Shallow vs. Deep Image Representations: A Comparative Study with Enhancements Applied For The Problem of Generic Object Recognition

Yasser M. Abdullah *yasware@gmail.com*
Faculty of Engineering/Department of IT
Aden University
Aden, Yemen

Mussa M. Ahmed *mussa_m7@yahoo.com*
Faculty of Engineering/Department of ECE
Aden University
Aden, Yemen

Abstract

The traditional approach for solving the object recognition problem requires image representations to be first extracted and then fed to a learning model such as an SVM. These representations are handcrafted and heavily engineered by running the object image through a sequence of pipeline steps which requires a good prior knowledge of the problem domain in order to engineer these representations. Moreover, since the classification is done in a separate step, the resultant handcrafted representations are not tuned by the learning model which prevents it from learning complex representations that might would give it more discriminative power. However, in end-to-end deep learning models, image representations along with the classification decision boundary are all learnt directly from the raw data requiring no prior knowledge of the problem domain. These models deeply learn the object image representation hierarchically in multiple layers corresponding to multiple levels of abstraction resulting in representations that are more discriminative and give better results on challenging benchmarks. In contrast to the traditional handcrafted representations, the performance of deep representations improves with the introduction of more data, and more learning layers (more depth) and they perform well on large-scale machine learning problems. The purpose of this study is six fold: (1) review the literature of the pipeline processes used in the previous state-of-the-art codebook model approach for tackling the problem of generic object recognition, (2) Introduce several enhancements in the local feature extraction and normalization steps of the recognition pipeline, (3) compare the enhancements proposed to different encoding methods and contrast them to previous results, (4) experiment with current state-of-the-art deep model architectures used for object recognition, (5) compare between deep representations extracted from the deep learning model and shallow representations handcrafted through the recognition pipeline, and finally, (6) improve the results further by combining multiple different deep learning models into an ensemble and taking the maximum posterior probability.

Keywords: Shallow Models, Deep Learning Models, Encoding Methods, Object Recognition, BoVW.

1. INTRODUCTION

Generic object recognition problem is simply to assign a label for an object image. The image may contain multiple objects (such as elephants, leopards, sunflowers, grand pianos, faces, etc.) and hence multiple labels should be assigned accordingly. This problem is one of the most fundamental problems in computer vision and pattern recognition and has wide range of applications like web content analysis and video surveillance. However, it is a challenging

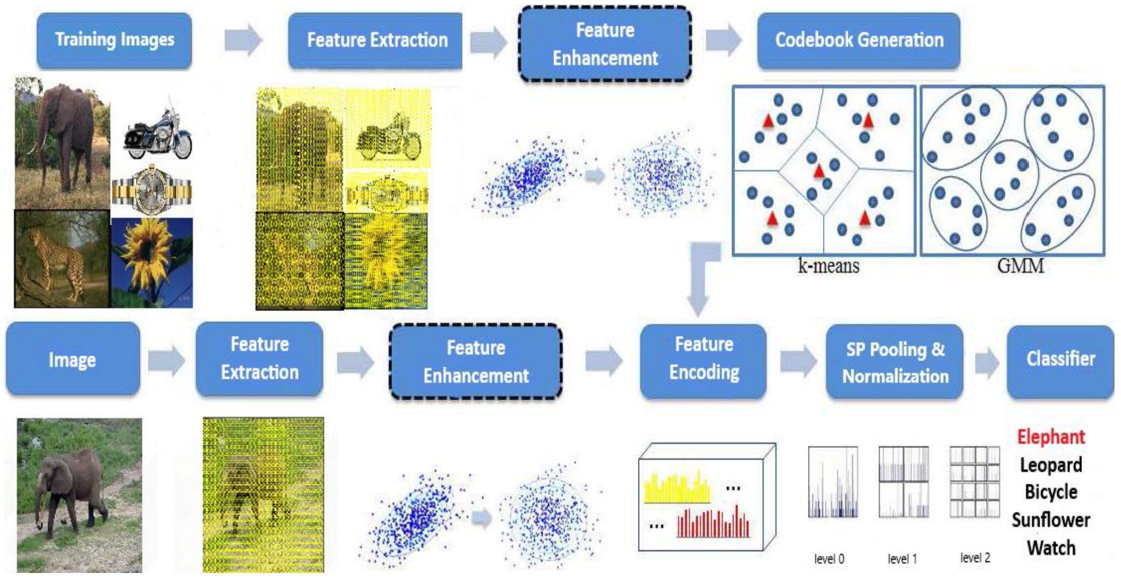


FIGURE 1: Bag of Visual Words (BoVW).

problem especially in the presence of intra-class variation, clutter, occlusion, deformation, illumination and viewpoint changes.

The best known shallow framework used for solving this problem is the Bag of Visual Words (BoVW) [1, 2] model which makes use of a pretrained codebook as shown in figure 1. BoVW passes the query image through a pipeline consisting of several steps to get the final representation describing the image. It starts out by extracting local features of the object image. The local features are detected and described to get local feature descriptors. Many feature detection and description methods were proposed in the literature and are reviewed in section 2. In a later step, these features are transferred from their feature space onto the codebook space using an encoding method. For a particular feature descriptor, the encoding methods can produce a single code or a block of codes for each codeword in the codebook. Many encoding algorithms were proposed in the literature and are discussed in section 4. To get a global representation of the image, the coding responses for each codeword is then integrated into a single code (or a block of codes) using a pooling method like sum or max pooling. Pooling process can be improved if performed spatially using a Spatial Pyramid (SP) [3]. At the end of the pipeline, the resultant image representation is normalized using a normalization method like ℓ_2 or power normalization. At this stage, the image hand-engineered representation is ready and can be fed to a machine learning model (e.g., SVM) to give the class of the query image.

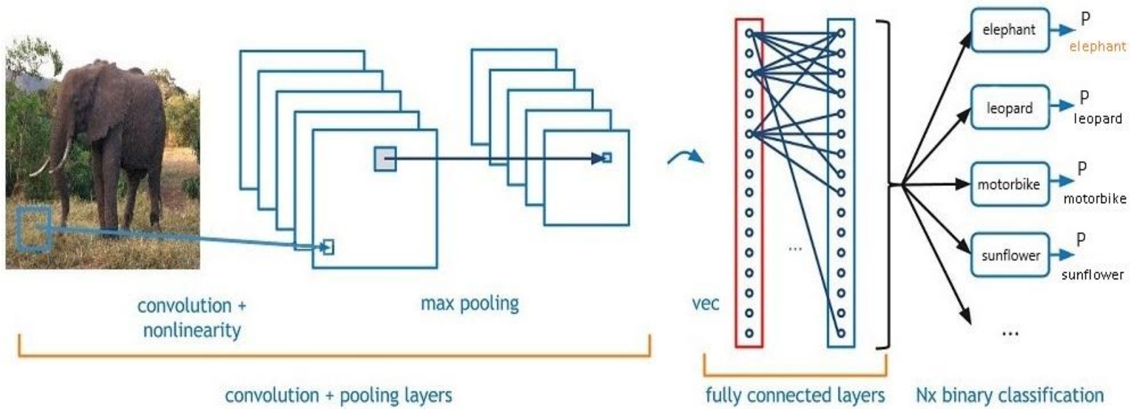


FIGURE 2: Deep model architecture (CNN).

To improve the performance of shallow representations, we proposed a new pipeline step in which the previously extracted local features are enhanced (see figure 1). In this step, feature descriptors are square-rooted as suggested by Arandjelović and Zisserman [4] and are reduced in dimensionality and decorrelated using Principal Component Analysis (PCA). Additionally, they are augmented by their spatial location [5]. BoVW model will be discussed in section 3.

On the contrary, a deep learning model doesn't need features to be extracted beforehand but the object image raw pixels can be fed directly to the deep model for both feature extraction and classification in a single step. In this model, features are deeply leant inside the deep model in multiple learning layers and no prior knowledge about the problem domain is required. Deep models are sometimes called generalized feature extractors since they can be used to extract features for different data domains such as text, audio and video. Figure 2 displays a deep model architecture used for object recognition and is called a convnet or Convolutional Neural Network (CNN). CNNs start learning directly from the input image and succeeding layers consume the output of the preceding layer(s). Mainly, they consist of many Learning layers, some pooling layers, and one classification layer. A learning layer can be a convolutional layer or a fully connected layer (FC). Convolutional layers are many in number and can accept and produce multidimensional data whereas FC layers are a few and only accept and produce one-dimensional data. FC layers can exist only at the end of the network whereas convolutional layers are dispersed throughout the network. Inside the learning layers, there exist a nonlinearity to help the network learn interesting functions. They also contain many free parameters that are adjusted during the supervised training process. The classification layer (final layer) computes the loss that the network has incurred in learning non-discriminative representations. The network then updates its free parameters so as to produce more discriminative representations to bring the network loss down. To reduce the dimensionality of the convolutional layer output, a pooling layer is used. In addition to dimensionality reduction, it summarizes local inputs by one statistic that gives the network invariance to simple translations. Deep learning and its models will be discussed in section 5.

In this study, we have conducted many experiments to compare the performance of shallow and deep representations. We evaluated our proposed enhancements to different encoding methods used in the BoVW codebook model and obtained better results than before. Also, we experimented with deep model architectures and compared between different architectures on one hand and between deep and shallow models on another hand. We studied further the discriminative power of deep and shallow features extracted from deep and shallow models respectively. Finally, we improved the results further by combining several deep models into an ensemble and taking the maximum posterior probability. All of experiment details are mentioned in section 6. Conclusion, future work and references are given in sections 7, 8 and 9 respectively.

2. LOCAL FEATURES

For any object image, local features can be interest points or interest regions. They are extracted from the appearance of an object and are local in the sense that they describe a local part of the object appearance. An image can produce several hundred (or thousands) of local features. The next subsections review the detection and description methods used to extract and describe image local features.

2.1 Feature Detectors

Many feature detectors were proposed in the literature such as Scale-Invariant Feature Transform (SIFT [6]) and Speeded Up Robust Features (SURF [7]). Feature detection is about detecting points (or regions) in an image that are repeatable – i.e., given a different image of the same object, the feature is distinctive enough that we can find it again in the correct location. A lot of methods were used for the detection process. Harris [8] has used the second moment matrix (Harris matrix) which contains image first derivatives in order to detect corner points. He used a corner quality measure that is based on eigenvalues of this matrix (Harris measure). To detect a feature at multiple spatial scales, a Gaussian scale-space for an image is constructed and features are detected at all scales. To prevent multiple detections for the same feature and select the scale that is most significant for the feature, Lindeberg [9] introduced the concept of automatic scale selection which allows to detect interest points in an image, each with its own characteristic scale. The characteristic scale is the one that gives the maximum of the scale-normalized Laplacian of Gaussian function in scale space. Lindeberg [9] also proposed another detection method that is based on the scale-normalized Hessian matrix of image second derivatives. In this method, an interest point is detected if the trace of this matrix (which is the scale-normalized Laplacian of Gaussian or LoG) is locally maximal in scale space. Lowe [6], in his SIFT detector, approximated the Laplacian of Gaussian (LoG) detector using the Difference of Gaussians (DoG) which is computationally more efficient. Another measure of detection is to use the determinant of the Hessian (DoH) as a feature quality measure. Mikolajczyk and Schmid [10] refined the previous methods, creating robust and scale-invariant feature detectors with high repeatability, which they coined Harris-Laplace and Hessian-Laplace. They used a scale-adapted Harris measure or the determinant of the Hessian matrix to select the location, and the Laplacian to select the scale. Bay et al. [7] noted that the discrete Gaussian filters used in the computation of the scale-normalized Hessian could be approximated by extremely simple box filters involving simple sums and differences of pixels and have used this idea, in their SURF detector, to approximate the determinant of the Hessian. The box filters can be applied very quickly compared to filters with floating-point coefficients. Moreover, if integral images [11] are used for the computation, then the speed of applying the box filter is independent of the filter size resulting in what they called Fast Hessian. Rosten and Drummond [12] proposed a fast detection algorithm for what they called FAST Corners. In their method, a candidate pixel p is compared to a discretized circle of pixels around it; if all the pixels on a contiguous arc of n pixels $(n = 12)$ around the circle are significantly darker or brighter than the candidate pixel, it is detected as a feature. They [13] later extended the FAST idea using a machine learning approach (a decision tree) based on the intensities of the sixteen surrounding pixels of the candidate pixel to yield a detector that is higher in performance and speed. Matas et al. [14] proposed a new type of feature called Maximally Stable Extremal Regions or MSERs. These too are extremely fast to compute and are based on the basic thresholding operation. A region (connected component) is detected as feature if it satisfies two conditions: (1) all the pixels inside that region are either all darker or all brighter than pixels in the boundary, (2) the region should be stable, i.e., it should change little to threshold variations.

It is important to mention that most of the above feature detectors are not robust to large viewpoint changes. If the viewpoint of an object undergoes a large affine transformation, then the region of the same pixels around the interest point in both cases would be different. The fundamental theory of affine-invariant regions was first proposed by Lindeberg and Gårding [15] and applied by other researchers including Baumberg [16], Schaffalitzky and Zisserman [17], and Mikolajczyk and Schmid [18]. An elliptical affine invariant region can be computed by an iterative procedure called affine adaptation. After applying this procedure, the resulting circular region and that region before the transformation are identical but up to a rotation. Mikolajczyk and Schmid [18] proposed to simultaneously detect feature point locations and corresponding affine-invariant regions using an iterative algorithm, resulting in Harris-Affine or Hessian-Affine features according to the type of detector used whether Harris or Hessian.

After determining the feature location and scale, the next problem for the detector is to determine the feature support region - i.e., the set of pixels in the feature neighborhood that should contribute to a feature’s descriptor. For scale-invariant features such as Harris-Laplace, Hessian-Laplace, and DoG where features are detected at a characteristic scale, the support region can be a circle drawn with a radius proportional to the feature characteristic scale. For features like MSERs and Hessian-Affine which produce affine-covariant regions, we can use the circular region produced at the end of the affine adaptation process as a support region. However, many description methods assume a square patch is given around the feature location as opposed to a circular one which means that we need to assign a reliable orientation to the feature to define the top edge of the square.

2.2 Feature Descriptors

Once a feature's location (and perhaps some additional information such as its scale or support region) has been determined, the next step is to describe the feature with a vector of numbers called a descriptor. Throughout the literature, large number of feature descriptors were proposed. Lowe's SIFT [6] descriptor used a square patch around the feature point as opposed to a circular one which requires a reliable orientation for the square patch to be calculated. He suggested estimating this orientation based on a histogram of pixel gradient orientations over the support region of a scale-invariant circle and taking the dominant gradient orientation as the patch orientation. The square patch is then rotated upwards to normalize its direction, resampled, and smoothed by a Gaussian. The gradient at each resampled pixel is estimated and weighted by a Gaussian. The square is then subdivided into 4×4 smaller sub-squares with one histogram of eight direction bins in each sub-square. These sixteen histograms are then concatenated to form a $4 \times 4 \times 8 = 128$-dimensional vector which is normalized twice to make it robust to illumination changes. Mikolajczyk and Schmid [19] proposed a SIFT variant called GLOH (Gradient Location and Orientation Histogram). Instead of the square grid, a log polar grid of three rings is created. The outer and second outer rings are subdivided into eight bin locations each whereas the center ring is undivided to give a total of seventeen bin locations. In each bin location, a histogram of sixteen quantized directions is computed. Histograms from different bin locations are concatenated to form a raw $16 \times 17 = 272$-dimensional descriptor. The dimensionality of the descriptor is then reduced using PCA to give a 128-dimensional vector. SURF descriptor is very efficient to compute because it uses Haar wavelets which are simple box filters. As in SIFT, the oriented square at a feature's detected scale is split into a 4×4 square grid. However, instead of computing gradient orientation histograms in each subsquare, Haar wavelet responses at twenty-five points in each subsquare are computed. The sums of the original and absolute responses in the x and y directions are computed in each sub-square, yielding a $4 \times 4 \times 4 = 64$-dimensional descriptor. Ke and Sukthankar [20] addressed the problem of dimensionality reduction of the SIFT descriptor using PCA in what they coined PCA-SIFT. They collected a large number of DoG keypoints and constructed 41×41 patches at the estimated scale and orientation of each keypoint. The x and y gradients at the interior pixels of each patch were collected into a $39 \times 39 \times 2 = 3042$-dimensional vector, and PCA was applied to determine a much smaller number of basis vectors (e.g., twenty or thirty-six). Thus, the high-dimensional vector of gradients for a candidate feature is represented by a low-dimensional descriptor given by its projection onto the learned basis vectors. Belongie et al. [21] proposed shape contexts as a method for matching shapes, which were modified by Mikolajczyk and Schmid [19] for feature point description. The approach is similar to GLOH in that a log-polar location grid is constructed at the feature point location. However, instead of using the gradients of all points to construct the histograms in each subregion, only edge points detected with the Canny detector [22] are allowed to contribute their gradient orientations. Each edge point's contribution is further weighted by its gradient magnitude. Johnson and Hebert [23] originally proposed spin images for describing features in range data. Lazebnik et al. [24] proposed modifying them to create feature descriptors for grayscale images. We simply compute a histogram of quantized intensities for each of several rings around the feature location, after the intensities have been normalized. The dimension of the descriptor is the number of intensity bins times the number of rings. Since there are no angular subdivisions of the rings, the descriptor is rotation-invariant.

Another class of descriptors that do not require the patch orientation estimation and explicit orientation, are called Invariant-based descriptors. They are based on invariant functions of the patch pixels with respect to a class of geometric transformations, typically rotations or affine transformations. Schmid and Mohr [25] popularized the idea of differential invariants for constructing rotation invariant descriptors. That is, the descriptor is constructed using combinations of increasingly higher-order derivatives of the Gaussian smoothed image. Moment-invariant [26] descriptors are another type of invariant descriptors that are computed using the image intensities and spatial locations.

3. BAG OF VISUAL WORDS MODEL

The first decade of this century has seen the rise of Bag of Visual Words (BoVW) (or Bag of Features (BoF)) in computer vision and has been applied to many problems such as object recognition and image retrieval. BoVW model was developed from the Bag of Words model used in document classification and is probably the most popular and effective shallow framework for object classification. In this model, an object image is passed through a pipeline consisting of typically five steps to give at the end the image final handcrafted representation (see figure 1). The five pipeline steps are (1) local feature extraction, (2) local feature enhancement, (3) feature descriptor encoding, (4) code pooling and finally, (5) normalization. In the first step, patches from an image are extracted and represented using feature descriptors such as SIFT. The extraction process employs different sampling strategies which can be dense or sparse. In dense sampling, a patch is extracted and represented at every n pixels on a fine fixed grid while in sparse sampling, patches are extracted only at interest points or regions detected by a feature detector. Multiscale features can also be extracted by using different patch sizes. It is worth noting that the sets of local descriptors produced for different images may have different cardinality especially if sparse sampling is used or if images have different resolutions.

The second step of the pipeline is optional, but we have seen much improvement in the final results when employed. To enhance the features, we first square-root their descriptors. After that, we reduce their dimensionality by learning from the training data a set of basis vectors using PCA and then projecting the features onto a subset of these basis vectors that has the highest projection variance. Feature descriptors are then decorrelated by whitening their descriptor dimensions. Moreover, the feature spatial location information is embedded inside its descriptor [5]. Let $\mathbf{x}_n$ be a feature descriptor, then the feature is augmented by its normalized spatial location: $[\mathbf{x}_n^{\mathrm{T}}, \frac{x}{h} - 0.5, \frac{y}{w} - 0.5]^{\mathrm{T}}$ where (x, y) is the descriptor $\mathbf{x}_n$ spatial location, and $h \times w$ are the image dimensions.

In the third step of the pipeline, the set of local features extracted in previous steps are encoded using an encoding process which makes use of a previously generated codebook (also called dictionary or vocabulary). The codebook is generated in an offline process using an unsupervised learning method such as k-means clustering [27, 28] or Gaussian Mixture Model (GMM, [29]). Local feature descriptors from all training images are clustered in feature space into k clusters to form the codewords of the codebook. Hence, the inputs to the encoding process are the image local feature descriptors and the previously learnt codebook. The encoding process in turn produces a coding matrix per image, one coding vector for each local descriptor, by using one of different encoding methods. Feature coding is a key component of object recognition and has been widely studied in the past several years. Various encoding methods were proposed and they differ in how they activate the codewords for a given local feature. These algorithms encode a feature descriptor by producing a response for each codeword in the dictionary. The coding response can have different dimensionality for different encoding methods. Encoding methods will be discussed in section 4.

The fourth step of the pipeline (pooling process) converts the image coding matrix into a vector (called the pooling vector) which forms the image global representation. For each codeword, codes from multiple local features are integrated dimension-wise into a single code (or a block of codes) using one of the classic pooling methods. Common examples include sum pooling and max pooling. Let C_n^m be the response of descriptor x_n for codeword m, and h^m be the pooling response for codeword m. In sum pooling, responses corresponding to the same codeword are summed for all descriptors, i.e., $h^m = \sum_n c_n^m$, whereas in max pooling, the maximum of the codes is taken, i.e., $h^m = \max_n c_n^m$.

In the last pipeline step, the image raw representation is normalized to form the image final representation. Different normalization methods exist such as l_1-normalization, l_2-normalization, power normalization, and intra-normalization [30]. In l_1-and l_2-normalization, the representation dimensions are divided by the corresponding l_1-, or l_2- norm respectively. In power normalization,

Encoding Algorithm	HA	SA	LSA	FV	VLAD	SPC	LCC	LLC	App. LLC	SC	GSC	LTC	SVC
$\mathbf{c}_{\boldsymbol{n}}^{\boldsymbol{m}}$ **dimensionality** $(\boldsymbol{R})$	1	1	1	$2D$	D	1	1	1	1	1	1	$H+1$	$D+1$
No. of codewords activated	1	M	K	M	1	$*$	$*$	M	K	1	K	$*$	1

TABLE 1: Coding response for different encoding methods. Top row: Coding Response dimensionality per codeword. Bottom row: number of codewords activated by a single feature descriptor. Star symbol ($*$) indicates a sparse set of codewords activated per feature descriptor and might be variable.

we apply to each dimension of the representation the following function: $sign(\boldsymbol{p}(i))|\boldsymbol{p}(i)|^{\alpha}$, where $\boldsymbol{p}(i)$ is the i^{th} dimension of the representation, and $0 \leq \alpha \leq 1$. Sign-square-root normalization is a special case of power normalization when $\alpha = 0.5$. Intra-normalization carries out normalization in a block by block manner and can be done using l_1- or l_2- normalization. Intra-normalization can be applied when the coding process produces a block of codes per codeword. We proposed different normalization strategies for different encoding methods which are based on l_2- and sign-square-root normalization and are detailed in section 6.1.

3.1 Spatial information

Generally, the BoVW model is orderless and hence discards all the information about the location of the patches and this in turn incurs a loss of information. The dominant approach to include spatial information is to use the Spatial Pyramid (SP). Inspired by the pyramid match kernel of Grauman
and Darrell [31], Lazebnik et al. [3] proposed to partition an image into a set of regions in a course-to-fine manner. They propose to partition the image into 1×1, 2×2, and 4×4, for a total of 21 regions. Each region is described independently and the region-level pooling vectors are then concatenated into an image-level pooling vector. Marszalek et al. [32] suggested a different partitioning strategy in which the image is partitioned into 1×1, 1×3, and 2×2, for a total of 8 regions.

3.2 Classification

After running the image through the recognition pipeline, the image final representation is ready for classification. Many machine learning models were tried throughout the literature but the most popular one is the SVM which gives better results when used in conjunction with BoVW model. The SVM searches the space of linear decision boundaries to find the one that gives the maximum margin between two binary classes. It optimizes a constrained convex quadratic problem that uses the dot product between input features (linear kernel) and hence the kernel trick can be used. The nonlinear kernel function transfers the input features into a higher dimensional space and computes the dot product there in that space. Since the mapping process is done implicitly, we benefit from the increase in dimensionality by noting that problems become more likely to be linearly separable in high dimensional spaces. Moreover, the dot product is computed in the mapped space by computing the kernel function in the original space.

Many non-linear kernels for BoVW model were proposed and used throughout the literature such as the histogram intersection kernel, pyramid match kernel, chi-square kernel, and Hellinger's kernel. Despite their better classification results, they are less efficient than the linear kernel. There exist a class of kernels (additive homogeneous kernels [33]) that are as efficient as linear ones up to the computation of an efficient explicit feature map. Several non-linear kernels can be approximated by explicitly computing their feature maps.

4. ENCODING METHODS

Let $\mathbf{X}$ be a set of D-dimensional local descriptors extracted from an image, i.e., $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D\times N}$, $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_M] \in \mathbb{R}^{D\times M}$ be the codebook of M codewords and $\mathbf{C}_n = [(\mathbf{c}_n^1)^{\mathrm{T}}, \dots, (\mathbf{c}_n^M)^{\mathrm{T}}] \in \mathbb{R}^{RM}$ be the coding vector produced by the encoding method corresponding to feature descriptor $\mathbf{x}_n$, where $\mathbf{c}_n^m$ is the coding response of the m^{th} codeword and R is the coding response dimensionality which can be a scalar or a block of codes. Each local descriptor $\mathbf{x}_n$ may activate one or more codewords during the encoding process. The dimensionality of the coding response per codeword and the number of activated codewords for a feature descriptor are different for different encoding methods and are given in table 1.

4.1 Hard Assignment (HA)

This coding method [1] forms the baseline encoding upon which other encoding methods improve. It is also called by other names like Vector Quantization (VQ) or histogram encoding. HA encodes a feature descriptor $\mathbf{x}_n$ by assigning the whole coding weight to the nearest codeword. HA is defined as:

$$c_n^m = \begin{cases} 1, & if\ m = \underset{i}{argmin}(\|\mathbf{x}_n - \mathbf{b}_i\|_2) \\ 0, & otherwise \end{cases} \quad , i = 1, \dots, M \tag{1}$$

4.2 Soft Assignment (SA)

Instead of assigning all the coding weight to the closest codeword, this method [34-36] distributes the coding weight among all the codewords in a soft manner proportional to how far each codeword is from the feature descriptor. It achieves this by making use of a distance function like the Gaussian kernel. In contrast to HA, SA takes into account the uncertainty of codewords in case two or more codewords are strong candidates for a given feature descriptor. SA is defined as:

$$c_n^m = \frac{\exp(-\beta\|\mathbf{x}_n - \mathbf{b}_m\|_2^2)}{\sum_{i=1}^{M}\exp(-\beta\|\mathbf{x}_n - \mathbf{b}_i\|_2^2)}, \tag{2}$$

where β is a smoothing factor controlling the softness of the assignment.

4.3 Localized Soft Assignment (LSA)

Unlike SA, this method [37] only considers the K closest codewords in the neighborhood of the feature descriptor. Let $\mathbf{B}_n = [\,\hat{\mathbf{b}}_k]_{k=1}^{K}$, be the K closest codewords to the feature descriptor $\mathbf{x}_n$. LSA is defined as:

$$c_n^m = \begin{cases} \psi(\mathbf{x}_n), & if\ \mathbf{b}_m \in \mathbf{B}_n \\ 0, & otherwise, \end{cases} \tag{3}$$

$$\psi(\mathbf{x}_n) = \frac{\exp(-\beta\|\mathbf{x}_n - \mathbf{b}_m\|_2^2)}{\sum_{k=1}^{K}\exp\left(-\beta\left\|\mathbf{x}_n - \hat{\mathbf{b}}_k\right\|_2^2\right)},$$

4.4 Fisher Vector

Fisher coding [38] is inspired by the Fisher kernel which describes a signal with a gradient vector derived from its generative probability density function [39]. In the context of object recognition, the signal is the image, the gradient vector is used for feature coding, and the probability density function is a Gaussian Mixture Model (GMM). A GMM of M components with parameters $\boldsymbol{\theta} = \{\,\boldsymbol{\theta}_m\}_{m=1}^{M}$, is given by:

$$p(\mathbf{x}_n|\boldsymbol{\theta}) = \sum_{m=1}^{M} w_m u_m(\mathbf{x}_n|\boldsymbol{\theta}_m), \tag{4}$$

where $u_m(\mathbf{x}_n|\,\boldsymbol{\theta}_m)$ denotes Gaussian □:

$$u_m(\mathbf{x}_n|\,\boldsymbol{\theta}_m) = \frac{exp\left\{-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_m)^{\mathrm{T}}\boldsymbol{\Sigma}_m^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_m)\right\}}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_m|^{1/2}}, \tag{5}$$

and $\{w_m, \boldsymbol{\mu}_{\mathrm{m}}, \boldsymbol{\Sigma}_{\mathrm{m}}\}$ denote the weight, the mean vector, and the covariance matrix of Gaussian m respectively and can be estimated using Expectation Maximization (EM) algorithm. We require $\forall_m: w_m \geq 0$ and $\sum_{m=1}^{M} w_m = 1$ to ensure $p(\mathbf{x}_{\mathrm{n}}|\,\boldsymbol{\theta})$ is a valid distribution. We assume a diagonal covariance matrix so $\boldsymbol{\Sigma}_m$ is reduced to a variance vector denoted by $\boldsymbol{\sigma}_m^2$. Supposing all the features are independent from each other, an image can be expressed as the log-likelihood of all the features:

$$l(\mathbf{X}\,|\boldsymbol{\theta}) = \sum_{n=1}^{N} log\ \ p(\mathbf{x}_n|\boldsymbol{\theta}). \tag{6}$$

We can define Fisher vector as the gradient of l with respect to $\boldsymbol{\theta}$ normalized by the square root of the inverse of Fisher information matrix $\mathbf{F}_{\boldsymbol{\theta}}$, i.e.,

$$\boldsymbol{\mathscr{g}} = \mathbf{F}_{\boldsymbol{\theta}}{}^{-1/2}\mathbf{g}, \tag{7}$$

where $\mathbf{g} = \{\partial l/\partial \boldsymbol{\mu}_m, \partial l/\partial \boldsymbol{\sigma}_m\}_{m=1}^{M}$. The gradient with respect to the weight parameter was omitted since it adds a little information [38]. Note that the Fisher information matrix $\mathbf{F}_{\boldsymbol{\theta}}$ has an approximated closed-form solution, and $\mathbf{F}_{\boldsymbol{\theta}}{}^{-1/2}$ normalization corresponds to the whitening of the dimensions. FV encoding is defined as:

$$\mathbf{c}_n^m = \left[\boldsymbol{\mathscr{g}}_{\mu_m}^{\mathrm{T}}, \boldsymbol{\mathscr{g}}_{\sigma_m}^{\mathrm{T}}\ \right], \tag{8}$$

$$\boldsymbol{\mathscr{g}}_{\mu_m}^{\mathrm{T}} = \frac{1}{\sqrt{w_m}}\gamma_n(m)\left(\frac{\mathbf{x}_n - \boldsymbol{\mu}_m}{\boldsymbol{\sigma}_m}\right),$$

$$\boldsymbol{\mathscr{g}}_{\sigma_m}^{\mathrm{T}} = \frac{1}{\sqrt{2w_m}}\gamma_n(m)\left(\frac{(\mathbf{x}_n - \boldsymbol{\mu}_m)^2}{\boldsymbol{\sigma}_m^2} - 1\right),$$

$$\gamma_m(m) = \frac{w_m u_m(\mathbf{x}_n|\ \boldsymbol{\theta}_m)}{\sum_{i=1}^{M} w_i u_i(\mathbf{x}_n|\ \boldsymbol{\theta}_i)}.$$

4.5 Vector of Locally Aggregated Descriptors (VLAD)

In FV, each codeword is represented by its first and second order statistics. In contrast, VLAD [30, 40] can be viewed as a hard version of FV and it only keeps the first order statistics. It encodes a feature descriptor $\mathbf{x}_n$ by the vector difference to its closest codeword. VLAD is defined as:

$$\mathbf{c}_n^m = \begin{cases} (\mathbf{x}_n - \mathbf{b}_m)\,, & if\ m = \mathrm{argmin}_i(\|\mathbf{x}_n - \mathbf{b}_i\|_2) \\ \ 0, & otherwise \end{cases}\quad , i = 1, \dots, M \tag{9}$$

4.6 Sparse Coding (SPC)

SPC [41] finds a sparse set of codewords with corresponding coefficients $\mathbf{r}_n$ that can reconstruct the input feature descriptor $\mathbf{x}_n$. SPC solves an l_1 regularized least-square optimization problem defined as:

$$\arg\min_{\mathbf{r}_n}\ \|\mathbf{x}_n - \mathbf{B}\mathbf{r}_n\|_2^2 + \lambda\|\mathbf{r}_n\|_1 \tag{10}$$

$$\text{s.t } \mathbf{1}^{\mathrm{T}}\mathbf{r}_n = 1.$$

Sparsity is attained in $\mathbf{r}_\mathrm{n}$ due to the l_1 norm which is known to induce sparsity. SPC is defined as:

$$c_n^m = \mathbf{r}_n(m), \tag{11}$$

where $\mathbf{r}_n(m)$ is the reconstruction coefficient of descriptor $\mathbf{x}_n$ for the codeword $\mathbf{b}_m$.

4.7 Local Coordinate Coding (LCC)

In practice, SPC tends to be local, i.e., it reconstructs a feature descriptor $\mathbf{x}_n$ using codewords in its neighborhood. But this locality cannot be ensured theoretically. Yu et al. [42] suggested a modification that explicitly encourages the coding to be local. They pointed out that under certain assumptions locality is more important than sparsity for successful nonlinear function learning using the obtained codes. LCC solves the following l_1 regularized least-square optimization problem:

$$\arg\min_{\mathbf{r}_n}\ \|\mathbf{x}_n - \mathbf{B}\mathbf{r}_n\|_2^2 + \lambda\sum_{m=1}^{M}|\mathbf{r}_n(m)|\ \|\mathbf{x}_n - \mathbf{b}_m\|_2^2 \tag{12}$$

$$\text{s.t } \mathbf{1}^{\mathrm{T}}\mathbf{r}_n = 1,$$

and LCC is defined as:

$$c_n^m = \mathbf{r}_n(m), \tag{13}$$

where $\mathbf{r}_n(m)$ is the reconstruction coefficient of feature descriptor $\mathbf{x}_n$ for the codeword $\mathbf{b}_m$.

4.8 Locality-Constrained Linear Coding (LLC)

The computational cost of LCC and SPC is high because their solution relies on iterative optimization. To address this problem, LLC [43] changed the term $|\mathbf{r}_n(m)|$ in (12) into a differentiable term $\mathbf{r}_n(m)^2$ (as in (14)) to yield an optimization problem that has a closed-form solution. LLC optimizes the following l_2 regularized problem:

$$\arg\min_{\mathbf{r}_n} \|\mathbf{x}_n - \mathbf{B}\mathbf{r}_n\|_2^2 + \lambda \sum_{m=1}^{M} (\mathbf{r}_n(m) \exp(\|\mathbf{x}_n - \mathbf{b}_m\|_2)/\sigma)^2 \tag{14}$$

$$\text{s.t } \mathbf{1}^{\mathrm{T}}\mathbf{r}_n = 1,$$

where σ is used for adjusting the weighted decay speed for the locality function (exponential term). The coding vector $\mathbf{r}_n$ corresponding to the feature descriptor $\mathbf{x}_n$ is given by:

$$\mathbf{r}_n = \tilde{\mathbf{r}}_n / \mathbf{1}^{\mathrm{T}}\tilde{\mathbf{r}}_n, \tag{15}$$

$$\tilde{\mathbf{r}}_n = (\mathbf{D}_n + \lambda\mathbf{S}_n) \setminus \mathbf{1},$$

$$\mathbf{D}_n = (\mathbf{B} - \mathbf{x}_n\mathbf{1}^{\mathrm{T}})(\mathbf{B} - \mathbf{x}_n\mathbf{1}^{\mathrm{T}})^{\mathrm{T}}$$

$$\mathbf{S}_n = daig(\mathbf{1} \odot trace(\mathbf{D}_n)),$$

where $\odot$ denotes element-wise multiplication. LLC is defined as:

$$c_n^m = \mathbf{r}_n(m) \tag{16}$$

To further enhance the encoding speed, approximated LLC was proposed. The second term in the above optimization problem (14) was omitted, and a feature is reconstructed just from the closest K codewords. Let $\mathbf{B}_n = [\,\hat{\mathbf{b}}_k]_{k=1}^{K}$ be the closest codewords to the feature descriptor $\mathbf{x}_n$ and $\hat{\mathbf{r}}_n$ be their corresponding coefficient codes, then the approximated LLC reduces the above optimization problem to:

$$\arg\min_{\hat{\mathbf{r}}_n} \|\mathbf{x}_n - \mathbf{B}_n\hat{\mathbf{r}}_n\|_2^2 \tag{17}$$

$$\text{s.t } \mathbf{1}^{\mathrm{T}}\hat{\mathbf{r}}_n = 1,$$

Approximated LLC is defined as:

$$c_n^m = \begin{cases} \hat{\mathbf{r}}_n(k), & if\ \mathbf{b}_m = \hat{\mathbf{b}}_{\boldsymbol{k}} \\ 0, & otherwise \end{cases} \quad , k = 1, \dots, K \tag{18}$$

where $\hat{\mathbf{r}}_n(k)$ is the reconstruction coefficient of the feature descriptor $\mathbf{x}_n$ for the codeword $\hat{\mathbf{b}}_{\boldsymbol{k}}$.

4.9 Salient Coding (SC)

SC considers a representation for each codeword that is salient when combined with max pooling. The idea behind SC [44] is that a codeword should receive a strong response if it is much similar with a feature than other codewords. That is, if a codeword is much closer to a feature belonging to this codeword than the other K closest codewords, then the response on this codeword is strong. This response is likely to be preserved during max pooling. As a result, the codeword can independently describe this feature without the help of other codewords. SC is defined as:

$$c_n^m = \begin{cases} \psi(\mathbf{x}_n), & if\ m = \operatorname{argmin}_i(\|\mathbf{x}_n - \mathbf{b}_i\|_2) \\ 0, & otherwise \end{cases} \quad , i = 1, \dots, M \tag{19}$$

$$\psi(\mathbf{x}_n) = \textstyle\sum_{k=2}^{K} \left(\left\|\mathbf{x}_n - \hat{\mathbf{b}}_k\right\|_2 - \left\|\mathbf{x}_n - \hat{\mathbf{b}}_1\right\|_2\right) / \left\|\mathbf{x}_n - \hat{\mathbf{b}}_k\right\|_2,$$

where $\psi(\mathbf{x}_n)$ denotes the saliency degree, and $[\hat{\mathbf{b}}_k]_{k=1}^{K}$ are the K closest codewords to the descriptor $\mathbf{x}_n$.

4.10 Group Salient Coding (GSC)

In SC, weak responses to a codeword are suppressed by the response of the feature that is closest to this codeword compared to others. This results in some features not represented. To alleviate the suppression effect, GSC [45] proposed an idea that is based on group coding. Different group sizes can be formed with a feature $\mathbf{x}_n$. In a group of size k, we only consider the feature $\mathbf{x}_n$ and the k nearest codewords. A saliency response for $\mathbf{x}_n$ is calculated for each group size and the response is shared among all the codewords in the group. Each feature $\mathbf{x}_n$ will have multiple coding vectors, one coding vector for each group size. The final coding vector is obtained by taking the maximum of the responses for each codeword over all group sizes. GSC is defined as:

$$c_n^m = \begin{cases} \max\limits_k \{\mathbf{v}_m^k\}, & if\ \mathbf{b_m} \in g(\mathbf{x}_n, k) \\ 0, & otherwise \quad k = 1, \dots, K \end{cases} \tag{20}$$

$$\mathbf{v}_m^k = \begin{cases} \psi(\mathbf{x}_n), & if\ \mathbf{b}_m \in g(\mathbf{x}_n, k) \\ 0, & otherwise, \end{cases}$$

$$\psi(\mathbf{x}_n) = \sum_{j=1}^{K+1-k}\left(\left\|\mathbf{x}_n - \hat{\mathbf{b}}_{k+j}\right\|_2 - \left\|\mathbf{x}_n - \hat{\mathbf{b}}_k\right\|_2\right),$$

where $g(\mathbf{x}_n, k)$ denotes the set of the k nearest codewords to $\mathbf{x}_n$, and K is the maximum group size.

4.11 Local Tangent Coding (LTC)

LTC [46] assumes that the codewords and feature descriptors are embedded in a smooth manifold. The main components of LTC are manifold approximation and intrinsic dimensionality estimation. Under the Lipschitz smooth condition, the nonlinear function $f(\mathbf{x}_n)$ can be approximated by a local linear function as:

$$f(\mathbf{x}_n) \approx \sum_{m=1}^{M}(\mu_m^n f(\mathbf{b}_m) + 0.5\mu_m^n \nabla f(\mathbf{b}_m)^{\mathrm{T}}(\mathbf{x}_n - \mathbf{b}_m)), \tag{21}$$

where μ_m^n is a scalar coefficient associated with codeword $\mathbf{b}_m$ that is used for representing the feature $\mathbf{x}_n$, and is obtained by LCC. The above approximate function (21) can be viewed as a linear function of a coding vector $[\mu_m^n, \mu_m^n(\mathbf{x}_n - \mathbf{b}_m)^{\mathrm{T}}]_{m=1}^{M} \in \mathbb{R}^{M(D+1)}$. LTC argues that there is a lower intrinsic dimensionality in the feature manifold. To get it, PCA is applied to the high dimensional term $\mu_m^n(\mathbf{x}_n - \mathbf{b}_m)$ using a projection matrix $\mathbf{U}_m = [\mathbf{u}_1, \dots, \mathbf{u}_H] \in \mathbb{R}^{D \times H}$ trained from the training data $(H < D)$. Theses directions correspond to the local tangent directions of the manifold. Therefore, the final coding vector for LTC is defined as:

$$\mathbf{c}_n^m = [\alpha\mu_m^n, \mu_m^n(\mathbf{x}_n - \mathbf{b}_m)^{\mathrm{T}}\mathbf{U}_m], \tag{22}$$

where α is a positive scaling factor to balance the two types of codes.

4.12 Super Vector Coding (SVC)

SVC [47] is a simple version of LTC. Unlike LTC, SVC uses just the closest codeword $\mathbf{b}_*$ to represent $\mathbf{x}_n$ and obtains μ_*^n via HA, i.e., $\mu_*^n = 1$. Moreover, SVC does not apply PCA to the term $\mu_m^n(\mathbf{x}_n - \mathbf{b}_m)$ in (21). In SVC, equation (21) simplifies to:

$$f(\mathbf{x}_n) \approx \left(f(\mathbf{b}_*) + 0.5\nabla f(\mathbf{b}_*)^{\mathrm{T}}(\mathbf{x}_n - \mathbf{b}_*)\right), \tag{23}$$

consequently, the coding vector simplifies to:

$$\mathbf{c}_n^m = \begin{cases} [\alpha, (\mathbf{x}_n - \mathbf{b}_m)^{\mathrm{T}}], & if\ m = \operatorname{argmin}_i(\|\mathbf{x}_n - \mathbf{b}_i\|_2) \\ 0, & otherwise \quad i = 1, \dots, M \end{cases} \tag{24}$$

5. DEEP LEARNING

5.1 Review

Deep Learning is about learning a task in a hierarchy of layers. The first layer learns from the input and succeeding layers learn from previous ones. Multilayer Perceptron (MLP), proposed in the past, is an architecture that can be built to go deeper but due to many problems it did not do. First of all, backpropagation [48] does not work well for deep MLPs. This is mainly due to the type of nonlinearities used. The traditional sigmoid and hyperbolic tangent activation functions kill the flowing gradients when neurons are saturated and hence layers down in the hierarchy do not receive theses gradients in order to update their weights. Moreover, proper initialization of weights is critical for backpropagation to work well. Improper initialization might cause most neurons down in the hierarchy to output zero as their activation which too kills the gradients from flowing. Second, the introduction of the Universal Approximation theorem [49] – which states that a single hidden layer can be used to approximate any continuous function to any desired precision – diverted people from going deeper with this model. Third, deep models are data hungry, and need much labeled data to train which was very difficult to get at that time. Lastly, deep models need much computation to train because most of the training is matrix multiplication and CPUs were not optimized for this operation.

Recently, the introduction of new activation functions [50-52], better initialization strategies [51, 53, 54], big data like ImageNet [55], better optimization methods like Adam optimization [56], better regularization methods [57-59], and finally the introduction of GPUs, all made deep model training possible.

Deep learning has changed the view of how problems are solved. In end-to-end deep learning, the input to the model is the signal itself, not features extracted from that signal. This produces a unified framework for the two steps that were considered separate for a long time: feature extraction and classification. Learning the features along with the classification boundary in one step produces more discriminative features far better than hand engineered ones and gives much better classification results.

5.2 Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN) is a powerful machine learning technique from the field of deep learning. CNNs were inspired by the experiments of Hubel and Weisel [60-62] in which they discovered that different neurons in the cat visual cortex fire for different light edge orientations. They also showed that locality is preserved in processing, i.e., nearby cells in the cortex represent nearby regions in visual field. They hypothesized through their experiments that the visual cortex has a kind of hierarchical organization where simple cells feed to other complex cells which in turn feed to hypercomplex cells. Fukushima [63] introduced the neurocognitron which models the visual cortex in computers. It is composed of a hierarchy of layers where each neuron in upstream layers looks at a small region of input in the downstream layer. Since backpropagation was not invented yet, Fukushima trained his model with an unsupervised procedure. LeCun et al. [64] built on Fukushima's work and trained his network (LeNet-5) with backpropagation. Many years later, Hinton and Salakhutdinov [65] described an effective way of initializing the weights that allows deep autoencoder networks to learn a low-dimensional representation of data. Krizhevsky et al. [66] designed a convolutional neural network that they called AlexNet. It is very similar to LeNet except that it is bigger, deeper, uses Rectified Linear Unit (ReLU) activation function, and is trained on GPUs with more training data (ImageNet). This network has won the ILSVRC2012 competition and reduced the error rate by a large margin. Since then, these models have drawn people attention and many deeper and wider CNNs were proposed. Common CNN examples include VGG [67], GoogleNet [68], ResNet [69, 70], DenseNet [71], etc. These networks have millions of parameters and need proper training.

CNNs are a special kind of multi-layer neural networks which have two fundamental differences: (1) neurons share weights, where many neurons in a layer share the same weights (2) they have sparse connections, i.e., not every neuron in upstream layer is connected to every other neuron in downstream layer. CNNs are composed of many layers of different types like convolutional layers, pooling layers, FC layers, softmax and classification layers. A convolutional layer is where downstream neurons are convolved (dot product operation) with a filter bank to give rise to many output feature maps, one map for each filter. Convolution filters need not to be flipped and this operation is similar to correlation. After filtering, a non-linearity such as ReLU is applied on the filtered input. It thresholds negative activations and passes positive activations as they are. A pooling layer down-samples each feature map by computing some statistic over it. Common statistics include average and the popular max pooling. Other pooling methods were recently proposed like fractional max-pooling [72] and spatial pyramid pooling [73]. At the end of the network, there exists the FC layers which are like the traditional MLP and it runs for a few layers before the softmax layer. The softmax layer uses the softmax function to convert class scores into class probabilities. The classification layer takes the class probabilities to compute the cross-entropy loss function to be optimized. Other types of layers that prevents the network from overfitting the training data were proposed like batch normalization [74] layer (which also speeds up the training process) and dropout layer [59].

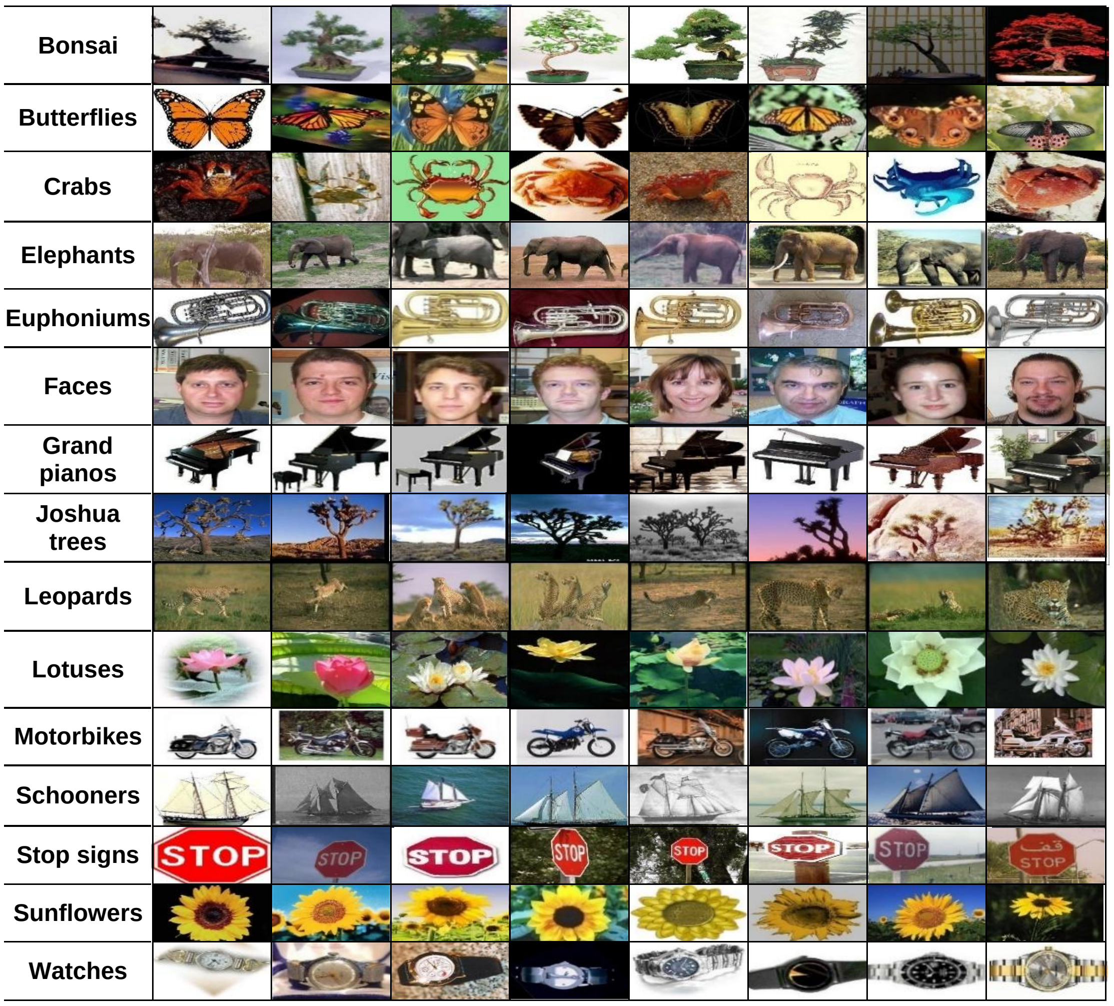


FIGURE 3: Example Images from The Image Dataset.

6. EXPERIMENTS

In this section, we describe four types of experiments carried out to evaluate the performance of shallow and deep image representations. The first experiment compares the performance of shallow image representations taking into consideration the enhancements proposed in the feature enhancements and normalization pipeline steps of the BoVW model for different encoding methods. Moreover, we contrast the results obtained to previous results obtained without using the proposed enhancements. The second experiment compares the performance of deep image representations on different CNN architectures finetuned on the training set. The third experiment evaluates the performance of handcrafted representations extracted from the BoVW pipeline and representations extracted from the CNN. The last experiment improves the performance by combining multiple CNNs into an ensemble.

The dataset used in the experiments consists of 2533 images each belonging to one of 15 object categories. These categories were taken from Caltech-101 dataset and include bonsais, butterflies, crabs, elephants, euphoniums, faces, grand pianos, Joshua trees, leopards, lotuses, motorbikes, schooners, stop signs, sunflowers, and watches. Each category contains different number of images and are of different resolutions. Example images from each category are shown in figure 3.

The dataset was split into two sets: training and testing sets. The first 30 examples of each class were used as a training set for a total of 450 images while the rest was used as a testing set for a total of 2083 images.

To demonstrate the results, we used MATLAB and two external libraries: VLFeat [75] and Lib-linear [76]. The results were reported using top 1 and top 2 accuracy evaluation metrics. Precisions for each category were also reported.

6.1 BoVW Pipeline Enhancements Experiments

In these experiments, we evaluate the performance of shallow representations for the enhancements proposed for different encoding methods. Furthermore, we compare these results to previous results that do not use the proposed enhancements.

Local Feature extraction and enhancement. All images were converted to grayscale even when color is available. Since all images have a medium resolution 400×500, images were processed as they are without resizing them. We only used a single descriptor type, i.e., SIFT descriptors extracted densely from an image with a stride of 4 pixels at 7 different scales with $\sqrt{2}$ scale increments. The spatial bin size of the SIFT descriptor was fixed to cover 8 pixels. The feature descriptors were enhanced by running them through the feature enhancement pipeline step where they were square-rooted as suggested in [4], reduced in dimensionality from 128 to 80 dimensions using PCA, and then whitened. Moreover, the reduced descriptors were augmented with their normalized spatial location to get 82-dimensional descriptors. For performance reasons, we didn’t use feature augmentation for FV.

Codebook Generation. A codebook of size 1024 codewords was constructed by clustering a random subset of feature descriptors from the training set using k-means clustering algorithm. The codebook was fixed in size (to 1024) for all experiments except for FV, and VLAD encoding methods which use a codebook of size 256 codewords. For VLAD, the codebook was constructed using k-means, whereas a GMM was used for FV. For GMM training, the mixture parameters were learnt from the training set using Expectation Maximization (EM, [29]) algorithm. The means of the GMM were initialized by the means of k-means algorithm run for a few iterations. The diagonal of the covariance matrix of each Gaussian was initialized by the diagonal of the covariance matrix of the points assigned to initial mean of each Gaussian. The mixing coefficients of each Gaussian were initialized by the proportion of points assigned to each Gaussian.

Encoding Methods. We compare the proposed pipeline enhancements to seven popular encoding methods used in the codebook model and for each we provide the parameters used in the encoding process:

- HA: It uses hard quantization and requires computing the nearest neighbor for each feature descriptor.
- LSA: It uses soft quantization and requires the computation of the k nearest neighbors to each feature descriptor. The parameter k was set to 5, and β was set to 2×10^{-4}.
- FV: It represents feature descriptors by their first and second order statistics for each codeword.
- VLAD: It represents each descriptor by the vector difference between the descriptor and its closest codeword. It requires the computation of the nearest neighbor.
- LLC: The approximated version of LCC was used to achieve both sparsity and locality. The parameter k was set to 5, and λ was set to 10^{-4}.
- SC: It requires the computation of the k nearest neighbors for each descriptor. Each descriptor is encoded just by its closest codeword as in HA. Unlike HA, the coding weight is not constant and depends on the distances of the descriptor from its k nearest codewords. The parameter k was set to 5.
- GSC: It is a soft version of SC and requires the computation of the k nearest neighbors for each descriptor. The parameter k was set to 5.

Spatial information. To introduce weak geometry to the representation, spatial pyramid was used for all experiments. It divides the image into three levels: 1×1, 2×2, and 4×4 for a total of 21 regions.

Pooling and Normalization. Different pooling and normalization strategies were used for different encoding methods. Region encodings are pooled separately and normalization is done for all regions. The following pooling and normalization strategies were used for each encoding method:

- HA: The resulting coding vectors from different feature descriptors are sum-pooled, sign-square-root normalized, globally l_2-normalized, and again sign-square-root normalized. The final representation has $1024 \times 21 = 21{,}504$ dimensions.
- LSA: Same as in HA.
- FV: Average-pooling is used for image coding vectors to get the pooling vector. Each gradient sub-vector of the pooling vector is l_2-normalized (intra-normalization), and each region is also l_2-normalized. After that, sign-square-root normalization followed by global l_2-normalization is done on the whole representation. The dimensionality of the representation is $82 \times 2 \times 256 \times 21 = 881{,}664$.

Encoding Methods	Top1 Accuracy (%)	Top2 Accuracy (%)
HA	96.35	98.94
LSA	96.74	98.90
FV	95.58	98.08
VLAD	96.69	98.80
LLC	97.12	99.09
SC	96.50	98.90
GSC	97.22	98.99

(a)

Encoding Methods	Top1 Accuracy (%)	Top2 Accuracy (%)
HA	91.60	96.45
LSA	93.95	98.03
FV	95.58	98.08
VLAD	94.38	98.22
LLC	93.61	98.03
SC	93.37	97.65
GSC	92.80	96.69

(b)

TABLE 2: Top 1 and top 2 accuracies of different encoding methods. (a) Using our proposed enhancements in the feature extraction and normalization steps of the recognition pipeline. (b) Previous methods implemented as in their original papers including enhancements introduced afterwards.

- VLAD: Coding vectors corresponding to image feature descriptors are sum-pooled, and the resulting pooling vector is sign-square-root normalized, and l_2-intra-normalized. The representation has $82 \times 256 \times 21 = 440{,}832$ dimensions.
- LLC: Max-pooling is used followed by global l_2 and sign-square-root normalization. The representation has $1024 \times 21 = 21{,}504$ dimensions.
- SC: Max-pooling followed by same normalization as in HA. The representation has $1024 \times 21 = 21{,}504$ dimensions.
- GSC: Max-pooling followed by the same normalization as in HA. The representation has $1024 \times 21 = 21{,}504$ dimensions.

Classifier. In all experiments, multi-class classification is done with 15 SVM classifiers trained using one-versus-all setting. We did not use nonlinear kernels or any explicit feature mapping. Only linear kernels were used. The SVM regularization parameter was set to 10.

Parameters used for previous results. We compared our enhancements to previous settings in the recognition pipeline. To achieve a fair comparison, we have tested the previous encoding methods with the parameters used in their original papers or with enhancements introduced afterwards. As before, we extracted SIFT descriptors with the same scales and sampling frequency. We do not perform square rooting for features except for FV. Moreover, features are neither augmented nor reduced in dimensionality (except for FV and VLAD which were reduced to 80 dimensions). The same pooling strategies and pyramid levels were used for all encoding methods as before. For normalization, we used l_2-normalization for individual regions and global l_2-normalization for the whole representation. Moreover, we used l_2-intra-normalization for encoding methods that produce a block of codes for each codeword, i.e., FV and VLAD. For classification, we used the SVM classifier as before. Intersection kernel was used for HA and LSA, Hellinger kernel for FV and VLAD, while a linear kernel was used for the rest of encoding methods. For nonlinear kernels, we used explicit feature mapping using additive homogeneous kernels.

Results and analysis. Results for the enhancements proposed are summarized in table 2(a). Different settings for spatial information, feature reduction, and pooling and normalization parameters were tested for each experiment. A performance increase was noticed when we introduce the feature enhancement step in the recognition pipeline and a sequence of representation normalization that is based on l_2- and sign-square-root normalization as described above. The performance increase was attained using the efficient and inexpensive linear kernel which yield same or better performance than nonlinear kernels using the above parameter settings. The results show that the relatively fast and sparse methods like GSC, LLC, HA, LSA, and SC yield better results than the slow and dense FV method. For this dataset, we see that among single-dimensional response hard coding methods, SC gives better results than HA. Moreover, the multi-dimensional response hard coding method, VLAD, gives better performance than single-dimensional response hard coding methods like HA and SC. Among sparse soft encoding methods, we see that GSC and LLC produce better result than LSA. Among multi-dimensional response coding methods, we see that VLAD produces better results than FV. Among all methods, we see that GSC produces the best result. Moreover, LLC and GSC produce the best results in terms of top 2 accuracy. Class precisions for different encoding methods using the proposed enhancements are shown in table 4(a).

Results for previous encoding methods are shown in table 2(b). We see that the FV outperforms other methods. Despite its best accuracy, FV is the slowest among all encoding methods since it needs more computation and memory resources for its multi-dimensional representation. However, we have achieved the best result using the faster and more efficient GSC encoding method using the proposed enhancements. Moreover, we see a large performance gap favoring our proposed enhancements over the previous methods for all encoding methods. Class precisions for previous results are given in table 4(b).

6.2 CNN Architectures Experiments

In these experiments, we finetune different pretrained deep CNN architectures on our dataset. Fine-tuning a network with transfer learning is usually much faster and easier than training a network with randomly initialized weights from scratch, and this might be necessary in case you don’t have enough training data. The early layers of a pretrained CNN learn low-level features (blobs, corners, edges) that are general to any task, and the last layers learn features that are task specific. Therefore, early layers were retained with their weights, and last layers were replaced with new layers of our own to learn features that are specific to our dataset.

CNN Architectures. Different CNNs expect different image sizes, so images were resized to fit the input layer of each network. Moreover, since the dataset contains a mixture of grayscale and RGB images, color preprocessing is important to ensure that all the images have the same number of channels required by the network. In the experiments, all images were converted to RGB. in the experiments, four popular and recent CNNs were used:

- AlexNet: It was named after Alex Krizhevsky [66] and has won the ILSVRC2012 competition on object classification and considered the first CNN-based winner. The network consists of 8 layers, 5 convolutional and 3 FC layers and uses 11×11, 5×5, and 3×3 convolutions. The network has around 60 million parameters and expects input of size 227×227 RGB image.

- VGG19: It was designed by Simonyan and Zisserman [67] and achieved the second rank in ILSVRC2014 object classification and the first in object localization. It consists of 19 layers, 16 convolutional and 3 FC layers. The network uses only 3×3 convolutions but lots of filters and is deeper than AlexNet. It contains around 138 million parameters and expects inputs of size 224×224 RGB image.

- GoogLeNet: It was designed by Google and has won the ILSVRC2014 competition on object classification. It is deeper than VGG19 and consists of 22 layers. It features 9 efficient inception modules where each module uses 5×5, 3×3, and 1×1 convolutions. It does not contain FC layers and contains only 5 million parameters. The network expects inputs of size 224×224 RGB image.

- ResNet: It was designed by Microsoft and has won the ILSVRC2015 competition on object classification. It is much deeper and consists of 152 layers. To improve the training process, it introduced a novel architecture with “skip connections” and featured heavy batch normalization.

CNN	Top 1 Accuracy (%)	Top 2 Accuracy (%)
AlexNet	97.95 ± 0.34	99.30 ± 0.19
VGG19	98.62 ± 0.11	99.57 ± 0.14
ResNet50	98.65 ± 0.14	99.49 ± 0.11
GoogleNet	98.75 ± 0.17	99.55 ± 0.07

(a)

Experiment	Top 1 Accuracy (%)	Top 2 Accuracy (%)
Deep features on linear SVM	99.04	99.38
Ensemble of 5 CNNs	99.47	99.52

(b)

TABLE 3: (a) Top 1 and top 2 accuracies for different deep network architectures. (b) Top row: deeply learnt features extracted from a CNN and trained on an SVM. Bottom row: ensemble of 5 CNN learners combining their predictions by taking the maximum posterior probability.

It expects inputs of size 224×224 RGB image. In the experiment, we have used only 50 layers (ResNet50).

Training Parameters. The last FC, softmax, and classification layers were removed from each network and new FC, softmax, and classification layers were added. The weights of the CNN initial layers were frozen while were allowed to update in last layers. The unfrozen layers were trained on the training set using Stochastic Gradient Descent with Momentum (SGDM) for a maximum of 10 epochs. The momentum was set to 0.9 and the effective learning rate was set to 10^{-3}. The l_2-regularization constant for weights of the FC layer was set to 10^{-4} with zero regularization for biases. Since these networks contain some stochastic layers, the networks were trained and tested for 5 times and the average accuracies and standard deviations were reported.

Results. Results for different CNN architectures are shown in table 3(a). They are better than previous results for shallow representation encoding methods. Some CNNs give better results than others on the dataset. For example, GoogLeNet gives the best result among all and is fast to train and test. Second comes Resnet50 which gives comparable result and is also fast to train and test. VGG19 gives good performance but is slow in training and testing. Finally comes AlexNet which gives relatively lower result but is fast at training and testing. Precisions for each class are given in table 4(c).

6.3 Shallow vs. Deep Representations Experiment

In this experiment, we compare between hand-engineered shallow features produced using the previous encoding methods and deeply learnt features extracted from a CNN. Here, we treat a CNN as a generalized feature extractor, and activations of the last newly added fully connected layer were used as the image representation. These representations were then fed to a linear SVM for classification. Same parameters for SVM were used as before.

Results and analysis. Results for this experiment are shown in table 3(b). Despite the low dimensionality of deep features (15 dimensions in our case), they produce better results than the high dimensional (tens of thousands) handcrafted features. Moreover, the performance gain is achieved without using any normalization method or applying nonlinear kernels. Class precisions are given in table 4(d).

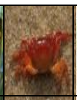
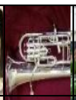
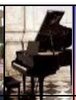
	Class / Expr.															
(a)	HA	93.88	86.89	79.07	97.06	91.18	99.75	98.55	97.06	100	86.11	97.01	100	94.12	94.55	93.78
	LSA	92.86	88.52	83.72	94.12	97.06	99.51	92.75	97.06	100	91.67	98.31	100	94.12	89.09	94.26
	SC	90.82	88.52	86.05	97.06	94.12	99.75	92.75	97.06	100	88.89	98.44	100	94.12	87.27	92.34
	GSC	92.86	88.52	83.72	97.06	97.06	100	94.20	97.06	100	91.67	98.83	100	94.12	89.09	95.22
	LLC	92.86	86.89	86.05	97.06	97.06	99.75	94.20	97.06	100	91.67	98.96	100	94.12	87.27	94.74
	VLAD	94.90	83.61	88.37	100	94.12	99.75	98.55	97.06	100	91.67	96.35	100	94.12	94.55	96.17
	FV	84.69	81.97	79.07	94.12	91.18	99.51	97.10	100.00	100	77.78	98.70	96.97	85.29	87.27	91.87
(b)	HA	94.90	85.25	83.72	94.12	94.12	99.26	98.55	91.18	100	86.11	84.90	96.97	94.12	94.55	94.90
	LSA	94.90	86.89	83.72	94.12	94.12	99.51	97.10	91.18	99.41	83.33	91.80	96.97	91.18	92.73	91.87
	SC	84.69	83.61	79.07	91.18	91.18	99.75	95.65	91.18	100	80.56	92.84	100	94.12	83.64	91.39
	GSC	90.82	85.25	79.07	91.18	91.18	99.51	95.65	91.18	99.41	86.11	88.93	100	97.06	94.55	93.30
	LLC	95.92	86.89	79.07	94.12	94.12	99.51	97.10	91.18	100	88.89	90.36	100	94.12	90.91	92.34
	VLAD	93.88	80.33	83.72	97.06	94.12	98.77	100	97.06	99.41	91.67	91.67	100	94.12	92.73	95.69
	FV	84.69	81.97	79.07	94.12	91.18	99.51	97.10	100	100	77.78	98.70	96.97	85.29	84.69	81.97
(c)	AlexNet	96.53	85.90	97.67	97.06	92.94	99.41	100	98.24	97.41	90.56	99.17	100	100	100	95.60
	VGG19	98.16	92.13	97.21	98.24	95.88	99.21	100	95.29	98.82	87.22	99.53	99.39	100	97.82	98.76
	ResNet50	97.96	92.79	95.81	97.65	96.47	99.26	100	96.47	95.41	93.89	99.77	100	97.96	92.79	95.81
	GoogLeNet	95.92	94.43	93.49	99.41	91.18	99.60	100	92.94	99.06	91.11	99.87	100	100	97.45	99.33
(d)	Deep features	96.12	87.87	96.74	98.24	94.71	99.51	100	98.24	98.00	89.44	99.27	100	100	100	96.17
(e)	CNN ensemble	97.96	96.72	97.67	100	97.06	100	100	100	98.24	94.44	100	100	100	100	100

TABLE 4: Class precisions for different methods. (a) Encoding methods with enhancements in the recognition pipeline. (b) Encoding methods with previous settings without the proposed enhancements. (c) Average precision for deep network architectures trained and tested five times. (d) Deeply learnt features extracted from a CNN and classified on an SVM. (e) Ensemble of 5 CNN architectures.

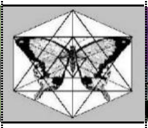
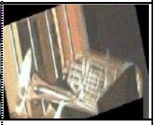
6.4 CNN Ensemble Experiment

In this experiment we combine multiple deep CNN learners into an ensemble using maximum posterior probability. The CNN learners are trained as before and are of different architectures. We achieved the best result using only five learners.

Results. The ensemble of deep network learners gives the best results (table 3(b)) among all experiments since it selects the most probable class among all the participating learners. Class precisions are given in table 4(e) and misclassified images along with their predictive probability are shown in figure 4.

7. CONCLUSION

In this study, we have reviewed the different steps used in the BoVW recognition pipeline used to extract shallow image representations. Moreover, we have provided a concrete mathematical framework for the different encoding methods used in the pipeline. Several enhancements in the local feature extraction and normalization pipeline steps were introduced that give better image representations and perform well than previous methods. We have achieved a top 1 accuracy of 97.22% for shallow representations using the GSC encoding method which outperforms the best encoding method (FV) in the previous results having a top 1 accuracy of $95.58\,\%$.

Actual Class	bonsai	bonsai	butterfly	butterfly	crab	euponium
Image						
Output class	joshua tree	joshua tree	lotus	schooner	lotus	grand piano
Posterior prob.	97.27%	58.80%	69.78%	76.86%	89.69%	47.74%

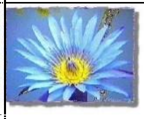
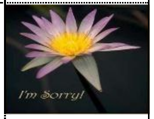
Actual Class	leopard	leopard	leopard	lotus	lotus
Image					
Output class	joshua tree	joshua tree	joshua tree	sunflower	sunflower
Posterior prob.	82.52%	88.81%	90.17%	67.30%	99.13%

FIGURE 4: Misclassified images for the 5 CNN ensemble experiment. Image actual class, predicted class and the posterior probability are shown above.

Instead of handcrafting the image representations, we have demonstrated that better representations can be extracted by learning them directly from the input image while training the CNN deep learning model. We have achieved a top 1 accuracy of $98.75 \pm 0.17\%$ using GoogleNet CNN and boosted the results to 99.47% using an ensemble of only 5 CNNs. As have been seen, the deeply learnt representations are concise, more discriminative, and outperform shallow representations.

8. FUTURE WORK

There are many parameters in the recognition pipeline that need to be further explored. These include: different types of feature descriptors, sampling strategies, clustering algorithms, codebook size, encoding methods, pooling methods, normalization methods, and classification models. Each pipeline step might include other parameters that requires further tuning. For CNNs, different architectures need to be explored. Deeper and wider CNNs are the state-of-the-art architectures. Other CNN tunable parameters include: initialization strategies, pooling methods, optimization algorithms, batch normalization, nonlinearities and many others. Moreover, we want to extend these ideas to video data which has a temporal dimension in addition to the spatial dimensions. Instead of extracting spatial SIFT descriptors, we need other descriptors that take the time dimension into consideration like Histogram of Optical Flow (HOF). The rest of the clustering, encoding, pooling, and normalization methods need to be tested on video data. In the same manner, CNNs need to be changed in architecture to include the temporal dimension of video data. There exist many applications for video recognition such as human action recognition and gesture recognition.

9. REFERENCES

[1] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. "Visual categorization with bags of keypoints". *Workshop on statistical learning in computer vision, ECCV*, 2004.

[2] J. Sivic and A. Zisserman. "Video Google: A text retrieval approach to object matching in videos". IEEE International Conference on Computer Vision, 2003.

[3] S. Lazebnik, C. Schmid, and J. Ponce. "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories". IEEE Conference on Computer Vision and Pattern Recognition, 2006.

[4] R. Arandjelović and A. Zisserman. "Three things everyone should know to improve object retrieval". IEEE Conference on Computer Vision and Pattern Recognition, 2012.

[5] J. Sánchez, F. Perronnin, and T. De Campos. "Modeling the spatial layout of images beyond spatial pyramids". Pattern Recognition Letters, vol. 33, no. 16, pp. 2216-2223, 2012.

[6] D. Lowe. "Distinctive image features from scale-invariant keypoints". International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, 2004.

[7] H. Bay, T. Tuytelaars, and L. Van Gool. "Surf: Speeded up robust features". European Conference on Computer Vision, 2006.

[8] C. Harris and M. Stephens ". A combined corner and edge detector". Alvey Vision Conference, 1988.

[9] T. Lindeberg. "Feature detection with automatic scale selection," International Journal of Computer Vision, vol. 30, no. 2, pp. 79-116, 1998.

[10] K. Mikolajczyk and C. Schmid. "Indexing based on scale invariant interest points". IEEE International Conference on Computer Vision, 2001.

[11] P. Viola and M. J. Jones. "Robust real-time face detection". International Journal of Computer Vision, vol. 57, no. 2, pp. 137-154, 2004.

[12] E. Rosten and T. Drummond. "Fusing points and lines for high performance tracking". IEEE International Conference on Computer Vision, 2005.

[13] E. Rosten and T. Drummond. "Machine learning for high-speed corner detection". European Conference on Computer Vision, 2006.

[14] J. Matas, O. Chum, M. Urban, and T. Pajdla. "Robust wide baseline stereo from Extremal Maximally Stable regions.," Image and Vision Computing, vol. 22, no. 10, pp. 761-767, 2004.

[15] T. Lindeberg and J. Gårding, "Shape-adapted smoothing in estimation of 3-D shape cues from affine deformations of local 2-D brightness structure." Image and Vision Computing, vol. 15, no. 6, pp. 415-434, 1997.

[16] A. Baumberg. "Reliable feature matching across widely separated views." in IEEE Conference on Computer Vision and Pattern Recognition., 2000.

[17] F. Schaffalitzky and A. Zisserman. "Multi-view matching for unordered image sets, or “How do I organize my holiday snaps?”". European Conference on Computer Vision, 2002.

[18] K. Mikolajczyk and C. Schmid. "Scale & affine invariant interest point detectors". International Journal of Computer Vision, vol. 60, no. 1, pp. 63-86, 2004.

[19] K. Mikolajczyk and C. Schmid. "A performance evaluation of local descriptors". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 10, pp. 1615–1630, 2005.

[20] Y. Ke and R. Sukthankar. "PCA-SIFT: A more distinctive representation for local image descriptors". IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2004.

[21] S. Belongie, J. Malik, J. Puzicha, and M. Intelligence. "Shape matching and object recognition using shape contexts". IEEE Transactions on Pattern Analysis and Machine Intelligence, no. 4, pp. 509-522, 2002.

[22] J. Canny. "A computational approach to edge detection". IEEE Transactions on Pattern Analysis Machine Intelligence, vol. 6, no. 6, pp. 679-698, 1986.

[23] A. Johnson, M. Hebert. "Using spin images for efficient object recognition in cluttered 3D scenes". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 5, pp. 433-449, 1999.

[24] S. Lazebnik, C. Schmid, J. Ponce. "A sparse texture representation using local affine regions". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pp. 1265-1278, 2005.

[25] C. Schmid, R. Mohr. "Local grayvalue invariants for image retrieval". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 5, pp. 530-535, 1997.

[26] J. Flusser. "On the independence of rotation moment invariants". Pattern Recognition, vol. 33, no. 9, pp. 1405-1410, 2000.

[27] S. Lloyd. "Least squares quantization in PCM". IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129-137, 1982.

[28] M. Muja and D. G. Lowe. "Fast approximate nearest neighbors with automatic algorithm configuration". International Conference on Computer Vision Theory and Applications, VISAPP, vol. 2, 2009.

[29] G. McLachlan and D. Peel. Finite mixture models. John Wiley & Sons, 2004.

[30] R. Arandjelovic and A. Zisserman. "All about VLAD". IEEE Conference on Computer Vision and Pattern Recognition, 2013.

[31] K. Grauman and T. Darrell, "Pyramid match kernels: Discriminative classification with sets of image features," in IEEE International Conference on Computer Vision (ICCV), 2005.

[32] M. Marszalek, C. Schmid, H. Harzallah, and J. van de Weijer. "Learning representations for visual object class recognition". Proceedings of the PASCAL Visual Object Classes Challenge, 2007.

[33] A. Vedaldi, A. Zisserman. "Efficient additive kernels via explicit feature maps," vol. 34, no. 3, pp. 480-492, 2012.

[34] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. "Lost in quantization: Improving particular object retrieval in large scale image databases," IEEE conference on Computer Vision and Pattern Recognition, 2008.

[35] V. Gemert, J. Geusebroek, C. Veenman, and A. Smeulders, "Kernel codebooks for scene categorization," European Conference on Computer Vision, 2008.

[36] V. Gemert, J. Geusebroek, C. Veenman, and A. Smeulders. "Visual word ambiguity". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 7, pp. 1271-1283, 2010.

[37] L. Liu, L. Wang, and X. Liu. "In defense of soft-assignment coding". IEEE International Conference on Computer Vision, 2011.

[38] F. Perronnin, J. Sánchez, and T. Mensink. "Improving the fisher kernel for large-scale image classification". European Conference on Computer Vision, 2010, pp. 143-156.

[39] T. Jaakkola and D. Haussler. "Exploiting generative models in discriminative classifiers". Advances in Neural Information Processing Systems, 1999.

[40] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, C. Schmid, and P. Pérez. "Aggregating local descriptors into a compact image representation". IEEE Conference on Computer Vision & Pattern Recognition, 2010.

[41] J. Yang, K. Yu, Y. Gong, and T. Huang. "Linear spatial pyramid matching using sparse coding for image classification". IEEE Conference of Computer Vision and Pattern Recognition, 2009.

[42] K. Yu, T. Zhang, and Y. Gong. "Nonlinear learning using local coordinate coding". Advances in Neural Information Processing Systems, 2009.

[43] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. "Locality-constrained linear coding for image classification". IEEE Conference on Computer Vision and Pattern Recognition, 2010.

[44] Y. Huang, K. Huang, Y. Yu, and T. Tan. "Salient coding for image classification". IEEE Conference on Computer Vision and Pattern Recognition, 2011.

[45] Z. Wu, Y. Huang, L. Wang, and T. Tan. "Group encoding of local features in image classification". International Conference on Pattern Recognition, 2012.

[46] K. Yu and T. Zhang. "Improved Local Coordinate Coding using Local Tangents". International Conference of Machine Learning, 2010.

[47] X. Zhou, K. Yu, T. Zhang, and T. Huang. "Image classification using super-vector coding of local image descriptors". European Conference on Computer Vision, 2010.

[48] D. Rumelhart, G. Hinton, and R. Williams. " Learning representations by back-propagating errors". Nature, vol. 323, pp. 533-536, 1986.

[49] G. Cybenko. "Approximation by superpositions of a sigmoidal function". Mathematics of Control, Signals, and Systems, vol. 2, no. 4, pp. 303-314, 1989.

[50] V. Nair and G. Hinton. "Rectified linear units improve restricted boltzmann machines". International Conference on Machine Learning, 2010.

[51] K. He, X. Zhang, S. Ren, and J. Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". IEEE International Conference on Computer Vision, 2015.

[52] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". arXiv preprint arXiv: 1511.07289v5, 2016.

[53] S. Kumar. "On weight initialization in deep neural networks". arXiv preprint arXiv: 1704.08863, 2017.

[54] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". International Conference on Artificial Intelligence and Statistics, 2010.

[55] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database". IEEE Conference on Computer Vision and Pattern Recognition, 2009.

[56] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". arXiv preprint arXiv: 1412.6980, 2014.

[57] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect". International Conference on Machine Learning, 2013.

[58] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. "Deep networks with stochastic depth". European Conference on Computer Vision, 2016.

[59] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". arXiv preprint arXiv:.1207.0580, 2012.

[60] D. Hubel and T. Wiesel. "Receptive fields of single neurones in the cat's striate cortex". The Journal of Physiology, vol. 148, no. 3, pp. 574-591, 1959.

[61] D. Hubel and T. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". The Journal of Physiology, vol. 160, no. 1, pp. 106-154, 1962.

[62] D. Hubel and T. Wiesel. "Receptive fields and functional architecture of monkey striate cortex". The Journal of Physiology, vol. 195, no. 1, pp. 215-243, 1968.

[63] K. Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". Biological Cybernetics, vol. 36, no. 4, pp. 193-202, 1980.

[64] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". Proc. Of IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.

[65] G. Hinton and R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". Science, vol. 313, no. 5786, pp. 504-507, 2006.

[66] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks". Advances in Neural Information Processing Systems, 2012.

[67] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". arXiv preprint arXiv: 1409.1556, 2014.

[68] C. Szegedy et al. "Going deeper with convolutions". IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1-9.

[69] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[70] K. He, X. Zhang, S. Ren, and J. Sun. "Identity mappings in deep residual networks". European Conference on Computer Vision, 2016.

[71] G. Huang, Z. Liu, L. Van Der Maaten, and K. Weinberger. "Densely connected convolutional networks". IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[72] B. Graham. "Fractional max-pooling". arXiv preprint arXiv: 1412.6071, 2014.

[73] K. He, X. Zhang, S. Ren, and J. Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". European Conference on Computer Vision, 2014.

[74] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". arXiv preprint arXiv:.1502.03167, 2015.

[75] A. Vedaldi and B. Fulkerson. "VLFeat: An open and portable library of computer vision algorithms". ACM International Conference on Multimedia, 2010.

[76] “Lib-linear”. Internet: https://www.csie.ntu.edu.tw/~cjlin/liblinear/, [Jun. 19, 2018].