figure 1: Visual bag of words. Source: lecture 13, slide 3

2 Origins of feature representation

2.1 Origin 1: Texture recognition

The bag of words algorithm was derives originally from algorithms used to detect textures present in images. These images are characterized by basic elements, or **textons**, that are repeated.

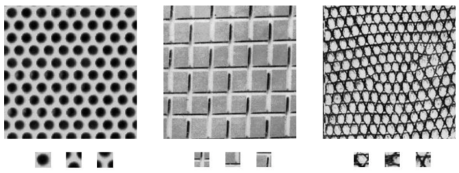


Figure 2: Texture Recognition. Source: lecture 13, slide 5, Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

These textures can be thought of as composed by many repetitions of small, basic elements.

One way to recognize images by their basic elements is to build a universal texton dictionary, or a collection of basic elements that can be used to construct new textures. For each image, we then produce a histogram that represents how much a specific texton appears in the image. Different histogram shapes signify different textures, with textons that occur more frequently having more weight than those that are infrequent.

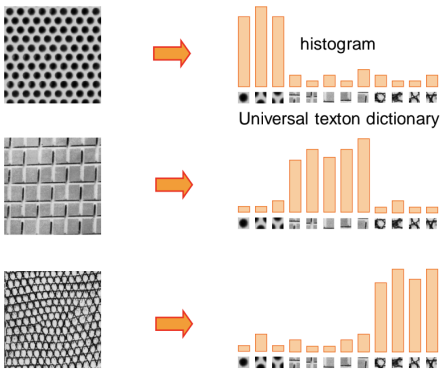


Figure 3: Texton Histograms. Source: lecture 13, slide 7

These texton histograms can be used as features in classification frameworks.

2.2 Origin 2: Bag-of-words models for text analysis

The other origin of the visual bag of words framework lies in text analysis. The goal of this method, used in natural language processing, is to run topic detection on a document. The bag of words model of text analysis disregards sentence structure and word order, attempting to characterize a document by word frequency. For example, a visualization of the bag of words of the State of the Union in 1948 gives an idea of the topics that were discussed.

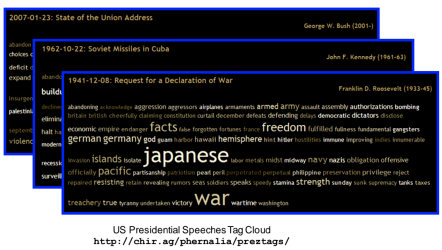


Figure 4: State of the Union Bag of Words. Source: lecture 13, slide 8, US Presidential Speeches Tag Cloud http://chir.ag/phernalia/preztags/

This method, used by natural language researchers to classify documents, is the process from which "Bag of Words" derives its name.

3 Bags of features

The Bags of Features method is similar to that of textual analysis. We will divide an image up into patches and count their frequencies in order to give us an idea of the content of the image without knowing its full shape.

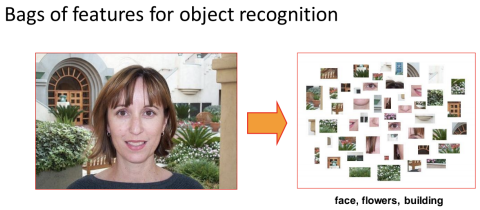


Figure 5: Bag of Features. Source: lecture 13, slide 9, Csurka et al. (2004), Willamowski et al. (2005), Grauman & Darrell (2005), Sivic et al. (2003, 2005)

3.1 Bags of features for object recognition

If we run our bag of features on a set of images with 6 types of objects, we can see that for certain objects our bag of features approach performs pretty well. In this experiment, bag of features is compared to the parts-and-shape model, which actually takes into account the arrangement of parts within the object. Though bag of features does not take into account such geometry, it does outperform the other model in every object type.

class	bag of features	bag of features	Parts-and-shape model
	Zhang et al. (2005)	Willamowski et al. (2004)	Fergus et al. (2003)
airplanes	**98.8**	97.1	90.2
cars (rear)	98.3	**98.6**	90.3
cars (side)	**95.0**	87.3	88.5
faces	**100**	99.3	96.4
motorbikes	**98.5**	98.0	92.5
spotted cats	**97.0**	—	90.0

Figure 6: Testing How Well Bag of Features Works. Source: Lecture 13, slide 10

3.2 Basic method

The idea of the bag of features method is first to take a set of images, then extract features from the image set and build up a dictionary with some common features across the images.

If we bring in a new image, we can first start by again extracting features from it. Then, we can "match" those features to the contents we have in the dictionary, and get the closest item match for each feature.

3.2.1 Step 1: Feature extraction

The first step in this process is to extract features from our set of images, then construct a visual library containing the types of objects we might see in our images. These features are then quantified, which allows us to represent images by the frequency with which different features appear in the image, as seen in the histograms below.

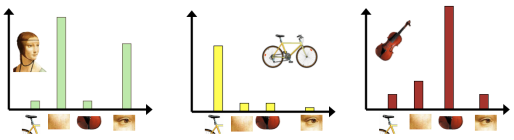


Figure 7: Representing Images are Frequencies of Dictionary "Visual Words". Source: Lecture 13, Slide 16

There are a number of ways we can extract features. One option is to divide the image into a regular grid and densely sample the image. Another option is using an interest point detector (i.e. Harris corner detector), allowing us to only extract patches with interesting structures. Other options also exist, including using random sampling or segmentation-based patches.

Whatever method is used, we will end up with patches from an image, which can then describe contents of every patch with a SIFT descriptor.

3.2.2 Step 2: Learn visual vocabulary

When we run our keypoint detector, we will extract patches corresponding to semantically equivalent points in our objects. In the following image example, the detector fires on the wheel in all the images with planes. We then can use preprocessing to extract a standardized version of each patch for use in our library.

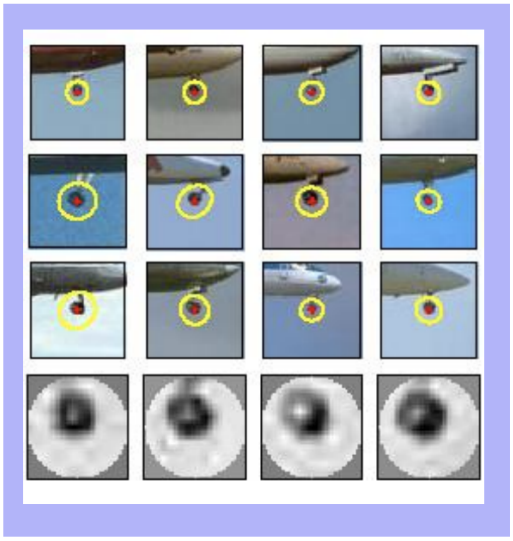


Figure 8: Example: Detector Outlines Wheel (Common Feature) in Plane Images. Source: lecture 13, slide 21, Sivic et al. (2015)

With our collection of patches derived from our many input images, we can describe these patches with feature **vectors**. We should be able to extract a big collection of descriptors from our "data set".

These vectors live in a high-dimensional feature space (3D in the case of the below image). In the figure below, we can see that the vectors are each represented by a dot in the graph. Similar feature vectors that represent similar patches should be in a close region in the feature space. This is represented in the image below: motorbikes might be one cluster of black dots, while airplane wheels would be in another.

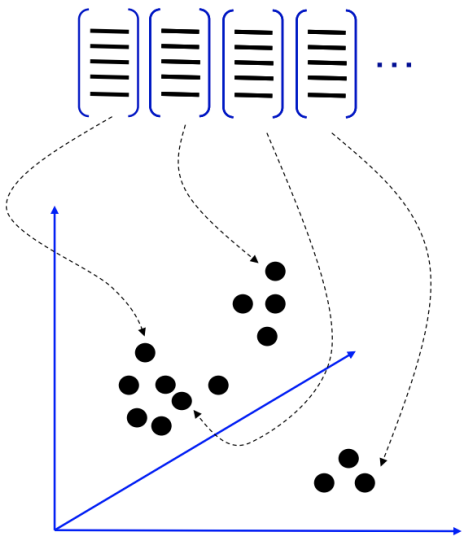


Figure 9: Features Vectors Represented in Multi-Dimensional Space. Source: lecture 13, slide 22, Sivic et al. (2015)

Groups of feature descriptors will reveal patches in the feature space that denote common elements in the images. These groups of similar looking patches can be clustered (using K-means, for example), allowing us to identify their location. We then use the center of each cluster to represent that cluster.

(a) Patches in the feature space after clustering.

(b) Representation of cluster centers to form visual vocabulary.

Figure 10: Identification of commonalities between patches using clustering allows us to define the visual vocabulary. Source: Lecture 13.1, slides 23-24, credit Josef Sivic.

Each center corresponds to a separate "visual word" that contributes to the visual vocabulary. In all, this method has given us a way to interpret separate feature vectors and identify commonalities to construct the visual vocabulary.

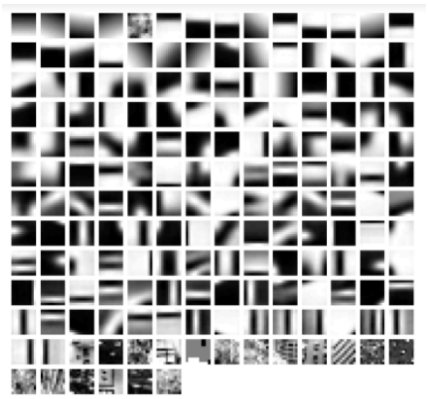


Figure 11: An example visual vocabulary. Source: lecture 13, slide 25, Fei-Fei et. al 2005.

A potential issue to consider is choice of vocabulary size. If the vocabulary size is too small, the visual words are not representative of all patches. Conversely, if the vocabulary size is too large, there may be quantization artifacts and overfitting.

3.2.3 Step 3: Quantize features using visual vocabulary

Clustering allows us to quantize features by learning the visual vocabulary or codebook. The codebook can be considered "universal" if learned on a sufficiently representative training set. The resulting universal codebook can be used to quantize features in general.

To do so, we employ a *vector quantizer* to map features to the nearest "codevector" (i.e. cluster center) produced by k-means clustering. It is important to remember that "codebook" corresponds to the visual vocabulary and "codevector" corresponds to a visual word in the visual vocabulary. This process is useful because it allows us to represent a high-dimensional feature descriptor as an index in the codebook.

3.2.4 Step 4: Represent images by frequencies of "visual words"

The final step of this process utilizes the created codebook to represent images using histograms of the visual dictionary.

The process is as follows: given an input image, for each patch, compute the descriptor of that patch and match it to the closest entry in the visual dictionary. After doing so, increment that entry's frequency value in the constructed histogram.

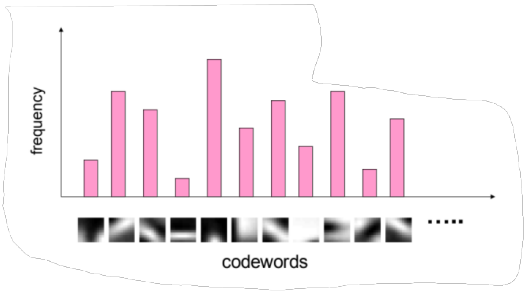


Figure 12: An example frequency histogram. The x-axis represents a particular visual word in the visual vocabulary while the y-axis represents the frequency value of a given visual word. Source: lecture 13.1, slide 30.

4 Bags of features and its Application

4.1 Image Classification

Bag of features can be used in image classification framework. We can learn the model for classification by: 1. Treat the Bag of Word representation as the input feature vector for a standard classifier. An example would be to use the representation within the k-nearest neighbors framework. 2. Cluster the Bag of Word vectors over a collection of images. An advantage is that class label in image collection is not needed.

4.2 Image Search

Image search or image matching process is the idea to use the bag of words representation to match a query image within a large database. This can be done by: 1. Build the database by extracting features from database images, 2. Learn a visual vocabulary using k-means and compute histograms, 3. Compute the importance, or the weights for each word 4. Create an inverted file mapping words that could be converted to images.

The reason why we need to weight the words is that just as in text, some words may contain more importance or information in a given sentence compared to other words, and we want to reflect this in our model. It is often assumed that a word that appears in all documents is less useful for matching.

By building the right model, we will be able to perform large-scale image search and find images that match the query image in a given database. This model is even able to achieve real time performance but one thing to note is that as the database grows, the performance with this approach will degrade.

4.3 Action Recognition

Bag of Words can also be applied to video for action recognition. We can extract video keypoint and compute a descriptor to build visual vocabularies. We can then use inversion of keypoints that take interest in video patches.

Following image shows how we can take input videos, perform feature extraction, compute a visual vocabulary of video patches, and compute corresponding histogram for every input video. This can be used for representation for classification for a new input.

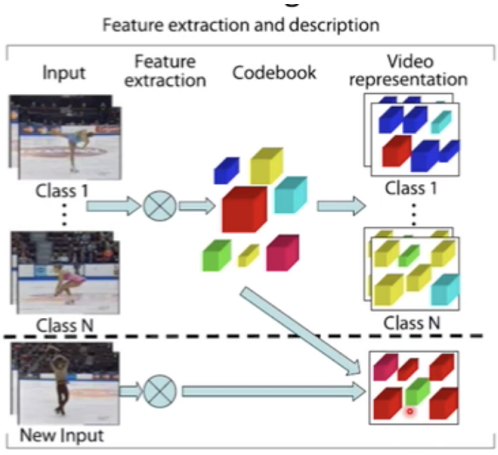


Figure 13: Bags of features for action recognition. Source: lecture 13.2, slide 15.

5 Spatial Pyramid Matching

One of the biggest weaknesses of visual bag of words is that it ignores spatial information about features, meaning that there is extra information in each image that isn't being leveraged for tasks such as object classification or scene categorization. Recall that image patches in a visual bag of words only retain information about their own content, not about their specific location image or relations to other objects in the image. This means that given a histogram of a visual bag of words, we don't have any way of rearranging patches to reconstruct the original image, since we are missing spatial information.

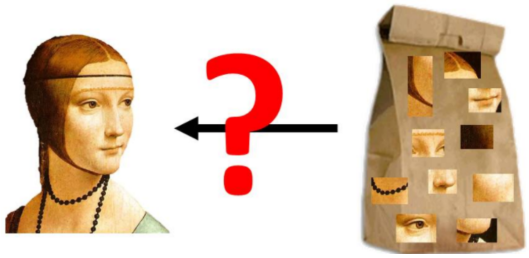


Figure 14: Bag of words can't be used to reconstruct an original image since spatial information is lost.

Luckily, one of the ways in which this can be remedied is through the use of pyramids to recover such valuable spatial information.

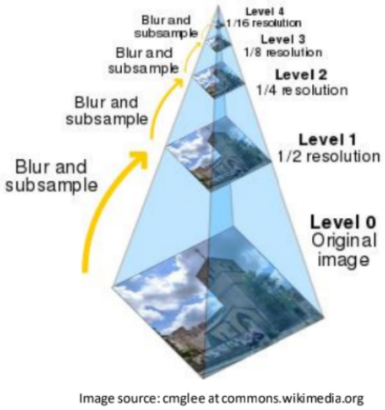


Figure 15: A spatial pyramid.

These are built by using copies of an image at multiple resolutions, where each level in the pyramid is 1/4 the size of the previous level. Thus, the lowest level has the highest resolution, and the highest level has the lowest resolution. It tries to summarize the content of the image at various resolutions.

At the lowest level, we compute a single histogram for the entire content of the image. At next level, we can divide image into 4 parts and for each part compute histograms of visual features within each part of image. Within each image, we're not preserving order or arrangement of features, but since we're accumulating features within 4 different histograms, we're still getting a loose sense of geometric arrangement. Top histograms would have sky-looking patches, bottom histograms would have ground-looking codewords.

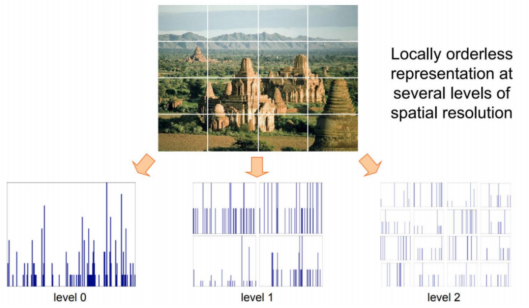
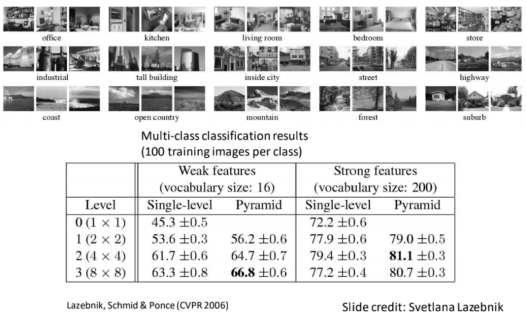


Figure 16: Example of spatial pyramid matching applied at different levels.

At the next level, we further divide into 16 blocks, and recompute histograms.

At the end, we can represent the image as a concatenation of all the histograms we have computed in order to capture some semblance of the spatial arrangement of features while retaining flexibility of histograms.

This model is useful for scene categorization.



Multi-class classification results
(100 training images per class)

	Weak features (vocabulary size: 16)		Strong features (vocabulary size: 200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0 (1×1)	45.3 ±0.5		72.2 ±0.6	
1 (2×2)	53.6 ±0.3	56.2 ±0.6	77.9 ±0.6	79.0 ±0.5
2 (4×4)	61.7 ±0.6	64.7 ±0.7	79.4 ±0.3	**81.1** ±0.3
3 (8×8)	63.3 ±0.8	**66.8** ±0.6	77.2 ±0.4	80.7 ±0.3

Lazebnik, Schmid & Ponce (CVPR 2006)

Slide credit: Svetlana Lazebnik

Figure 17: Results of spatial pyramid matching applied to scene categorization

This is also useful for object classification.

Multi-class classification results (30 training images per class)

	Weak features (16)		Strong features (200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0	15.5 ±0.9		41.2 ±1.2	
1	31.4 ±1.2	32.8 ±1.3	55.9 ±0.9	57.0 ±0.8
2	47.2 ±1.1	49.3 ±1.4	63.6 ±0.9	**64.6** ±0.8
3	52.2 ±0.8	**54.0** ±1.1	60.3 ±0.9	64.6 ±0.7

Figure 18: Results of spatial pyramid matching applied to object classification

Notice that the pyramid method always achieves higher performance than the single-level approach. This makes sense, since the pyramid methods capture more information about the images that vanilla bag of words doesn't (spatial information). Also note that generally, the more levels that are added to the method, the higher performance you get. Moreover, we also see that strong features in the pyramid outperform weaker features.

6 Naive Bayes Algorithm

6.1 Description

The Naive Bayes Algorithm is a method which is used to classify images based on a histogram of occurrences on "visual words." Let us represent the entire histogram of words as **x**.

To create our prediction, we attempt to maximize the following posterior probability:

$$c^* = arg\max_c P(c|\mathbf{x})$$

To realistically compute this value, we must make the assumption of conditional independence. That is, we assume that the appearance of words are conditionally independent given the class c. Then, all we need to do to calculate the joint probability is multiply the probability of each word appearing. For image x,

$$P(x|c) = \Pi_{i=1}^{m} P(x_i|c)$$

where x_i is the event of visual word v_i being present in the image and m is the size of our vocabulary (number of possible words).

We will then calculate these probabilities by calculating the frequency that a given word appears given each class. For a given word and class, we simply sum up the number of times the word appears in all documents classified as that class, and normalize the count with the value of the word count. We do this for each word+class combination to obtain all possible values of $P(v_i|c)$. Lastly, we compute the class priors, that is for each class we calculate $P(c)$, which is simply the number of documents with that class normalized by the total number of documents. With all of this information, we can now calculate:

$$\begin{aligned} P(c|x) &= \frac{P(c)P(x|c)}{\sum_{c'} P(c')P(x|c')} \\ &= \frac{P(c)\prod_{i=1}^{m} P(x_i|c)}{\sum_{c'} P(c')\prod_{i=1}^{m} P(x_i|c')} \end{aligned}$$

which tells us the probability a given image belongs to a class.

6.2 Posterior

Using the equations above, we can compute the probability that an image represented by x belongs to class category c.

$$\begin{aligned} P(c|x) &= \frac{P(c)P(x|c)}{\sum_{c'} P(c')P(x|c')} \\ \iff P(c|x) &= \frac{P(c)\prod_{i=1}^{m} P(x_i|c)}{\sum_{c'} P(c')\prod_{i=1}^{m} P(x_i|c')} \end{aligned}$$

Thus, we see that the probability that x belongs to class c_1 is

$$P(c_1|x) = \frac{P(c_1)\prod_{i=1}^{m} P(x_i|c_1)}{\sum_{c'} P(c')\prod_{i=1}^{m} P(x_i|c')}$$

and the probability that x belongs to c_2 is

$$P(c_2|x) = \frac{P(c_2)\prod_{i=1}^{m} P(x_i|c_2)}{\sum_{c'} P(c')\prod_{i=1}^{m} P(x_i|c')}$$

Note that the denominators are the same, we can now say that for the general class c

$$P(c|x) \propto P(c)\prod_{i=1}^{m} P(x_i|c)$$

and we can take the log

$$\log P(c|x) \propto \log P(c) + \sum_{i=1}^{m} P(x_i|c)$$

6.3 Classification

We can estimate the class that the image represented by x belongs to by:

$$\begin{aligned} c^* &= argmax P(c|x) \\ \iff c^* &= argmax \log P(c|x) \end{aligned}$$

Thus, from the equations in the previous sections,

$$c^* = argmax \log P(c) + \sum_{i=1}^{m} \log P(x_i|c)$$

7 Summary

In this lecture we have learned about the visual bag of words algorithm to classify images.