# Project 3 Report: Feature Encoding

**Mingquan Feng**
517030910373

**Mingjie Li**
517030910344

**Shangning Xu**
517030910384

## Abstract

Feature encoding is the technique of constructing a global feature vector from a image's set of local descriptors, extracted by digital image processing methods like SIFT. In this project, we experiment with different approaches to local descriptor extraction, like SIFT and Selective search, and different approaches to feature encoding, like bag of words, VLAD, Fisher vector, super vector and LLC. Encoded features from the Animals with Attributes 2 dataset are used to train SVM and deep neural networks to compare the strength of different local descriptor extraction and feature encoding methods. Our experiment shows that, while selective search with DNN gives the best accuracy, the accuracy is inferior to that of pre-extracted deep learning features with linear SVM.

## 1 Introduction

In this project, we experiment with two feature extraction methods, SIFT [6] and selective search [9], and five feature encoding methods, bag of words [4], VLAD [5], Fisher vector [8], super vector [11] and locality-constrained linear coding (LLC) [10]. For classifiers, SVM with rbf kernel and ResNet for deep neural networks (DNN) are used.

This paper is organized as follows: Section 2 and 3 introduce the theory that underlies various methods of local descriptor extraction and feature encoding, respectively. Section 4 present our experiments with these methods and analysis of results. Section 5 concludes the report.

## 2 Local Descriptor Extraction

### 2.1 Scale-Invariant Feature Transform (SIFT)

The scale-invariant feature transform (SIFT)[6] is a feature detection algorithm in computer vision to detect and describe local features in images. The pipeline of SIFT detection can be summarized in the following manner.

1. *Scale-space extrema detection:* Inspired by the behavior of complex cells in the cerebral cortex of mammalian vision, SIFT first constructs the image pyramid and the difference-of-gradient (DoG) pyramid. For an original image $I(x, y)$, we can first build a pyramid by upsampling or downsampling to generate an image pyramid. Then we expand each level by using Gaussian filters $G(x, y, \sigma)$ with different bandwidths to generate images with different blurring levels $L(x, y, \sigma)$. (For simplicity, I omit the index to indicate the scale (or size) of the image.)

$$L(x, y, \sigma) = I(x, y) * G(x, y, \sigma), \text{ where } G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (1)$$

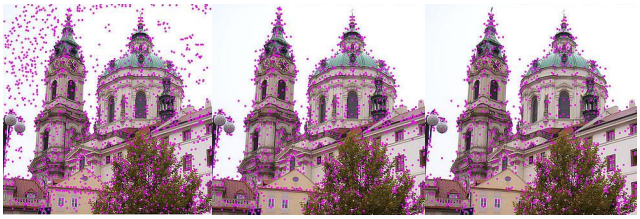

Figure 1: After scale space extrema are detected (their location being shown in the leftmost image) the SIFT algorithm discards low-contrast keypoints (remaining points are shown in the middle image) and then filters out those located on edges. Resulting set of keypoints is shown on last image.

Keypoints are then taken as maxima/minima of the Difference of Gaussians (DoG) that occur at multiple scales. Specifically, a DoG image $D(x, y, \sigma)$ is given by

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{2}$$

Once DoG images have been obtained, keypoints are identified as local minima/maxima of the DoG images across scales. This is done by comparing each pixel in the DoG images to its eight neighbors at the same scale and nine corresponding neighboring pixels in each of the neighboring scales. If the pixel value is the maximum or minimum among all compared pixels, it is selected as a candidate keypoint.

2. *Keypoint localization:* Scale-space extrema detection produces too many keypoint candidates, some of which are unstable. The next step in the algorithm is to perform a detailed fit to the nearby data for accurate location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge. The detailed algorithm can bee seen in Wikepedia-SIFT. Figure 1 shows a visualization of the discarding of unstable candidate points.[1]
3. *Orientation assignment:* In this step, each keypoint is assigned one or more orientations based on local image gradient directions. This is the key step in achieving invariance to rotation as the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.

$$\begin{aligned} m(x, y) &= \sqrt{[L(x+1, y) - L(x-1, y)]^2 + [L(x, y+1) - L(x, y-1)]^2} \\ \theta(x, y) &= \text{atan2}(L(x, y+1) - L(x, y-1), L(x+1, y) - L(x-1, y)) \end{aligned} \tag{3}$$

The peaks in this histogram correspond to dominant orientations. Once the histogram is filled, the orientations corresponding to the highest peak and local peaks that are within 80% of the highest peaks are assigned to the keypoint.

4. *Keypoint descriptor:* Based on the orientation calculated in the previous step, we can calculate a rotation invariant descriptor for each key point. This is done by using the set of orientation histograms created on $4 \times 4$ pixel neighborhoods with 8 bins each.

In conclusion, this approach transforms an image into a large collection of local feature vectors, each of which is invariant to image translation, scaling, and rotation, and partially invariant to illumination changes and affine or 3D projection.

### 2.2 Selective Search

An image usually contains much information. The objects in an image usually have different shape, scale, color and texture. To detect an object from an image is very difficult. Selective Search[9] is a method to address this problem based on merging of similar regions. This algorithm can be summarized as the following pipeline.

1. Generate a set of regions $R = \{r_1, r_2, \ldots, r_n\}$ based on [2].

[1] From https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

2. Calculate the similarities of neighboring region pairs.
3. Iteratively merge the most similar neighboring regions until there is no neighboring regions.

In selective search, there are 4 metrics jointly for calculating the similarity. When calculating the similarities, we usually normalize them into $[0, 1]$, where 1 denotes that the similarity is maximal.

$s_{color}(r_i, r_j)$ measures color similarity. Specifically, for each region we obtain one-dimensional color histograms for each color channel using 25 bins. This leads to a color histogram $C_i = \{c_i^1, c_i^2, \ldots, c_i^n\}$ for each region $r_i$ with dimensionality $n = 75$ when three color channels are used. The colour histograms are normalized using the L1 norm. Similarity is measured using the histogram intersection:

$$s_{color}(r_i, r_j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k) \tag{4}$$

$s_{texture}(r_i, r_j)$ measures texture similarity. We take Gaussian derivatives in eight orientations using $\sigma = 1$ for each color channel. For each orientation for each color channel we extract a histogram using a bin size of 10. This leads to a texture histogram $T_i = \{t_i^1, t_i^2, \ldots, t_i^n\}$ for each region $r_i$ with dimensionality $n = 240$ when three color channels are used. Texture histograms are normalised using the L1 norm. Similarity is measured using histogram intersection:

$$s_{texture}(r_i, r_j) = \sum_{k=1}^{n} \min(t_i^k, t_j^k) \tag{5}$$

$s_{size}(r_i, r_j)$ encourages small regions to merge early. This forces regions in $S$, *i.e.* regions which have not yet been merged, to be of similar sizes throughout the algorithm. This is desirable because it ensures that object locations at all scales are created at all parts of the image. For example, it prevents a single region from gobbling up all other regions one by one, yielding all scales only at the location of this growing region and nowhere else. $s_{size}(r_i, r_j)$ is defined as the fraction of the image that ri and rj jointly occupy:

$$s_{size}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(r_j)}{\text{size}(im)} \tag{6}$$

$s_{fill}(r_i, r_j)$ measures how well region $r_i$ and $r_j$ fit into each other. The idea is to fill gaps: if $r_i$ is contained in $r_j$ it is logical to merge these first in order to avoid any holes. On the other hand, if $r_i$ and $r_j$ are hardly touching each other they will likely form a strange region and should not be merged. To keep the measure fast, we use only the size of the regions and of the containing boxes. Specifically, we define $BB_{ij}$ to be the tight bounding box around $r_i$ and $r_j$. Now $s_{fill}(r_i, r_j)$ is the fraction of the image contained in $BB_{ij}$ which is not covered by the regions $r_i$ and $r_j$:

$$s_{fill}(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(r_i) - \text{size}(r_j)}{\text{size}(im)} \tag{7}$$

In [9], the authors used a combination of the above four as the final similarity measure.

$$s(r_i, r_j) = a_1 s_{color}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + a_3 s_{size}(r_i, r_j) + a_4 s_{fill}(r_i, r_j) \tag{8}$$

# 3 Feature encoding

## 3.1 Bag of words

The bag-of-words model [4] in computer vision is inspired by the synonymous model in text mining, where each document is seen as a multiset ("bag") of words, ignoring order information, and the bag of words is a sparse vector of occurrence counts of words. For feature encoding, the following procedure is used to compute bags of words:

1. Learning a codebook: all images' local features are pooled and applied K-Means clustering. The result of clustering with $K$ clusters is a codebook of size $K$.
2. Encoding: For each image, its local features are assigned a cluster. The encoded feature vector $f$ is a vector of occurrence counts of cluster assignment. That is,

$$f[i] = \text{number of local descriptors that belong to cluster } i \tag{9}$$

Jurie and Triggs [4] suggest that encoding features with K-Means clustering has the adverse effect that clusters centers are distributed around dense neighborhoods, which mainly consist of *generic* features prevalent in most images, like edges. However, neighborhoods that are most informative for classification tend to have intermediate frequencies. As a result, highly-discriminant neighborhoods occur rarely, and when they occur, there are two few of them to "swing" the prediction.

### 3.2 Vector of locally aggregated descriptors

Vector of locally aggregated descriptors (VLAD), first prosposed by Jégou et al. [5], preserves the first-order information in descriptors. As for bag of words, we first learn a codebook $C = c_1, \ldots, c_k$ of $k$ visual words with k-means. Each local descriptor $x$ is associated to its nearest visual word $c_i = \mathrm{NN}(x)$. The idea of the VLAD descriptor is to accumulate, for each visual word $c_i$, the differences $x - c_i$ of the vectors $x$ assigned to $c_i$. This characterizes the distribution of the vectors with respect to the center.

Assuming the local descriptor to be $d$-dimensional, the dimension $D$ of our representation is $D = kd$. In the following, we represent the descriptor by $v_{i,j}$, where the indices $i = 1, \ldots, k$ and $j = 1, \ldots, d$ respectively index the visual word and the local descriptor component. Hence, a component of $v$ is obtained as a sum over all the image descriptors:

$$v_{i,j} = \sum_{x \text{ such that } \mathrm{NN}(x)=c_i} x_j - c_{i,j} \tag{10}$$

where $x_j$ and $c_{i,j}$ respectively denote the $j$th component of the descriptor $x$ considered and of its corresponding visual word $c_i$. The vector $v$ subsequently $L_2$-normalized by $v := v/||v||_2$.

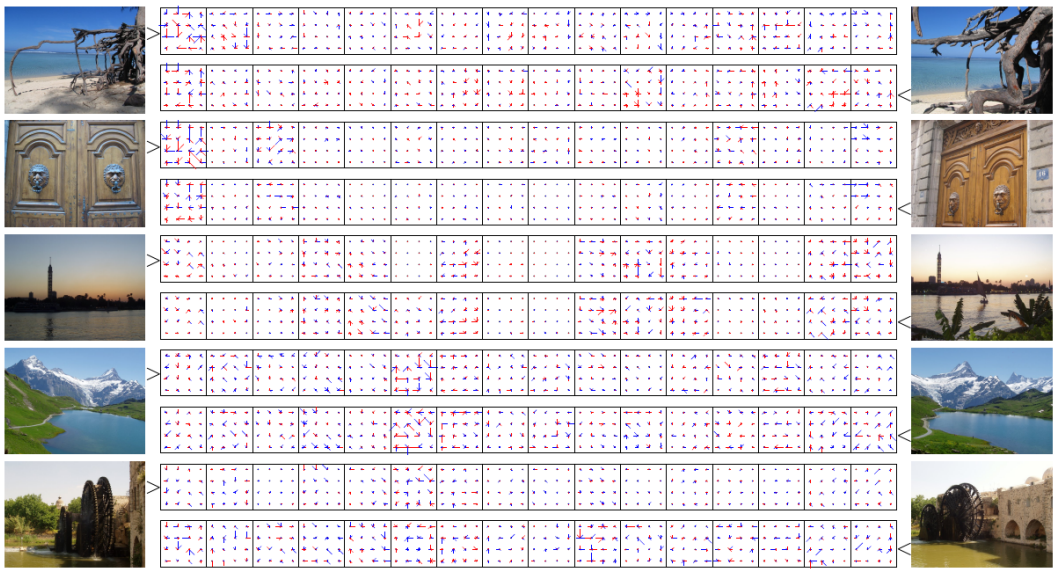

Figure 2: Images and corresponding VLAD descriptors, for $k = 16$ centroids ($D = 16 \times 128$). The components of the descriptor are represented like SIFT, with negative components in red.

Figure 2 shows that the descriptors are relatively sparse (few values have a significant energy) and very structured: most high descriptor values are located in the same cluster, and the geometrical structure of SIFT descriptors is observable. For sufficiently similar images, the closeness of the descriptors is obvious.

### 3.3 Fisher vector

Perronnin and Dance [7] first proposed a formulation that is similar to Fisher vector (FV). The problem at hand is image categorization, that is, assigning one or multiple labels to an image based on its semantics content, instead of feature encoding. In their formulation, the idea is to characterize the training images $X = \{x_t, 1, \ldots, T\}$ with the gradient vector $\nabla_\lambda \log p(X|\lambda)$, where $\lambda$ is the parameter for PDF $p$. $p$ is approximated with a GMM model with $\lambda = \{w_i, \mu_i, \Sigma_i, i = 1, \ldots, N\}$. The name "Fisher vector" and its application to feature encoding appears in [8], as introduced below.

#### 3.3.1 Fisher kernel

Let $X = \{x_t, t = 1 \ldots T\}$ be a sample of $T$ observations $x_t \in \mathscr{X}$. Let $u_\lambda$ be a probability density function which models the generative process of elements in $\mathscr{X}$ where $\lambda$ denotes the vector of $M$ parameters of $u_\lambda$. In statistics, the score function is given by the gradient of the log-likelihood of the data on the model:

$$G_\lambda^X = \nabla_\lambda \log u_\lambda(X) \tag{11}$$

This gradient describes the contribution of the individual parameters to the generative process. In other words, it describes how the parameters of the generative model $u_\lambda$ should be modified to better fit the data $X$.

From the theory of information geometry, a parametric family of distributions $\mathscr{U} = \{u_\lambda, \lambda \in \mathbb{R}^M\}$ can be regarded as a Riemanninan manifold $M_\Lambda$ with a local metric given by the Fisher information matrix $F_\lambda \in \mathbb{R}^{M \times M}$:

$$F_\lambda = E_{x \sim u_\lambda}\left[G_\lambda^X G_\lambda^{XT}\right] \tag{12}$$

The similarity between two samples $X$ and $Y$ using the Fisher kernel (FK) can be defined as:

$$K_{FK}(X, Y) = G_\lambda^{XT} F_\lambda^{-1} G_\lambda^Y \tag{13}$$

Since $F_\lambda$ is positive semi-definite, so is its inverse. Using the Cholesky decomposition $F_\lambda^{-1} = L_\lambda^T L_\lambda$, the FK can be re-written explicitly as a dot-product:

$$K_{FK}(X, Y) = \mathscr{G}_\lambda^{XT} \mathscr{G}_\lambda^Y \tag{14}$$

where

$$\mathscr{G}_\lambda^X = L_\lambda G_\lambda^X = L_\lambda \nabla_\lambda \log u_\lambda(X) \tag{15}$$

We call this normalized gradient vector the FV of $X$. The dimensionality of the FV $\mathscr{G}_\lambda^X$ is equal to that of the gradient vector $G_\lambda^X$.

#### 3.3.2 Fisher vector in feature encoding

To apply FK to feature encoding, let $X = \{x_t, t = 1 \ldots T\}$ be the set of $D$-dimensional local descriptors extracted from an image, e.g. a set of SIFT descriptors. Assuming that the samples are independent, the FV can be rewritten as follows:

$$\mathscr{G}_\lambda^X = \sum_{t=1}^{T} L_\lambda \nabla_\lambda \log u_\lambda(x_t) \tag{16}$$

The GMM is again chosen to approximate the probability distribution $u_\lambda$, with $\lambda = \{w_k, \mu_k, \Sigma_k, k = 1, \ldots, K\}$, then

$$u_\lambda(x) = \sum_{k=1}^{K} w_k u_k(X) \tag{17}$$

where $u_k$ denotes the Gaussian $k$. We assume diagonal covariance matrices which is a standard assumption and denote by $\sigma_k^2$ the variance vector, i.e. the diagonal of $\Sigma_k$. Given the GMM model,

the normalized gradients are

$$\mathscr{G}_{\alpha_k}^X = \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} (\gamma_t(k) - w_k) \tag{18}$$

$$\mathscr{G}_{\mu_k}^X = \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} \gamma_t(k) \frac{x_t - \mu_k}{\sigma_k} \tag{19}$$

$$\mathscr{G}_{\sigma_k}^X = \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} \gamma_t(k) \frac{1}{\sqrt{2}} \left[ \frac{(x_t - \mu_k)^2}{\sigma_k^2} - 1 \right] \tag{20}$$

Note that $\mathscr{G}_{\alpha_k}^X$ is a scalar while $\mathscr{G}_{\mu_k}^X$ and $\mathscr{G}_{\sigma_k}^X$ are $D$-dimensional vectors. The final FV is the concatenation of the gradients $\mathscr{G}_{\alpha_k}^X$, $\mathscr{G}_{\mu_k}^X$, $\mathscr{G}_{\sigma_k}^X$ for $k = 1, \ldots, K$ and is therefore of dimension $E = (2D + 1)K$. The FV is further normalized by the sample size $T$.

### 3.4 Super vector encoding

Super vector encoding [11] is similar to the Fisher encoding. The clustering algorithm can be hard, i.e. K-means, or it can be soft, i.e. GMM. Here we adopt the latter.

$$\begin{aligned}
q_{ki} &= \frac{p\left(\mathbf{x}_i | \mu_k, \Sigma_k\right) \pi_k}{\sum_{j=1}^{K} p\left(\mathbf{x}_i | \mu_j, \Sigma_j\right) \pi_j}, \quad k = 1, \ldots, K \\
p_k &= \frac{1}{N} \sum_{i=1}^{N} q_{ik} \\
\mathbf{u}_k &= \frac{1}{\sqrt{p_k}} \sum_{i=1}^{N} q_{ik} \left(\mathbf{x}_t - \mu_k\right) \\
s_k &= s\sqrt{p_k} \\
\mathbf{f}_{\text{super}} &= \left[s_1, \mathbf{u}_1^\top, \ldots, s_K, \mathbf{u}_K^\top\right]^\top
\end{aligned}$$

where $s$ is a constant chosen to balance $s_k$ with $\mathbf{u}_k$ numerically. Here $s_k$ is a size-1 scaler, denoting the mass of each cluster (0-order information). $\mathbf{u}_k$ is a size-D vector, denoting difference between local features and cluster centers (1-order information). Therefore, $\mathbf{f}_{\text{super}}$ is a size-K(D+1) vector. Compared to the Fisher encoding, the super vector encoding differs at:

1. Super vector encoding considers 0-order and 1-order information, therefore its vector size is KD+K, while Fisher encoding considers 1-order and 2-order information and vector size is 2KD.
2. super vector encoding normalizes each cluster by the square root of the posterior probability $\sqrt{p_k}$ rather than of the prior probability $\sqrt{\pi_k}$.

### 3.5 Locality-constrained Linear Coding (LLC)

Locality-constrained Linear Coding (LLC) [10] explicitly encourages the coding to be local, and theoretically pointed out that under certain assumptions locality is more essential than sparsity, for successful nonlinear function learning using the obtained codes. Specifically, let $\mathbf{X}$ be a set of D-dimensional local descriptors $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$, and $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_M] \in \mathbb{R}^{D \times M}$ is a codebook with $M$ entries, the output coding is $\mathbf{C} \in \mathbb{R}^{M \times N}$.

$$\begin{aligned}
\arg\min_{\mathbf{C},\mathbf{B}} \quad & \sum_{i=1}^{N} \left\|\mathbf{x}_i - \mathbf{B}\mathbf{c}_i\right\|^2 + \lambda \left\|\mathbf{d}_i \odot \mathbf{c}_i\right\|^2 \\
\text{st.} \quad & \mathbf{1}^\top \mathbf{c}_i = 1, \forall i \\
& \left\|\mathbf{b}_j\right\|^2 \leq 1, \forall j
\end{aligned}$$

where $\mathbf{d}_i \in \mathbb{R}^M$ is the locality adaptor that gives different freedom for each basis vector proportional to its similarity to the input descriptor $\mathbf{x}_i$.

$$\begin{aligned}\mathbf{d}_i &= \exp\left(\frac{\text{dist}\left(\mathbf{x}_i, \mathbf{B}\right)}{\sigma}\right) \\ &= \exp\left(\frac{\left[\text{dist}\left(\mathbf{x}_i, \mathbf{b}_1\right), \ldots, \text{dist}\left(\mathbf{x}_i, \mathbf{b}_M\right)\right]^T}{\sigma}\right)\end{aligned}$$

Such optimization problem can be solved by solved using Coordinate Descent method to iteratively optimizing $\mathbf{C}(\mathbf{B})$ based on existing $\mathbf{B}(\mathbf{C})$.

Notice that $\|\mathbf{d}_i \odot \mathbf{c}_i\|^2$ is locality regularization term, and it has several attractive properties:

1. Better reconstruction: in LLC, each descriptor is more accurately represented by multiple bases, and LLC code captures the correlations be- tween similar descriptors by sharing bases.
2. Local smooth sparsity: the explicit locality adaptor in LLC ensures that similar patches will have similar codes, therefore it avoids over-completeness of the codebook.
3. Analytical solution: solution of LLC can be derived analytically, and has fast approximation, therefore it avoids computationally demanding optimization procedures.

# 4 Experiments

## 4.1 Dataset

Download Animals with Attributes (AwA2) dataset [2]. This dataset consists of 37322 images of 50 animal classes. We randomly split the images in each category into 60% for training and 40% for testing, using stratified sampling. We also use 5-fold cross-validation within the training set to select optimal parameters.

## 4.2 SIFT

In our experiments, we implemented SIFT feature extraction using the `opencv`[3] library in Python. *I.e.* the function `cv2.xfeatures2d.SIFT_create()` is used to create a SIFT detector. For the three feature encoding methods, we implemented them by ourselves, except for the clustering part.

The input of a SIFT detector is a gray-scale image, so we need to transform the RGB image into gray-scale via the `cv2.cvtColor` function. To achieve this, the second parameter of this function should be set as `cv2.COLOR_BGR2GRAY`. The `detectAndCompute` method of the SIFT detector will return (1) $N_{SIFT}$ key points and (2) their local descriptors of 128 dimensions. To have a intuitive understanding of the statistical order of $N_{SIFT}$, we visualized the histogram of $N_{SIFT}$ of the AwA2 dataset as a preliminary try. Figure 3 shows a histogram of $N_{SIFT}$.

As we can see, the average and median of the number of key points in a single image is usually large. Therefore, we need to set a constraint on the number of key points $N_{kp}$ selected. Figure 4 shows an example of detecting SIFT key points (annotated by red circles in figures) with different $N_{kp}$. Figure 4(a) is the non-constrained case, where all 6764 key points are selected. Then we tried different $N_{kp}$, namely 2500,500,100 in Figure 4(b)4(c)4(d) respectively. We can find that when $N_{kp}$ is relatively large, there is still a lot of background information included. However, when $N_{kp}$ is small, the key points will focus on the main parts in the image (*i.e.* animals in our images, see Figure 4(d)).

This is one extreme case – when the number of key point is so large that it includes too much background information. It is solved by constraining the number of key points detected in our experiment. Another extreme case is that, there is no key point in one image. This is due to the extreme smoothness of the input image. Figure 5 gives one example (from `dolphin_10180.jpg` in the AwA2 dataset). For this case, we have no choice but drop these images. Fortunately, this is the only example of this case.

[2] https://cvml.ist.ac.at/AwA2/

[3] https://opencv.org/

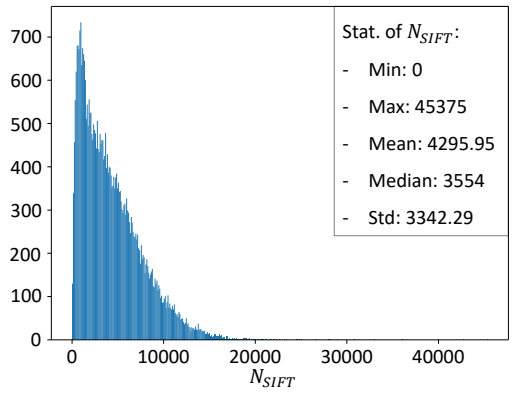


Figure 3: The distribution of $N_{SIFT}$ in AwA2 dataset, along with some of its statistical values.

(a) (b) (c) (d)

Figure 4: Constrain the number of key points $N_{kp}$ selected. The leftmost figure is SIFT without this constraint (in this case $N_{SIFT} = N_{kp} = 6764$), and the rest are $N_{kp} = 2500, 500, 100$ respectively.

After obtaining the local descriptors in each image, we can see each image as a bag of local descriptors. In the next step, we performed feature encoding (*i.e.* BoW, VLAD, Fisher Vector) to generate global descriptors of each image. The pipeline is to learn a codebook first, and then generate global features according to the clusters in the learned codebook.

Due to the constraint of memory and excessive computational cost, it is impossible to learn a codebook using all local descriptors throughout the dataset. Consider the case that there are 100 local descriptors in each image, then we need to perform clustering on a dataset of size 3732200 vectors if we use all the data. Therefore, in our experiments, we empirically chose the number of local descriptors sampled in each class be $5 \times N_{cluster}$, where $N_{cluster}$ is the cluster number. For example, when we learn a codebook with 128 clusters, a total of 32000 samples of local descriptors will be sampled out.

The generated global features are fed into an SVM with rbf kernel and $C = 1.0$ for classification. In our experiments, we mainly focus on the performance of different feature encoding methods, different settings of $N_{kp}$, and different settings of $N_{cluster}$.

Figure 5: An example of failing to detect even one single SIFT key point in one image.

First, we compare different settings of $N_{kp}$ for classification. In our experiment, we fix the number of clusters $N_{cluster}$ to be 128. Since the dimension of local descriptors is 128, the dimension of BoW, VLAD and Fisher are 128, 16384, 32768 dimension. For fair comparison and computational efficiency, we reduce the dimension to 128 by PCA in the latter 2 cases. We also perform L2 normalization to the features encoded before training the SVM. Table 1 shows the result. As we can see, a larger $N_{kp}$ will lead to higher performance. This is because when more key points in one image are considered, more types of feature patterns can be encoded. This result is in line with our intuition.

Table 1: Classification accuracy on training and testing set using different $N_{kp}$ with $N_{cluster} = 128$.

| | **BoW** | | **VLAD** | | **Fisher** | | **Super** | | **LLC** | |
|---|---|---|---|---|---|---|---|---|---|---|
| $N_{kp}$ | train | test | train | test | train | test | train | test | train | test |
| 10 | 38.6% | 12.4% | 44.2% | 12.9% | 38.8% | 12.5% | 32.2% | 32.9% | 12.8% | 12.5% |
| 100 | 46.1% | 20.8% | 55.4% | 25.7% | 47.2% | 21.9% | 52.5% | 52.3% | 22.7% | 22.9% |
| 500 | 45.7% | 27.7% | 60.1% | 35.1% | 48.0% | 58.2% | 54.2% | 52.9% | 32.8% | 32.5% |

Then, we considered the effect of changing the number of clusters $N_{cluster}$. We fix the number of key points extracted in each input image, and explore the performance difference of different $N_{cluster}$. When training SVM, for the case of $N_{cluster} = 128$, we reduce VLAD and Fisher features into 128 feature vectors. For $N_{cluster} = 256$, reduce to 256; For $N_{cluster} = 512$, reduce to 512. Table 2 and Table 3 show the result, which use $N_{kp} = 100$ and $N_{kp} = 500$ respectively. As we can see, though a larger cluster number may lead to an increase in performance, the increase is small. This is because the number of clusters is enough for different types of patterns, so a larger $N_{cluster}$ does not help much. We think $N_{kp}$ may determine the richness of local patterns, and $N_{kp}$ may determine a saturated $N_{cluster}$, for cluster numbers larger than which, the performance will not be boosted significantly.

Table 2: Classification accuracy on training and testing set using different $N_{cluster}$ with $N_{kp} = 100$.

| | **BoW** | | **VLAD** | | **Fisher** | | **Super** | | **LLC** | |
|---|---|---|---|---|---|---|---|---|---|---|
| $N_{cluster}$ | train | test | train | test | train | test | train | test | train | test |
| 128 | 46.1% | 20.8% | 55.4% | 25.7% | 47.2% | 21.9% | 52.5% | 52.3% | 22.7% | 22.9% |
| 256 | 57.0% | 21.7% | 65.0% | 25.9% | 59.2% | 22.7% | 59.9% | 59.5% | 24.2% | 23.7% |
| 512 | 68.4% | 22.3% | 75.3% | 24.2% | 70.3% | 23.5% | 65.3% | 64.2% | 30.3% | 29.5% |

Table 3: Classification accuracy on training and testing set using different $N_{cluster}$ with $N_{kp} = 500$.

| | **BoW** | | **VLAD** | | **Fisher** | | **Super** | | **LLC** | |
|---|---|---|---|---|---|---|---|---|---|---|
| $N_{cluster}$ | train | test | train | test | train | test | train | test | train | test |
| 128 | 45.7% | 27.7% | 60.7% | 35.1% | 48.0% | 28.2% | 54.2% | 52.9% | 32.8% | 32.5% |
| 256 | 56.5% | 28.8% | 68.9% | 35.3% | 59.1% | 29.8% | 58.3% | 55.9% | 39.1% | 39.8% |
| 512 | 66.7% | 30.1% | 77.8% | 34.3% | 69.3% | 30.4% | 57.3% | 58.8% | 41.3% | 40.4% |

The comparison of different methods of feature encoding can also be seen in Table 123. For performance, VLAD usually performs the best. BoW may suffer from insufficient encoded information, while Fisher may suffer from the non-convergence of the GMM. We did not record the time cost, but we found Bow is the fastest and LLC is the slowest. With the trade off between time cost and performance, we believe VLAD is a good feature encoding method.

### 4.3 Selective Search

In our experiments, we used the `selectivesearch` package[4] in Python to extract region proposals. In our experiments, we used one NVIDIA GeForce RTX2080Ti GPU to train the models. We have done two parts of experiments in this section, as is shown in Figure 6. For the baseline, we directly train ResNet-14/32[3] for the training set with 60% data and test the model on the testing set containing the rest 40%. Their performance on testing set is 78.5% and 77.6% respectively. Then we compare this end-to-end training with feature encoding methods.

[4]https://github.com/AlpacaDB/selectivesearch

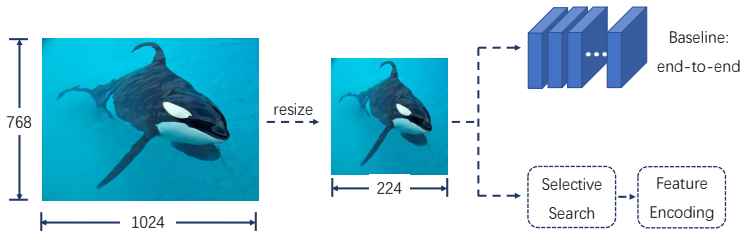


Figure 6: For comparison of the deep feature encoding method, we learned an end-to-end model to compare its result with selective search.

Selective search is a useful method to extract region proposals in an unsupervised manner. For the sake of convenience, we just perform selective search on the resized ($224 \times 224$) images. Then one problem is how to transform these region proposals into feature vectors. We searched some literatures on region proposal and feature encoding and did not find a standard way. Therefore, we defined 3 intuitive ways to achieve this.

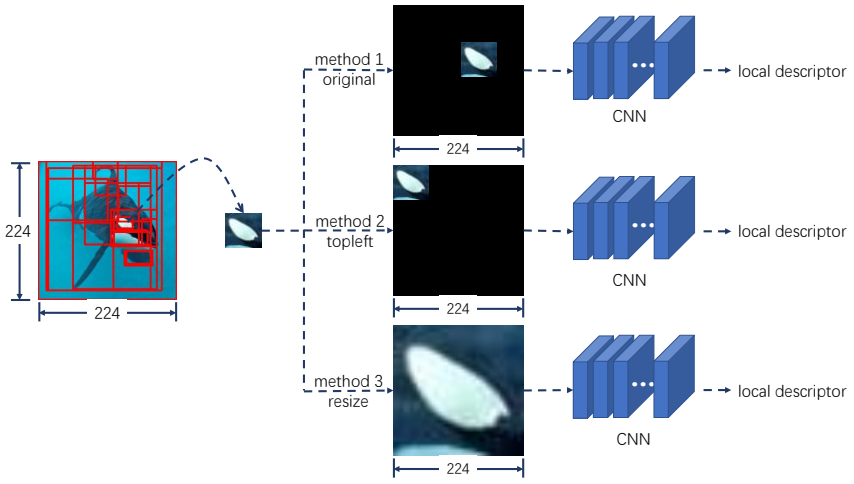


Figure 7: The pipeline of how to transform one region proposal to its corresponding local descriptor. We have tried 3 methods: (1) original (2) topleft (3) resize.

As is shown in Figure 7, for each region proposal, we can use the 3 methods to transform it into a fix-length feature vector. The first method (termed *original*) is to force other pixels except for the region to zero. This method not only preserves the pattern information but positional information as well. The second method (termed *topleft*) is to move the proposed region to the topleft of the input. In this way, we only focus on the deep patterns of proposed regions. The third method (termed *resize*) is to resize the proposed region to ($224 \times 224$). This is an intuitive method to let the input fit in with the input requirement of the DNN.

Then we can extract features using a DNN. In this part we used the ResNet-34 pretrained on the 60% AwA2 dataset, as is shown in Figure 7. Table 4 shows the result. As we can see, the performance of *original* and *topleft* is similar to each other. This may indicate that DNNs mainly focus on deep features inside the input image, instead of the position of a certain pattern inside an image. This may in some way explain the success of DNNs.

Another thought is to use different DNNs to extract features from the region proposals. For fair comparison, we changed the AwA2 pretrained ResNet34 into an ImageNet pretrained one. The results are shown in Table 5. As we can see, the performance is better than results in Table 4. This is because models trained on ImageNet may encode more feature patterns, while models trained merely on AwA2 dataset may encode irrelevant noises, which leads to overfitting.

Table 4: Classification accuracy on training and testing set using the **AwA2 pretrained ResNet34** and different methods.

| | **BoW** | | **VLAD** | | **Fisher** | |
|---|---|---|---|---|---|---|
| | train | test | train | test | train | test |
| original | 95.7% | 69.5% | 91.6% | 68.7% | 26.2% | 8.9% |
| topleft | 93.5% | 69.9% | 90.6% | 68.5% | 31.7% | 12.2% |
| resize | 38.9% | 14.3% | 48.1% | 16.0% | 35.1% | 12.8% |

Table 5: Classification accuracy on training and testing set using the **ImageNet pretrained ResNet34** and different methods.

| | **BoW** | | **VLAD** | | **Fisher** | |
|---|---|---|---|---|---|---|
| | train | test | train | test | train | test |
| original | 87.3% | 80.3% | 87.4% | 82.9% | 81.2% | 65.1% |
| topleft | 87.7% | 82.2% | 87.2% | 83.3% | 52.1% | 32.9% |
| resize | 38.2% | 17.3% | 54.6% | 24.2% | 37.4% | 15.3% |

### 4.4 LLC

We use implementation of LLC in paper [1], instead of original paper for simplicity. We set parameters $M = 32$ and $\beta = 1e^{-6}$. Notice that this version of LLC is does not iteratively optimize codebook $B$ and coding $C$. Instead, it directly takes clustering as codebook, then calculate analytical solution of coding. We use GMM with $covariance\_type =' diag'$ as clustering method. In our experiment, we find that eocoding process of LLC is very time-consuming, it cost about 10x time than Super vector. In calculation of LLC coding, it requires matrix multiplication, matrix inverse and sorting, which are all complex operations. However, the accuracy of LLC is similar with BoW, which is a very efficient algorithm.

## 5 Conclusion

In this report, we implement and evaluate different local descriptor extraction algorithms and feature encoding algorithms. Specifically, we experiment SIFT and Selective Search to extract local descriptors, and we use BoW, VLAD, Fisher vector, Super vector and LLC to encode local features. We observe that Selective Search can effectively enhance final accuracy of classifier, and that VLAD is optimal algorithm considering time-accuracy trade off.

## References


[1] Ken Chatfield, Victor S Lempitsky, Andrea Vedaldi, and Andrew Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *BMVC*, volume 2, page 8, 2011.

[2] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] F. Jurie and B. Triggs. Creating efficient codebooks for visual recognition. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 604–610 Vol. 1, 2005.

[5] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, 2010.

[6] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[7] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[8] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3): 222–245, 2013.

[9] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[10] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3360–3367. IEEE, 2010.

[11] Xi Zhou, Kai Yu, Tong Zhang, and Thomas S. Huang. Image classification using super-vector coding of local image descriptors. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 141–154, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15555-0.